Table of Contents

UNIT – I
PHP

What is PHP?

PHP is an recursive acronym for "PHP: Hypertext Preprocessor"

PHP is a widely-used, open source (free to download, use, modify and rerelease) scripting language

PHP scripts are executed on the server. – server side scripting language

PHP History

- PHP was developed by Rasmus Lerdorf for monitoring his online resume in 1995. It slowly became popular. This first version PHP/F1 has some basic functions.
- PHP/F1 2.0 was released and was quickly replaced by PHP 3.0 in 1997
- PHP 3.0 was interesting. It was developed by and Andi Gutmans and Zeev suraski. It includes support for wider range of data bases (Oracle and My SQL). This version had greater portability. It was used by thousands of web servers.
- PHP 4.0 was released in 2003. It used a new engine and provided better performance, greater reliability and scalability and support for web servers other than Apache. It also supports OOP features and built-in session management.
- PHP 5.0 offers a complete object oriented program models. It also includes better exception handling, a more consistent XML tool kit, improved My SQL support and a better memory manager. **PHP 5.0 was the best** in the languages 10 year history. A survey was conducted by net craft in April 2004. It showed that PHP was in use for over 15 million websites.

Features of PHP

PHP as a programming language for the web has many advantages than JSP, ASP and VB.NET. Because of its simplicity and open source availability it is the widely used scripting language for websites. The features of PHP that are listed below makes PHP the most enjoyable language for the developer community.

Simplicity

Even a beginner can easily to learn PHP because it has

- Clearly written manual
- Consistent and logical syntax
- It may take only two hours for a programmer to learn PHP and write scripts.
- It has Kiss principle for (Keep it simple stupid) rapid application development.
- It can also access c library and take advantage of that code.

Portability

The PHP scripts can be made to work on different platformsfrom PHP 3.0 This is because PHP code is interpreted not compiled.The meaning of this portability is developer can write code on windows and deploy on Unix with interpreter for PHP. PHP scripts can be run on Microsoft windows, Unix, Mac OS and OS/2.

Speed

PHP scripts run faster that most scripting languages like ASP, JSP and Perl. when PHP 4.0 was released it had new parsing engine which made it faster. PHP 5.0 was even faster through the use of optimized memory manager and object handler.

Open source

Since PHP is open source, its source code is freely available on the web. We need not pay any license. So cost of developed website application is reduced but reliability and performance is not affected. This open-source approach also makes the application to fix bug easily.

Extensible

The language is set to be extensible, it can support new technology and add new methods to it. PHP create has built-in architecture which allows developers easily add new technologies and modules.

Example of add on modulus:-

- Dynamically create images
- PDF and SWF files
- Connect to IMAP and POP 3
- Interface with my SQL, oracle, postqre SQL & SQ lite
- Handle electronic payments
- Pass ML documents
- Execute perl, java, com.code

What is a PHP File?

PHP files can contain text, HTML, CSS, JavaScript, and PHP code

PHP codes are executed on the server, and the result is returned to the browser as plain HTML

PHP files have extension ".php"

What Can PHP Do?

- PHP can generate dynamic page content
- PHP can create, open, read, write, delete, and close files on the server
- PHP can collect form data
- PHP can send and receive cookies
- PHP can add, delete, and modify data in database
- PHP can be used to control user-access
- PHP can encrypt data
- PHP is not limited to output HTML. We can output images, PDF files, and even Flash movies. We can also output any text, such as XHTML and XML.

Why PHP?

PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
PHP is compatible with almost all servers used today (Apache, IIS, etc.)
PHP supports a wide range of databases
PHP is free. Download it from the official PHP resource: www.php.net
PHP is easy to learn and runs efficiently on the server side

What's new in PHP 7

PHP 7 is much faster than the previous popular stable release (PHP 5.6)
PHP 7 has improved Error Handling
PHP 7 supports stricter Type Declarations for function arguments
PHP 7 supports new operators (like the spaceship operator: <=>)

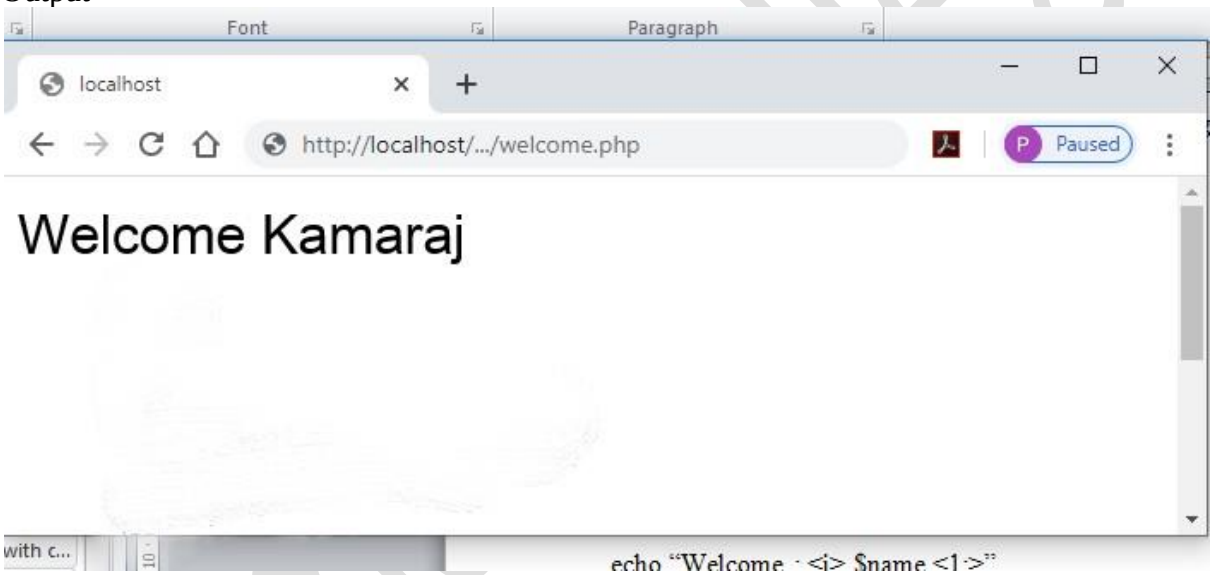
Running a PHP script

1. Start Windows
2. Click start –program files-Accessories-notepad
3. Start typing necessary – commends/coding
4. Click file – Save as
5. Type filename with 'PHP extension'
6. Click save button
7. Start PHP interpreter.
8. Run in browser

Welcome.php

```
<? PHP
$ name = "Kamaraj ";
echo "Welcome : <i> $name <1:>"
?>
```

Output



PHP statements and comments

A PHP script consists of one or more statements, with each statement ending with a semicolon. Blank lines within the script are ignored by the parser. Everything outside the tags is also ignored by the parser, and returned as it; only the code between the tags is read and executed. The semicolon can be omitted on the last line of a PHP block, because the closing ?> includes an implicit semicolon.

For greater readability, we should add comments to our PHP code.
There are three comment styles listed here:

```
// this is a single-line comment
# so is this
/* and this is a
multiline
comment */
```

Variables

Variables are the building blocks of any programming language. A variable can be thought of as a programming construct used to store both numeric and nonnumeric data. The contents of a variable can be altered during program execution, and variables can be compared and manipulated using operators.

PHP supports a number of different variable types—Booleans, integers, floating point numbers, strings, arrays, objects, resources, and NULLs—and the language can automatically determine variable type by the context in which it is being used. Variables need not be declared. Every variable has a name, which is preceded by a dollar ($) symbol, and it must begin with a letter or underscore character, optionally followed by more letters, numbers, and underscores. For example, $popeye, $one_day, and $INCOME are all valid PHP variable names, while $123 and $48hrs are invalid variable names.

Variable names in PHP are case-sensitive; $count is different from $Count or $COUNT.

Saving form input in variable

PHP can be used for creating dynamic websites. In a dynamic website, user can interact with website by providing text or by selecting options. Form is one of the quickest and easiest ways to add interactivity to our website. Through forms we can read user data into PHP variables.

Sample form

Home.html

```
< html>
< head></ head>
< body>
< farm action=" message . PHP" method  =" post">
Enter your message:< input type=" text" name=" MSG"=" 30">
< input type=" submit" value=" send ">
</ form>
</ body>
</ html>
```

Output

Message. php

```
<? PHP
// retrieve form data in a variable
$input=$- post[" MSG"];
// print it
echo " you said <i> $input </i>”
```

?>
Output
You said: Hello

We use two files (html and PHP files) for retrieving a single text data got from a user in a web form into php variable $input. The web page is displayed with the message **Enter yourmessage:** In the text box near the message, the user will type some message in few words after the text box there will be a submit button. After entering the message in the text box user will click submit bottom. When submit button is clicked the PHP script message.php is executed and the message displayed will be the text entered by the user.

Working

Here we have used HTML coding for displaying the input box. The <form> tag of html is used with input option. The form tag also contains two main attributes **action** and **method**. The action attribute specifies the name of the script. Here the name is message.php. The script will process the message entered into the form. The method attribute of the < form> tag specifies the manner in which form data will be submitted. Here it is POST method.

We have to run the home.html file in a browser. This is a client side script. On running this file a form is displayed in the webpage with a line.

When we click the submit button the server side script in the action attribute will be called for execution. That script will display another webpage that prints the string we have entered.The $_POST variable would have stored the string we have entered. It will be stored in$_POST['msg']. Then it will be assigned to php variable $input.We can display it using echo statement.

Constants:

A constant is used to store fixed values.We can declare and use constants using define() function.

The syntax is

define (" constant name", value);

Example:

define (" gold price",3800);
define (" PI",3.14)

We can use just like a variable. The only difference is the variable will start with $ sign and a constant doesn't have $ sign.The constant name can have small letters (lower case).

There are some predefined built-in PHP variables and constants:-
The function phpinfo() displays all the variables and constant.

There are 6 basic scope rules.

- Built-in super doper Global variables are visible everywhere within the script.
- Constants once declared is always visible globally.
- Global variables in a script are visible throughout the script but not inside function.
- Variables inside function that are declared Global refers to global variable of same name.
- Variables created inside function and declared as static are invisible from outside the function.
- Variable created inside function are local to the function and stops to exist when the function terminate.

Operators

PHP provides the set of operators for performing arithmetic, relational, logical and string operations.

Arithmetic operators:

Operator symbol	Meaning
+	Addition
_	Subtraction
*	Multiplication
/	Division
%	Division remainder
=	Assignment

Relational operator:

Operator Symbol	Meaning
<	Less than
<=	Less than equal to
>	Greater than
>=	Greater than equal to
= =	**equal to**
= = =	**Equal to and of same type**
! = =	**Not equal to or not of same type**
<> aka ! =	Not equal to

Logical operator:

Operator Symbol	Meaning
&&	Logical AND
||	Logical OR
XOR	Logical XOR
!	Logical not

Other operators:

Operator Symbol	Meaning
.	String contraction
.=	Concatenation and assignment
++	Auto increment
+=	Increment and assignment
--	Auto decrement
-=	Decrement and assignment

Exercise:-

Give the output of the following

$ n1= 5;
$ n2= 10;
$ n3=2;
$s1="good";
$s2="day";
a) $n1++i
b) $ n1-=n2
d) $ s1.$s2
e) $s1.=$s2
f) ($n1>$n2) && ($n2 < $n3)
g) $n1 + =$n2+$n3
h) $n2!=$n3
i) $n3=$n1%$n2
j) $n3= $n2/$n1

Conditional statements

- Conditional statement are statements that alters the sequence of execution based upon the condition provided on the statement.
- There are branching statements and looping statements in PHP.

If statement

The simplest form of conditional statement is if statement.
The general Syntax is

```
If (conditional)
{
// true do this;
}
```

Example:

```
If($ r<100)
    echo" very low";
```

The if else statement

```
if( conditional)
{
// true do this;
}
else
{
// false do this
}

if($r<100)
   echo" very low";
else
{
   echo" very high";
   echo" thank you";
}
```

else if

```
if conditional
{
// true do this;
}
else it
{
// for do this
}
else
{
// statement
}
```

Example:

```
If ($ lontry=="GK")
   echo "hondon";
else if ($ country ==" US")
{
   echo " washington";
}
:
:
else
{
   echo "unknown capital"
```

}
Write a PHP script for checking whether number is greater than 50 or not.

The ternary operator

The ternary operator is represented by? It provides shortcut Syntax for single statement if else block. The following code snippets show how it works.

```
<?php
if($ dial count>10)
{
  $MSG=" command after two attempt
}
else
{
  $MSG=" dialing.....";
}
```

The above code with if else can be replaced with ternary operator as follows.

```
<?php
$MSG=$ dial count>10? Cannot connect after 28 terms:" dialing.....";
?>
```

Nesting conditional statement

If PHP we can nest conditional statements each inside each other. For example we can use nested if as shown below.

```
<?php
if($country ==”India”)
{
  if ($state = =”Tamil Nadu”)
  {
     if($city = = “Thoothukudi”)
     {
        $home=true;
     }
   }
}
>?
```

Repeating action with loop

A loop is a control structure that enables us to provide repetition of set statements based upon set of conditionals. There are three loops in PHP.

- While loop
- Do loop
- For loop

Write a script for displaying multiplication table as shown

```
<?php
//define number and limits for multiplication tables
$num=1
$upperlimit=10;
$lowerlimit=12;
//loop and multiply to create table
```

```
do
{
echo"$num x$lowerlimit="($num*$lowerlimit);
$lowerlimit ++;
}
while ($lowerlimit <=$upperlimit)
?>
```

	_	☐	X
1x7=7 2x7=14 3x7=21 4x7=28 5x7=35			

Write a PHP script for printing ever number between any two limits.

```
<?
$lower limit=10;
$upper limit =50;
for ($ lower limit -1.2 1==0)
        $ lower limit++;
for($i=lower limit;$i=$upper limit;$i++)
  echo $i;
?>
```

```
<?
<form action=PHP-SELF method=post>
Enter lower limit:<input type="text"name="lower"size="30">
Enter upper limit:<input type="text"name="lower"size="30">
<input type="submit "value="send">
</form>
$lower limit=$_post ["Lower"];
$lower limit i$i <=$upper limit,Bi++)
{
echo $;
}
?>
```

	_	☐	X
Enter a lower limit:7 Enter a upper limit:14 Submit			

	_	☐	X
8 10 12 14			

Using the switch () statement

An alternative to the if… else () family of control structure is PHP switch- case() statement which does almost the same thing. A Switch () statement evaluates a conditional expression. Depending on the result an appropriate case() block is executed.

```
<?php
switch (conditional variable)
```

```
{
case possible result #1;
  Do this;
case possible result #2;
  Do this;
  ……..
case possible result #n;
  Do this;
case default;
  Do this;
}
?>
```

Here is a rewrite of last example using switch case();

```
<?php
switch($country)
{
case ‘UK’;
  $capital=’London’;
  break;
case ‘US’;
  $capital =’Washington’;
  break;
case ‘FR’;
   $capital=’peril’;
  default;
$capital=’Unknown;
  break;
}
?>
```

A couple of important keywords are here. That **break** keyword is used to break out of the switch () statement block and move immediately to the lines following it, while the **default** keyword is used to execute a default set of statement when the variable passed to switch () does not satisfy any of the conditions listed within the block.

UNIT – II
ARRAY

Array

Array is complex variable that enables us to store multiple values in a single variable. We use this to store related information. This value can have same name and similar action performing and them.

An array is defined as following.

```
<?php
// define array
$ fruits={' apple',' mango',' orange'}
$ fruits[0]=' grapes'
?>
```

Here $ fruits is an array variable. It contains 3 values" grapes, orange, mango". The elements of the array are accessed using an index. The index number of first element is zero, the index number of second element is one and soon.

So $ fruits[0]=' grapes'

Suppose if we want to add new element to this array. We can write statement

$fruits[3]=' banana';

Array using the notation key value pair

In PHP we have a different type of Array which relates key and value. This type of array is also known as hash or associative array.

Example of this array is given below. Position as key value.

```
<?
$fruits={ yellow=' mango'
          Purple=' grapes'
          Orange=' orange'
}
?>
```

Here $fruits is an array variable of type key values. We access element by using the key instead of position number. $fruits[yellow]='mango'. We are using the key yellow instead of position 0.

Creating an array:

Method 1:

```
$ Fruits=array(' strawberry'; grape' vanilla'; caramel'}
```

Method 2:

```
$ fruits[0]=' strawberry';
$ fruits[1]=' grapes';
```

Method 3:

```
$ fruits[' red']=' apple';
$ fruits[' yellow']=' mango';
```

Modifying an array element:

To add an element to the array we can write the statement $fruit[5] ='lemon'. Suppose if we don'tknow the last position to add a new element in array we can simply write

```
$ fruit[]=' strawberry';
```

Deleting an array element:

To remove an element we can use array_ push or array_ pop(). These two functions are built-in array functions in PHP.

Processing array with loop:

In PHP we can process arrays using loops like for, while, etc..
We also have a special loop in PHP known as for each loop for array processing.
Example for array processing with loop

```
<html>
<head></head>
<body>
Today shopping list;
<ul>
<?php
    $shoppinglist=array('green gram','bengal gram','rice');
    for($x=0;$x< sizeof($shoppinglist);x++)
   {
        echo<li>$shoppinglist[$x];
   }
   ?>
</ul>
</body>
</html>
```

In this example for loop is used to interact with the array. This loops extract elements from the array and prints in the screen one after another as an order list. The sizeof() function returns the number of elements in the array.

foreach loop()

This loop runs for each element of the array movning through the element of array on each interation. In **for loop**, we have condition statement and iteration(increment/decrement) statement. Condition statement and iteration statement are not needed in foreach() loop.
The syntax of foreach() loop is.

```
foreach (array variable as loop variable)
{
// loop statement
}
```

example using foreach() loop

```
<html>
<head></head>
<body>
Today's shopping list;
<ul>
<?php
$shoppinglist=array ('green gram', 'bengal gram', ''rice');
    foreach($shoppinglist as $ item)
{
       echo"<li>$item";
}
?>
</ul>
```

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

For each loop for an associative array:

Consider the following example for associative array

```
< html>
< head></ head>
< body>
$ can see
<ul>
<?php
// define associative array
$ animals=array(' dog'='Tripsy',' cat'=>'Tabitha', 'parrot'='polly)';
//iterate over it
foreach($ animal as $ key=$ value)
{
echo"</ u>A $ key name $ value";
    }
?>
</ul>
</ body>
</ html>
```

Output:

- A dog named Tripsy
- A cat named Tabitha
- A parrot named polly

Using array functions:

There are many built-in array functions in PHP that we can use along with array.

1. is_array()

This function check whether the variable in PHP is a array variable or not. It returns Boolean value as output

2. array_key()

This function returns the list of key in associative array for example this function will return 'dog', cat', 'parrot' from the array $animals.

3. array_value()

This function will return only the array element in an associative array. For example, this function returns 'Tripsy', 'Tabitha', 'polly' from the animals array,

4. list()

This list function assigns array elements to array variable. ex:

```
$flavours=array(' strawberry', 'grape', 'vanilla');
list($f1, $f2, $f3) = $flavours;
$f1 will have strawberry
$f2 will have grape
and so on
```

5. extract()

The extract() function iterates through (associative array) converting the key value pairs into corresponding variable value pairs.

$ fruits= array('red'= ‘apple’, ' yellow'=' banana', 'purple'= ‘grapes’);
Extract ($ fruits):
$red will have 'apple'
$yellow will have' banana'
$purple will have' grapes'

6. Array_push() function:

It adds an element from the end of the array.
Array_push($student, 'John');
The element John is added to the $students array.

7. Array_pop()

This function removes an element from the end of the array.
ex:-
array-pop($students)

8. array-shift ()

This function is used to pop element at the beginning of the array.
array-shift ($student)

9. array-unshift:

This function adds element at the beginning of the array.
array-unshift ($students, ‘Ronald');

10. Explode ()

The explode function splits a string into smaller components on the basis of a user defined character and then returns those element in an array.
ex:-
$string=' this is a book';
$words = explode (‘$string’, ‘ ‘);
This function returns and array variable $words will contain (‘this’, ‘is’, ‘book’)

11. implode ()

The implode function create a single string from all the element of an array joining them together with user defined separator.
EX:-
$words=array(‘This’,’ is’, ‘a’, ‘book’, ‘of’, ’Hindi’);
$string=implode(‘ ’,$words);
$string=’This is a book of Hindi’;

Function

A function is a set of program statements that perform a specific task. Functions can be called or executed from anywhere in the program. All the programming languages have built-in functions and also allow us to create user defined function. For example we can use the function with name pow from c library math.h or we can define our own function.

Usage of function:-

1. Reducing repetition:

User defined function enables developer to extract commonly used pieces of code as separate package.So it reduces unnecessary code repetition and redundancies also makes the code easier to understand and debug.

2. Easy maintenance:-

Because functions are defined oncee and used many times, they are used to maintain the code. During code maintenance if we want to change from values in calculation, we need to change only in the function. We need not Traverse the whole program for making change in one value.

3. Improves abstraction:

Function forces programmer to think in abstract terms. We need not worry about implementation of the function. It is enough that we know the function name, number of arguments and return type of the function.

Creating user defined function

defining function

Consider the example below which define a function for displaying Shakespeare quote in a webpage.

```
<?php
//define a function
function displayShakesphearQuote()
{
echo'some are born great, some achieve  greatness and some have greatness through upon them'
;.
}
```

invoking function

```
<?php
….
….
displayShakesphearQuote();
?>
```

- PHP functions are defined with the function keyword following by name of the function.
- The name of function should follow the rule for naming variables.
- After function name list of arguments enclosed in parenthesis() should be present.
- It is optional and can be omitted if no argument is present.
- After the first line of the function the body of the function should be present in between the curly brackets {}.
- The function code can contain any valid PHP statements, which includes loops conditional statements and call to other function.
- Invoking a function is done by calling a function with its name.
- If the function had arguments we have to specify the arguments during invoking.

Example of PHP script for function with argument.

```
<?php
//define a function
```

```
function triangle_area ($base,$height)
{
$area=$base*$height*0.5
return $area;
}
//invoke a function
$ta=triangle_area(10,50);
echo'the area of a triangle is $ta';
?>
```

Using arrays as function arguments aand return value

PHP fully supports passing arrays to function.

Ex:-

```
<?php
//define a function with array as argument
function addDomainToUsername($u,$d)
{
   //great empty result array
   $result_array=array();
   forreach($u as $element)
  {
       $result array[]=$element,'@',$d;
   }
   return $result_array;
}
$users=array('John', 'jim' ,'harry');
$newusers=addDomainToUsername($users, 'guese.me.domain');
?>
```

Write a PHP script for calculation of ncr value using recursive factorial function

```
<?php
fuction fact($F);
{
If f==o
return 1;
  else
    return f*fact f-1;
}

$n=5
$r=3
$RES=fact ($n/(fact $r*fact$n-$r)
echo "The result of ncr calculation";
Echo $RES
?>php
```

Defining global and local variable:-

The variables defined inside functionsare local variables. They cannot be used from outside the function. If we want to use a variable throughout the script in all functions we have to declare that variable with Global key word.

Global variable can be used in PHP script for counting the number of visitors in a website. This is done by the following statement.

Global $count

Super Global variables:-

There are some variables provided by the PHP interpreter that we can use in our PHP script. These variables can be accessed from anywhere in our program. There are variables which are used commonly for creation of websites with many web pages. All these Global variables starts with $_

They are

$_SERVER

$_POST

Used to collect data after submitting HTML form with method 10 =" post"

$_GET

Collect form data after submitting HTML form with method = "GET"

$_REQUEST

Used to collect data after submitting an HTML form.

$_GLOBALS

It contains all the super Global variables in installed PHP version.

$_FILES

$_SESSION

$_COOKIE

Include and require:-

Advantage of the function are realised when the function or statement calling the function remains in the same script (same PHP file). we can also use function that can be defined in other PHP files by importing the function definition.For importing PHP provides two useful functions. They are

(i) include()

(ii) require()

Eg:

```
<?php
//import file
include (“c:/php files /files.php”)
//we can call functions form first.php here.
?>
```

Include function generates warning message if the value in the file or function cannot be found but the execution of script continues.

Require function:-

Ex:-

```
<?php
//import files
require(“c:/php files /First.php”)
//we can call function from the first.php here
?>
```

The require function forces file to be included in the script. If the file or function cannot be found. It generates fatal error. This error stops script execution.

UNIT - III
SESSION

Session

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

What is a PHP Session?

When you work with an application, you open it, do some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is a problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.

Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.

So; Session variables hold information about one single user, and are available to all pages in one application.

If you need a permanent storage, you may want to store the data in a database.

Start a PHP Session

A session is started with the session_start() function.Session variables are set with the PHP global variable: $_SESSION.Now, let's create a new page called "demo_session1.php". In this page, we start a new PHP session and set some session variables:

```
<?php
    // Start the session
    session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
    // Set session variables
    $_SESSION["favcolor"] = "green";
    $_SESSION["favanimal"] = "cat";
    echo "Session variables are set.";
?>

</body>
</html>
```

Get PHP Session Variable Values

Next, we create another page called "demo_session2.php". From this page, we will access the session information we set on the first page ("demo_session1.php").

Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (session_start()).

Also notice that all session variable values are stored in the global $_SESSION variable:

Example

```
<?php
        session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
        // Echo session variables that were set on previous page
        echo "Favorite color is " . $_SESSION["favcolor"] . ".<br>";
        echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>

</body>
</html>
```

Another way to show all the session variable values for a user session is to run the following code:

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
print_r($_SESSION);
?>

</body>
</html>
```

How does it work? How users are identified?
Most sessions set a user-key on the user's computer that looks something like this: 765487cf34ert8dede5a562e4f3a7e12. Then, when a session is opened on another page, it scans the computer for a user-key. If there is a match, it accesses that session, if not, it starts a new session.

Modify a PHP Session Variable

To change a session variable, just overwrite it by using the following statement

```
$_SESSION["favcolor"] = "yellow";
```

Destroy a PHP Session

To remove all global session variables and destroy the session, use session_unset() and session_destroy()

Cookie

What is a Cookie?

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

Create Cookies With PHP

A cookie is created with the setcookie() function.

Syntax
setcookie(name, value, expire, path, domain, secure, httponly);
Only the name parameter is required. All other parameters are optional.

PHP Create/Retrieve a Cookie

The following example creates a cookie named "user" with the value "John Doe". The cookie will expire after 30 days (86400 * 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer).

We then retrieve the value of the cookie "user" (using the global variable $_COOKIE). We also use the isset() function to find out if the cookie is set:

Example

```
<?php
$cookie_name = "user";
$cookie_value = "Jegan Doe";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1 day
?>
<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
   echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
   echo "Cookie '" . $cookie_name . "' is set!<br>";
   echo "Value is: " . $_COOKIE[$cookie_name];
}
?>

</body>
</html>
```

Note: The setcookie() function must appear BEFORE the <html> tag.

Note: The value of the cookie is automatically URLencoded when sending the cookie, and automatically decoded when received (to prevent URLencoding, use setrawcookie() instead).

Modify a Cookie Value

To modify a cookie, just set (again) the cookie using the setcookie() function:

Example

```
<?php
$cookie_name = "user";
$cookie_value = "Atul  Kailash";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/");
?>
<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
   echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
   echo "Cookie '" . $cookie_name . "' is set!<br>";
   echo "Value is: " . $_COOKIE[$cookie_name];
}
?>

</body>
</html>
```

Delete a Cookie

To delete a cookie, use the setcookie() function with an expiration date in the past:

Example

```
<?php
// set the expiration date to one hour ago
setcookie("user", "", time() - 3600);
?>
<html>
<body>

<?php
echo "Cookie 'user' is deleted.";
?>

</body>
</html>
```

Check if Cookies are Enabled

The following example creates a small script that checks whether cookies are enabled. First, try to create a test cookie with the setcookie() function, then count the $_COOKIE array variable:

Example

```
<?php
setcookie("test_cookie", "test", time() + 3600, '/');
?>
<html>
<body>
```

```
<?php
if(count($_COOKIE) > 0) {
   echo "Cookies are enabled.";
} else {
   echo "Cookies are disabled.";
}
?>

</body>
</html>
```

PHP fscanf() Function

Definition and Usage

The fscanf() function parses the input from an open file according to the specified format.

Note: Any whitespace in the format string matches any whitespace in the input stream. This means that a tab (\t) in the format string can match a single space character in the input stream.

Syntax
fscanf(file, format, mixed)

Parameter	Description
file	Required. Specifies the file to check
format	Required. Specifies the format.

Possible format values:

%% - Returns a percent sign
%b - Binary number
%c - The character according to the ASCII value
%d - Signed decimal number
%e - Scientific notation (e.g. 1.2e+2)
%u - Unsigned decimal number
%f - Floating-point number (local settings aware)
%F - Floating-point number (not local settings aware)
%o - Octal number
%s - String
%x - Hexadecimal number (lowercase letters)
%X - Hexadecimal number (uppercase letters)
Additional format values. These are placed between the % and the letter (example %.2f):

+ (Forces both + and - in front of numbers. By default, only negative numbers are marked)
' (Specifies what to use as padding. Default is space. Must be used together with the width specifier. Example: %'x20s (this uses "x" as padding)
- (Left-justifies the variable value)
[0-9] (Specifies the minimum width held of to the variable value)
.[0-9] (Specifies the number of decimal digits or maximum string length)
Note: If multiple additional format values are used, they must be in the same order as above.

parse_ini_file() function Definition and Usage
The parse_ini_file() function parses a configuration (ini) file and returns the settings.

Tip: This function can be used to read in one's own configuration files, and has nothing to do with the php.ini file.

Note: The following reserved words must not be used as keys for ini files: null, yes, no, true, false, on, off, none. Furthermore, the following reserved characters must not be used in the key: {}|&~!()^".

Syntax
parse_ini_file(file, process_sections, scanner_mode)
Parameter Description
file Required. Specifies the ini file to parse
process_sections Optional. If set to TRUE, it returns is a multidimensional array with section names and settings included. Default is FALSE
scanner_mode
Optional. Can be one of the following values:

INI_SCANNER_NORMAL (default)
INI_SCANNER_RAW (means option values will not be parsed)
INI_SCANNER_TYPED (means that boolean, null and integer types are preserved when possible. "true", "on", "yes" are converted to TRUE. "false", "off", "no", "none" are converted to FALSE. "null" is converted to NULL. Numeric strings are converted to integer type if possible)

Getting file information with stat
Definition and Usage
The stat() function returns information about a file as an array.

Note: The results from this function will differ from server to server. The array may contain the number index, the name index, or both.
Note: The result of this function is cached. Use clearstatcache() to clear the cache.

Syntax
stat(filename)
Parameter Description
filename Required. Specifies the path to the file

Return Value:
An array with the following elements:
[0] or [dev] - Device number
[1] or [ino] - Inode number
[2] or [mode] - Inode protection mode
[3] or [nlink] - Number of links
[4] or [uid] - User ID of owner
[5] or [gid] - Group ID of owner
[6] or [rdev] - Inode device type
[7] or [size] - Size in bytes

[8] or [atime] - Last access (as Unix timestamp)
[9] or [mtime] - Last modified (as Unix timestamp)
[10] or [ctime] - Last inode change (as Unix timestamp)
[11] or [blksize] - Blocksize of filesystem IO (if supported)
[12] or [blocks] - Number of blocks allocated
It returns an E_WARNING on failure

Example

Get information about a file using stat():

```
<?php
$stat = stat("test.txt");
echo "Access time: " .$stat["atime"];
echo "<br>Modification time: " .$stat["mtime"];
echo "<br>Device number: " .$stat["dev"];
?>
```

Output
Access time: 1566689194
Modification time: 1550577107
Device number: 2049

Fseek

Definition and Usage
The fseek() function seeks in an open file.
This function moves the file pointer from its current position to a new position, forward or backward, specified by the number of bytes.
Tip: You can find the current position by using ftell()!
Syntax
fseek(file, offset, whence)

Parameter	Description
file	Required. Specifies the open file to seek in
offset	Required. Specifies the new position (measured in bytes from the beginning of the file)
whence	Optional. Possible values:

SEEK_SET - Set position equal to offset. Default
SEEK_CUR - Set position to current location plus offset
SEEK_END - Set position to EOF plus offset (to move to a position before EOF, the offset must be a negative value)

Return Value: 0 on success, otherwise -1

Copying files with copy

Definition and Usage
The copy() function copies a file.
Note: If the to_file file already exists, it will be overwritten.
Syntax
copy(from_file, to_file, context)

Parameter	Description
from_file	Required. Specifies the path to the file to copy from
to_file	Required. Specifies the path to the file to copy to
context	Optional. Specifies a context resource created with stream_context_create()

Return Value: TRUE on success, FALSE on failure
Example

```
<?php
echo copy("source.txt","target.txt");
?>
```

Deleting files

Deletion of files is done using unlink() function.
Definition and Usage
The unlink() function deletes a file.

Syntax
unlink(filename, context)

Parameter	Description
filename	Required. Specifies the path to the file to delete
context	Optional. Specifies the context of the file handle. Context is a set of options that can modify the behavior of a stream

Return Value: TRUE on success, FALSE on failure
Example
Delete a file using unlink()

```
<?php
$file = "test.txt";

if (!unlink($file)) {
  echo ("Error deleting $file");
} else {
  echo ("Deleted $file");
}
?>
```

Reading and writing binary files

A better method to open files is with the fopen() function.
The file may be opened in one of the following modes:

Modes	Description
r	**Open a file for read only**. File pointer starts at the beginning of the file
w	**Open a file for write only**. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a	**Open a file for write only**. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x	**Creates a new file for write only**. Returns FALSE and an error if file already exists

r+	**Open a file for read/write**. File pointer starts at the beginning of the file
w+	**Open a file for read/write**. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a+	**Open a file for read/write**. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x+	**Creates a new file for read/write**. Returns FALSE and an error if file already exists
b	**Opens the file in binary mode for reading/writing.**

PHP Read Single Character - fgetc()

The fgetc() function is used to read a single character from a file.

The example below reads the "webdictionary.txt" file character by character, until end-of-file is reached. After a call to the fgetc() function, the file pointer moves to the next character.

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one character until end-of-file
while(!feof($myfile)) {
  echo fgetc($myfile);
}
fclose($myfile);
?>
```

Locking files

In PHP we can lock the file completely or for editing. We have to use flocl() function

Definition and Usage

The flock() function locks and releases a file.

Syntax

```
flock(file, lock, block)
```

Parameter	Description
file	Required. Specifies an open file to lock or release
lock	Required. Specifies what kind of lock to use.

Possible values:

LOCK_SH - A shared lock (reader). Allow other processes to access the file

LOCK_EX - An exclusive lock (writer). Prevent other processes from accessing the file

LOCK_UN - Release the lock

LOCK_NB - Avoid blocking other processes while locking

block	Optional. Set to 1 to block other processes while locking

Return Value: TRUE on success, FALSE on failure

Example

Lock and release a file:

```
<?php
$file = fopen("test.txt","w+");

// exclusive lock
if (flock($file,LOCK_EX)) {
  fwrite($file,"Add some text to the file.");
  fflush($file);
  // release lock
  flock($file,LOCK_UN);
} else {
  echo "Error locking file!";
}
fclose($file);
?>
```

UNIT – IV
MYSQL

What is MySQL?

MySQL is a database system used on the web
MySQL is a database system that runs on a server
MySQL is ideal for both small and large applications
MySQL is very fast, reliable, and easy to use
MySQL uses standard SQL
MySQL compiles on a number of platforms
MySQL is free to download and use
MySQL is developed, distributed, and supported by Oracle Corporation
MySQL is named after co-founder Monty Widenius's daughter: My
The data in a MySQL database are stored in tables. A table is a collection of related data, and it consists of columns and rows.

Databases

Databases are useful for storing information categorically. A company may have a database with the following tables:

Employees
Products
Customers
Orders

PHP + MySQL Database System
PHP combined with MySQL are cross-platform (you can develop in Windows and serve on a Unix platform)

Database Queries

A query is a question or a request.
We can query a database for specific information and have a recordset returned.
Look at the following query (using standard SQL):

SELECT LastName FROM Employees
The query above selects all the data in the "LastName" column from the "Employees" table.

Download MySQL Database

If you don't have a PHP server with a MySQL Database, you can download it for free here: http://www.mysql.com
Facts About MySQL Database
MySQL is the de-facto standard database system for web sites with HUGE volumes of both data and end-users (like Facebook, Twitter, and Wikipedia).
Another great thing about MySQL is that it can be scaled down to support embedded database applications.

UNIT - V
PHP WITH MYSQL

PHP with MySQL Database

Usage of MySQL commands in PHP

With PHP, you can connect to and manipulate databases.
MySQL is the most popular database system used with PHP.
PHP 5 and later can work with a MySQL database using:

1. MySQLi extension (the "i" stands for improved)
2. PDO (PHP Data Objects)

Earlier versions of PHP used the MySQL extension. However, this extension was deprecated in 2012.
Should I Use MySQLi or PDO?
If you need a short answer, it would be "Whatever you like".

Both MySQLi and PDO have their advantages:

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

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

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

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

MySQL Examples in Both MySQLi and PDO Syntax
In this, and in the following chapters we demonstrate three ways of working with PHP and MySQL:

MySQLi (object-oriented)
MySQLi (procedural)
PDO
MySQLi Installation

For Linux and Windows: The MySQLi extension is automatically installed in most cases, when php5 mysql package is installed.

For installation details, go to: http://php.net/manual/en/mysqli.installation.php

PDO Installation

For installation details, go to: http://php.net/manual/en/pdo.installation.php

Database connectivity

Open a Connection to MySQL

Before we can access data in the MySQL database, we need to be able to connect to the server:

Example (MySQLi Object-Oriented)

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

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

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

Note on the object-oriented example above:

$connect_error was broken until PHP 5.2.9 and 5.3.0. If you need to ensure compatibility with PHP versions prior to 5.2.9 and 5.3.0, use the following code instead:

```
// Check connection
if (mysqli_connect_error()) {
   die("Database connection failed: " . mysqli_connect_error());
}
```

Example (MySQLi Procedural)

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

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

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

Example (PDO)

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

```
$password = "password";

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

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

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

Close the Connection

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

MySQLi Object-Oriented:

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

MySQLi Procedural:

```
mysqli_close($conn);
```

PDO:

```
$conn = null;
```

Processing result sets of queries

PHP Select Data From MySQL

Select Data From a MySQL Database

The SELECT statement is used to select data from one or more tables:

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

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

```
SELECT * FROM table_name
```

To learn more about SQL, please visit our SQL tutorial.

Select Data With MySQLi

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

Example (MySQLi Object-oriented)

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

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

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

if ($result->num_rows > 0) {
   // output data of each row
   while($row = $result->fetch_assoc()) {
      echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "<br>";
   }
} else {
   echo "0 results";
}
$conn->close();
?>
```

Code lines to explain from the example above:

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

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

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

Output

```
id: 1 - Name: John Doe
id: 2 - Name: Mary Moe
id: 3 - Name: Julie Dooley
```

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

Example (MySQLi Procedural)

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

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

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

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

mysqli_close($conn);
?>
```

You can also put the result in an HTML table:

Example (MySQLi Object-oriented)

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

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

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

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

```
} else {
   echo "0 results";
}
$conn->close();
?>
```

Output

ID	Name
1	John Doe
2	Mary Moe
3	Julie Dooley

Select Data With PDO (+ Prepared Statements)

The following example uses prepared statements.

It selects the id, firstname and lastname columns from the MyGuests table and displays it in an HTML table:

Example (PDO)

```
<?php
echo "<table style='border: solid 1px black;'>";
echo "<tr><th>Id</th><th>Firstname</th><th>Lastname</th></tr>";

class TableRows extends RecursiveIteratorIterator {
   function __construct($it) {
      parent::__construct($it, self::LEAVES_ONLY);
   }

   function current() {
      return "<td style='width:150px;border:1px solid black;'>" . parent::current(). "</td>";
   }

   function beginChildren() {
      echo "<tr>";
   }

   function endChildren() {
      echo "</tr>" . "\n";
   }
}

$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
   $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
```

```
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $stmt = $conn->prepare("SELECT id, firstname, lastname FROM MyGuests");
    $stmt->execute();

    // set the resulting array to associative
    $result = $stmt->setFetchMode(PDO::FETCH_ASSOC);
    foreach(new TableRows(new RecursiveArrayIterator($stmt->fetchAll())) as $k=>$v) {
        echo $v;
    }
}
catch(PDOException $e) {
    echo "Error: " . $e->getMessage();
}
$conn = null;
echo "</table>";
?>
```

Output

Id	Firstname	Lastname
1	John	Doe
2	Mary	Moe
3	Julie	Dooley

Handling errors

PHP mysqli_error() Function

Returns the last error description for the most recent function call, if any

Syntax

```
mysqli_error(connection);
```

Parameter Description

connection This parameter is Required. It Specifies the MySQL connection to use

Return Value: Returns a string with the error description. "" if no error occurred

Example

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno())
  {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
  }

// Perform a query, check for error
if (!mysqli_query($con,"INSERT INTO Persons (FirstName) VALUES ('Glenn')"))
  {
  echo("Error description: " . mysqli_error($con));
  }
```

```
mysqli_close($con);
?>
```

Debugging and diagnostic functions

In addition to the functions discussed in previous sections, PHP's MySQL API comes with a number of ancillary functions that may be used to find out more about the databases and tables on the MySQL server or to obtain server status information. Table lists the important functions in this category.

Function	What It Does
mysql_get_server_info()	Returns the version number of the MySQL server
mysql_get_proto_info()	Returns the version number of the MySQL protocol
mysql_get_client_info()	Returns the version number of the MySQL client
mysql_get_host_info()	Returns information on the MySQL host
mysql_thread_id()	Returns the thread ID for the current MySQL connection
mysql_list_dbs()	Returns a list of databases available on the MySQL server
mysql_list_tables()	Returns a list of tables available in a specified MySQL database
mysql_list_fields()	Returns information about the fields of a specified MySQL table
mysql_stat()	Returns status information about the MySQL server
mysql_info()	Returns information about the last executed query
mysql_db_name()	Returns a name of a database from the list generated by mysql_list_dbs()
mysql_tablename()	Returns a name of a table from the list generated by mysql_list_tables()
mysql_ping()	Tests the server connection

Validating user input through Database layer and Application layer

You may have heard the acronym GIGO before. It stands for "Garbage In, Garbage Out" and, very simply, it means that bad input produces bad output. Or, in PHP-MySQL terms, if the raw data inserted into a MySQL database is flawed, any subsequent analysis or report based on that data is sure to be misleading or incorrect.

That's where input validation comes in. By sanitizing and validating user input before it reaches the database, a developer guarantees the integrity of the database and creates a sound foundation for future calculations and operations. Such input validation is a critical part of your PHP-MySQL application development.

we can …

- Use database constraints to reduce the incidence of empty or duplicate records
- Set field data types to make input data more consistent
- Ensure that required form fields are filled
- Validate the length and data type of user input
- Use regular expressions for more complex pattern matching
- Capture validation errors in a single list, instead of one by one

The database layer in this context refers to the MySQL server, and the built-in functions and features it provides. The application layer here is the PHP script, or application, that interacts with the MySQL server to perform calculations or read/write data.

Setting Input Constraints at the Database Layer

When it comes to maintaining the integrity of your database, a powerful tool is provided by the database system itself: the capability to restrict the type of data entered into a field or make certain fields mandatory, using field definitions or constraints.

Using the NULL Modifier

MySQL enables you to specify whether a field is allowed to be empty or if it must necessarily be filled with data, by placing the NULL and NOT NULL modifiers after each field definition. This is a good way to ensure that required fields of a record are never left empty, because MySQL will simply reject entries that do not have all the necessary fields filled in. Here's an example of this in action:

```
mysql>CREATE TABLE products (
->id int(4),
->name varchar(50)
->);
Query OK, 0 rows affected (0.06 sec)
```

Here, the name field can hold NULL values, which means the following

INSERT will go unchallenged,

```
mysql>INSERT INTO products VALUES (NULL, NULL);
Query OK, 1 row affected (0.06 sec)
```

and create the following nonsense entry in the table:

```
mysql>SELECT * FROM products;
+------+------+
| id | name |
+------+------+
| NULL | NULL |
+------+------+
1 row in set (0.11 sec)
```

Now, look what happens if you make the name field mandatory:

```
mysql>CREATE TABLE products (
->id int(4),
->name varchar(50) NOT NULL
->);
Query OK, 0 rows affected (0.05 sec)

mysql>INSERT INTO products VALUES (NULL, NULL);
ERROR 1048: Column 'name' cannot be null
```

Of course, because MySQL makes a distinction between a NULL value and an empty string (''), the following record—which is also meaningless—would be accepted.

```
mysql>INSERT INTO products VALUES ('', '');
Query OK, 1 row affected (0.05 sec)

mysql>SELECT * FROM products;
```

```
+------+------+
| id | name |
+------+------+
| 0 | |
+------+------+
1 row in set (0.00 sec)
```

Thus, while the NOT NULL modifier can help reduce the incidence of empty or incomplete records in a database, it is not a comprehensive solution. It needs to be supplemented by application-level verification to ensure that empty strings are caught before they get to the database.

Using the UNIQUE Modifier

Using MySQL's built-in validation mechanisms has an important advantage: it makes it easy to perform certain types of validation that would be lengthy and time-consuming to write code for. Consider, for example, the situation of ensuring that a particular field contains only unique values. MySQL makes it possible to do this, simply by attaching a UNIQUE modifier to the field, as in the following example:

```
mysql>CREATE TABLE users (
->username VARCHAR(50) NOT NULL UNIQUE
->);
Query OK, 0 rows affected (0.06 sec)

mysql>INSERT INTO users (username) VALUES ('tim');
Query OK, 1 row affected (0.06 sec)

mysql>INSERT INTO users (username) VALUES ('jon');
Query OK, 1 row affected (0.00 sec)
```

Now, if you attempt to enter another record with the value *tim* in the username field, MySQL will reject your entry with an error:

```
mysql>INSERT INTO users (username) VALUES ('tim');
ERROR 1062: Duplicate entry 'tim' for key 1
```

If you have to perform this type of validation at the application layer, the only way to do it would be to select all the records in the table, scan the username field to obtain a list of all values present in it, and check the user's input against each to eliminate duplication. Needless to say, this is expensive, both in terms of CPU cycles and time. Fortunately, the UNIQUE modifier renders it unnecessary.

Using Field Data Types

Of course, checking for mandatory and unique values are just small pieces of a much bigger picture. It's also necessary to make sure that the data being entered is of the correct type—after all, you don't want string values in a numeric field or decimal values in a timestamp field. To this end, MySQL also requires you to specify the type of data a particular field can hold at the time of defining a table. Input that does not match the named data type is automatically converted into a more acceptable, though in correct, value. Here's an example of this:

```
mysql>CREATE TABLE items (
->id INT(2) NOT NULL,
```

```
->price INT(4) NOT NULL
->);
Query OK, 0 rows affected (0.05 sec)

mysql>INSERT INTO items (id, price) VALUES (1, 'five');
Query OK, 1 row affected (0.00 sec)

mysql>SELECT * FROM items;
+----+-------+
| id | price |
+----+-------+
| 1 | 0 |
+----+-------+
1 row in set (0.05 sec)
```

In this case, because the price field has been constrained to only store integers, the string *five* has been converted into a *0* and saved. Of course, this isn't perfect. Sure, you were able to avoid storing a string instead of a number, but you also simply replaced one problem with another: the field now contains a 0 instead of a valid price. This is a good example of how database constraints can help restrict input errors, yet not solve them completely. To close the gap between what should happen and what actually happens, it's necessary to also validate input at the application layer, before it even reaches the database.

Validating Input at the Application Layer

When it comes to catching errors in user input, the best place to do this is at the point of entry—the application itself.

Checking for Required Values

One of the most common mistakes a novice programmer makes is forgetting to check for required field values. This can result in a database with numerous empty records, and these empty records can, in turn, affect the accuracy of your queries. To see what I mean by this, consider the following users table:

```
mysql>CREATE TABLE users (
->username varchar(8) NOT NULL DEFAULT '',
->password varchar(8) NOT NULL DEFAULT ''
->) TYPE=MyISAM;
Query OK, 0 rows affected (0.05 sec)
```

When inserting a record into this table, values must be specified for bothusername and password fields (this is reinforced by the use of NOT NULLconstraints on these fields).Here's a script that enforces these constraints at theapplication level:

```
<html>
<head>
<basefont face="Arial">
</head>
<body>
<?php
if (!$_POST['submit'])
```

```
{
// form not submitted
?>
<form action="<?=$_SERVER['PHP_SELF']?>" method="post">
Username: <input type="text" name="username">
<br />
Password: <input type="password" name="password">
<br /><br />
<input type="submit" name="submit" value="Sign Up">
</form>
<?php
}
else
{
// form submitted
// check the username field
$username = ↲
(!isset($_POST['username']) || trim($_POST['username']) == "") ↲
? die ('ERROR: Enter a username') : ↲
mysql_escape_string(trim($_POST['username']));
// check the password field
$password = ↲
(!isset($_POST['password']) || trim($_POST['password'] == "")) ↲
? die ('ERROR: Enter a password') : ↲
mysql_escape_string(trim($_POST['password']));
// connect to database
// open connection
$connection = mysql_connect('localhost', 'guest', 'pass') ↲
or die ('Unable to connect!');
// select database
mysql_select_db('db2') or die ('Unable to select database!');
// create query
$query = "INSERT INTO users (username, password) ↲
VALUES ('$username', '$password')";
// execute query
$result = mysql_query($query) ↲
or die ("Error in query: $query. " . mysql_error());
// close connection
mysql_close($connection);
}
?>
</body>
</html>
```

To see this in action, attempt to submit the previous form with either of the two fields empty. The script will simply die() with an error message. This is because of the additional lines of input validation code in the script. In the previous listing, the validation test consists of checking for data in the username and password field, by using a combination of PHP's isset() and trim() functions. The isset() function checks whether the named variable is set or not, and

returns false if the variable has either not been set or assigned a NULL value. The trim() function removes the white space around the ends of the string, and then compares it with an empty string ("") to ensure that it contains at least one character. If both tests return true, then the script proceeds to connect to the database and insert the record into the table.

If either one returns false, the user clearly has not entered the corresponding form value, and the script terminates immediately, without even attempting to open a connection to the database. This listing makes it clear that a simple conditional test is all you need to ensure that required fields in your forms are never left empty. In the absence of these lines of validation code, the script would save a record to the database without first checking it for validity. In the context of the previous example, this means a user could successfully sign up without providing a username or a password—a clear error that also has serious ramifications for the security of your application (because an empty record exists, a user could gain access to the application, even without a username and password). This can be easily avoided by testing the user's input before it is saved to the database, in the manner described previously.

Restricting the Size of Input Data

MySQL enables you to control the length of a particular field by adding a size modifier to the field data type. Now, the way MySQL works, values greater than the specified length are automatically truncated, with no notification or exception generated to let the user know about the change. This is disturbing, because it means that user data can easily get corrupted without the user's awareness. As an example, go back a couple of pages and read the definition of the users table created in the previous section. You'll see that the two fields of the table are restricted to eight characters each. Now, if a user enters the user name *jamesscott,* MySQL will automatically (and silently) truncate it to eight characters and save it as *jamessco*. Obviously, any subsequent attempt by the user to log in as *jamesscott* will fail, as MySQL will have no record of that particular username.

One way around this is, of course, to set sensible length restrictions for your database fields. However, this must be coupled with application-level input validation of entered data, to alert users if their input goes above the prescribed limit and to allow them to modify it. To see an example of this in action, consider the following database table, which constrains data entered into the title field to 50 characters:

```
mysql>CREATE TABLE news (
->id INT (10) NOT NULL,
->title VARCHAR(50) NOT NULL
->);
Query OK, 0 rows affected (0.05 sec)
```

And here's the PHP script that replicates this constraint in a form:

```
<html>
<head>
<basefont face="Arial">
</head>
<body>
<?php
if (!$_POST['submit'])
{
// form not submitted
?>
```

```
<form action="<?=$_SERVER['PHP_SELF']?>" method="post">
Title: <input type="text" name="title">
<br />
<input type="submit" name="submit" value="Save">
</form>
<?php
}
else
{
// form submitted
// trim the title field
$title = trim ($_POST['title']);
// check its length
if (strlen($title) > 50)
{
die ('ERROR: Title contains more than 50 characters');
}
// connect to database
// save record
}
?>
</body>
</html>
```

To see this in action, try entering a string greater than 50 characters in the titlefield. When you submit the form, you'll see an error message, and the data will notbe saved to the database until you correct the error.The code behind this is straightforward—just pass the user input to PHP'sstrlen() function, which returns the length of the string. You can then wrap thisin an if() test to ensure that only strings under the specified limit pass muster.

Another simple way to implement a field length restriction is to use the MAXLENGTH attribute of the INPUT form tag. This attribute enables you to specify the maximum number of characters that can be entered ina form text input field. This is a quick way to restrict the length of user input. Note, this assumes the device you're displaying the form on supports this attribute. The major browsers all support it, but PDAs or cell phones running a reduced HTML implementation might not. For this reason, an application-level check should be performed, regardless of browser features like MAXLENGTH support.

Checking the Type of Input Data

You've already seen how MySQL automatically "corrects" values that don't match the data type specified in the table definition. Often, the assumptions MySQL makes when performing these corrections aren't true, and the corrected (but incorrect)values subsequently affect the integrity of your database. Therefore, an important test of user input involves checking the data type of input values against the database's expectations, and raising an error in the event of a mismatch.

To see an example of this, consider the following table definition:

```
mysql>CREATE TABLE items (
->itemID INT(11) NOT NULL AUTO_INCREMENT,
->itemName VARCHAR(255) NOT NULL DEFAULT '',
```

->itemSPrice FLOAT NOT NULL DEFAULT '0',
->itemCPrice FLOAT NOT NULL DEFAULT '0',
->itemQuantity INT(11) NOT NULL DEFAULT '0',
->PRIMARY KEY (itemID)
->) TYPE=MyISAM;
Query OK, 0 rows affected (0.07 sec)

Now, if you attempt to enter a string into any of the INT or FLOAT fields, MySQLwill simply convert that string to a 0. At first glance, this might seem like an intelligentthing to do, because it avoids having to deal with error messages. However, it isn't,because thedatabase now contains incorrect data.
What is needed, then, is a way to verify the data type of a value beforeallowing it to beentered into the database. A useful PHP function to accomplishthis is the is_numeric()function, demonstrated in the next example:

```
<html>
<head>
<basefont face="Arial">
</head>
<body>
<?php
if (!$_POST['submit'])
{
// form not submitted
?>
<form action="<?=$_SERVER['PHP_SELF']?>" method="post">
Item name:
<br />
<input type="text" name="itemName">
<br />
Item sale price:
<br />
<input type="text" name="itemSPrice">
<br />
Item cost price:
<br />
<input type="text" name="itemCPrice">
<br />

Item quantity:
<br />
<input type="text" name="itemQuantity">
<br/><br />
<input type="submit" name="submit" value="Enter Data">
</form>
<?php
}
else
{
// form submitted
```

```
// check the itemName field
$itemName = ↲
(!isset($_POST['itemName']) || trim($_POST['itemName']) == "") ↲
? die ('ERROR: Enter the item name') : ↲
mysql_escape_string(trim($_POST['itemName']));
// check the itemSPrice field
if(!isset($_POST['itemSPrice']) || ↲
trim($_POST['itemSPrice']) == "")
{
die ('ERROR: Enter the item\'s selling price');
}
elseif(!is_numeric(trim($_POST['itemSPrice'])))
{
die ('ERROR: Enter numeric value for the item\'s selling price');
}
else
{
$itemPrice = floatval(trim($_POST['itemSPrice']));
}
// check the itemCPrice field
if(!isset($_POST['itemCPrice']) || ↲
trim($_POST['itemCPrice']) == "")
{
die ('ERROR: Enter the item\'s cost price');
}
elseif (!is_numeric(trim($_POST['itemCPrice'])))
{
die ('ERROR: Enter numeric value for the item\'s cost price');
}
else
{
$itemCost = floatval(trim($_POST['itemCPrice']));
}
// check the itemQuantity field
if(!isset($_POST['itemQuantity']) || ↲
trim($_POST['itemQuantity']) == "")
{
die ('ERROR: Enter the quantity');
}
elseif (!is_numeric(trim($_POST['itemQuantity'])))
{
die ('ERROR: Enter numeric value for quantity');
}
else
{
$itemQuantity = intval(trim($_POST['itemQuantity']));
}
// connect to database
// save record
```

```
}
?>
</body>
</html>
```

Load this script, and try entering a nonnumeric value for the itemSPrice, itemCPrice, and item Quantity fields. Each attempt will be rejected with the display of an error message.

In this example, the first test is to ensure that the field is not empty. If this is true, the second test involves checking whether the value entered is a numeric string, with the is_numeric() function. Only if the user input passes both tests is it allowed to proceed into the database.

You cannot use the is_int() or the is_float() functions to test if a value submitted through a web form is an integer or a floating-point value. This is because data submitted through a form is always stored asa string within the special $_POST array. All you can do is use the is_numeric() function to check if a value is a numeric string.

In addition to the is_numeric() function, you may also use PHP's character type extension to further test input before saving them to your database. The important functions supported by this extension are listed in Table

Function	What It Does
ctype_alnum()	Check if a value contains only alphanumeric characters.
ctype_alpha()	Check if a value contains only alphabetic characters.
ctype_digit()	Check if a value contains only numeric characters.
ctype_print()	Check if a value contains only printable characters.
ctype_space()	Check if a value contains only white space characters.

Here's an example that illustrates some of these functions in action:

```
<html>
<head>
<basefont face="Arial">
</head>
<body>
<?php
if (!$_POST['submit'])
{
// form not submitted
?>
<form action="<?php echo $SERVER['PHP_SELF']; ?>" ↲
method="post">
First Name:
<br />
<input type="text" name="firstName">
<br />
Last Name:
```

```
<br />
<input type="text" name="lastName">
<br />
Age:
<br />
<input type="text" name="age">
<br /><br />
<input type="submit" name="submit" value="Enter Data">
</form>
<?php
}
else
{
// form submitted
// check the firstName field
if (!isset($_POST['firstName']) || ↲
trim($_POST['firstName']) == "")
{
die ('ERROR: Enter first name');
}
elseif(!ctype_alpha(trim($_POST['firstName'])))
{
die ('ERROR: Enter alphabetic value for first name');
}
else
{
$firstName = mysql_escape_string(trim($_POST['firstName']));
}
// check the lastName field
if(!isset($_POST['lastName']) || ↲
trim($_POST['lastName']) == "")
{
die ('ERROR: Enter last name');
}
elseif(!ctype_alpha(trim($_POST['lastName'])))
{
die ('ERROR: Enter alphabetic value for last name');
}
else
{
$lastName = mysql_escape_string(trim($_POST['lastName']));
}
// check the age field
if(!isset($_POST['age']) || trim($_POST['age']) == "")
{
die ('ERROR: Enter age');
}
elseif (!ctype_digit(trim($_POST['age'])))
{
```

```
die ('ERROR: Enter numeric value for age');
}
else
{
$age = floatval(trim($_POST['age']));
}
// connect to database
// save record
}
?>
```

In this script, after performing the basic tests, the ctype_alpha() function tests the input string for alphabetic characters, while the ctype_digit() function tests for digits from 0 to 9. Note, for values containing more than one character, every character is tested and a positive result is returned only if all the characters satisfy the data type requirements. While this rigidity is useful at times, it can also be a double-edged sword, for example, the number 20.50 would not pass a ctype_digit() test, because it contains a decimal point (which is not a digit).

Checking for Illegal Input Values

In addition to the tests listed in previous sections, an application's particular businesslogic often demands custom validation routines of its own. To illustrate this, considertheexample of a form that asks the user to enter a positive two-digit number. Here,it is necessary to write a validation test to check if the user's input falls between10 and 99 (both inclusive) and to display an error if it doesn't. Take a look at thenext script, which demonstrates what the code for such a validation test wouldlook like:

```
<html>
<head>
<basefont face="Arial">
</head>
<body>
<?php
if (!$_POST['submit'])
{
// form not submitted
?>
<form action="<?=$_SERVER['PHP_SELF']?>" method="post">
Enter any positive two-digit number:
<input type="text" name="num" size="2">
<br />
<input type="submit" name="submit" value="Check">
</form>
<?php
}
else
{
// form submitted
// check for presence of number
$num = ↲
(!isset($_POST['num']) || trim($_POST['num']) == "" || ↲
!is_numeric($_POST['num'])) ↲
```

```
? die ('ERROR: Enter a number') : trim($_POST['num']);
// check for number range
if ($num < 10 || $num > 99)
{
die ('ERROR: Enter a number between 10 and 99');
}
}
?>
</body>
</html>
```

This type of custom validation can play an important role in avoiding commonerrors, such as the dreaded division-by-zero error. Harking back to the example inthe previous section, assume you have a table containing the following data,

```
mysql>SELECT * FROM items;
+--------+----------+------------+------------+--------------+
| itemID | itemName | itemSPrice | itemCPrice | itemQuantity |
+--------+----------+------------+------------+--------------+
| 1 | Syringe | 10 | 5 | 200 |
| 2 | Swab | 1 | 0.25 | 1000 |
| 3 | Pump | 95 | 0 | 5 |
+--------+----------+------------+------------+--------------+
3 rows in set (0.00 sec)
```

and you'd like to calculate the percentage profit on each item using the formula Percentage Profit = (Profit/Cost Price) * 100. You'd probably need to run a SELECT query like this:

```
mysql>SELECT itemName, (((itemSPrice - itemCPrice)/itemCPrice) * 100) ↵
AS percentProfit FROM items;
+----------+---------------+
| itemName | percentProfit |
+----------+---------------+
| Syringe | 100 |
| Swab | 300 |
| Pump | NULL |
+----------+---------------+
3 rows in set (0.00 sec)
```

Notice how one of the records in the output displays a NULL value. This isbecause the cost price for that item was stored as 0, causing a division-by-zeroerror and forcing MySQL to display a NULL as the result of the calculation.This error might have been avoided if the application developer had thoughtto include a custom check to avoid zero values entering the database. Here's anexample of what that test might have looked like:

```
<html>
<head>
<basefont face="Arial">
</head>
<body>
```

```
<?php
if (!$_POST['submit'])
{
// form not submitted
?>
<form action="<?=$_SERVER['PHP_SELF']?>" method="post">
Item name:
<br />
<input type="text" name="itemName">
<br />
Item sale price:
<br />
<input type="text" name="itemSPrice">
<br />
Item cost price:
<br />
<input type="text" name="itemCPrice">
<br />
Item quantity:
<br />
<input type="text" name="itemQuantity">
<br/><br />
<input type="submit" name="submit" value="Enter Data">
</form>
<?php
}
else
{
// form submitted
// check the itemCPrice field
$itemCost = ( ↲
!isset($_POST['itemCPrice']) || trim($_POST['itemCPrice']) == "") ↲
? die ('ERROR: Enter the item\'s cost price') : ↲
(!is_numeric(trim($_POST['itemCPrice']))) ↲
? die ('ERROR: Enter numeric value for the item\'s cost price') : ↲
floatval(trim($_POST['itemCPrice']));
// check if itemCPrice field is equal to zero
if($itemCost == 0)
{
die ('ERROR: Please enter an item cost price greater ↲
than zero');
}
// connect to database
// save record
}
?>
```

Validating Dates

Dates often play an important role in an application's business logic, and usersare prone to errors when entering these values. Luckily, PHP comes with acheckdate() function that

provides an easy way to validate user-provideddate values.To see how this works, consider the following simple script, which asks theuser to enter a date, and then tests it for validity:

```
<html>
<head>
<basefont face="Arial">
</head>
<body>
<?php
if (!$_POST['submit'])
{
// form not submitted
?>
<form action="<?=$_SERVER['PHP_SELF']?>" method="post">
Day <input type="text" name="day" size="2">
Month <input type="text" name="month" size="2">
Year <input type="text" name="year" size="2">
<br />
<input type="submit" name="submit" value="Check">
</form>
<?php
}
else
{
// form submitted
// check date
if (!checkdate($_POST['month'], $_POST['day'], $_POST['year'])) ↵
{
die ('ERROR: Enter a valid date');
}
}
?>
</body>
</html>
```

To see this in action, enter an incorrect date—for example, 31 February 2005—andsubmit the form. Your date will be rejected with an error message.Most of the magic here happens with the checkdate() function. Thisfunction accepts three numeric arguments, representing the month, day, andyear, respectively, and returns true if the combination is a validGregorian date.

A good idea is to always check user-supplied date values in this manner beforeusing them.

(Un)Intelligent Automation

MySQL expects application developers to enforce date checking within theirapplication. If you enter an invalid date, MySQL will either store it as is,or convert it to a series of zeroes. From the usability point of view, bothalternatives are equally bad. Read more about this at **http://dev.mysql.com/doc/mysql/en/Using_DATE.html**.

Validating Multiple-Choice Input

Checkboxes and drop-down lists are an important component of web forms, and it'soften necessary to include validation for these controls in your PHP applications.Normally, theuser's selections from these controls are submitted to the formprocessor in the form of an array, and it's necessary to use PHP's array functions tovalidate them.To see this in action, consider the following script, which requires the user tofill out a brief user profile and select at least three hobbies and two subscriptionsfrom the multiple-choice controls presented.

```
<html>
<head>
<basefont face="Arial">
</head>
<body>
<?php
if (!$_POST['submit'])
{
// form not submitted
?>
<form action="<?=$_SERVER['PHP_SELF']?>" method="post">
Username:
<br />
<input type="text" name="username">
<p />
Password:
<br />
<input type="password" name="password">
<p />
Date of Birth:
<br />
Month <input type="text" name="month" size="2">
Day <input type="text" name="day" size="2">
Year <input type="text" name="year" size="4">
<p />
Hobbies (select at least <b>three</b>):
<br />
<input type="checkbox" name="hobbies[]" value="Sports">Sports
<input type="checkbox" name="hobbies[]" value="Reading">Reading
<input type="checkbox" name="hobbies[]" value="Travel">Travel
<input type="checkbox" name="hobbies[]" value="Television">Television
<input type="checkbox" name="hobbies[]" value="Cooking">Cooking
<p />
Subscriptions (Select at least <b>two</b>):
<br />
<select name="subscriptions[]" multiple>
<option value="General">General Newsletter</option>
<option value="Members">Members Newsletter</option>
<option value="Premium">Premium Newsletter</option>
</select>
<p />
<input type="submit" name="submit" value="Sign Up">
```

```
</form>
<?php
}
else
{
// form submitted
// validate "username", "password" and "date of birth" fields
$username = (!isset($_POST['username']) || ↲
trim($_POST['username']) == "") ↲
? die ('ERROR: Enter a username') : trim($_POST['username']);
$password = (!isset($_POST['password']) ↲
|| trim($_POST['password'] == "")) ↲
? die ('ERROR: Enter a password') : trim($_POST['password']);
if (!checkdate($_POST['month'], $_POST['day'], $_POST['year']))
{
die ('ERROR: Enter a valid date');
}
// check the "hobbies" field for valid values
$hobbies = ((sizeof($_POST['hobbies']) < 3) ? ↲
die ('ERROR: Please select at least 3 hobbies') : ↲
implode(',', $_POST['hobbies']));
// check the "subscriptions" field for valid values
$subscriptions = ((sizeof($_POST['subscriptions']) < 2) ? ↲
die ('ERROR: Please select at least 2 subscriptions') : ↲
implode(',', $_POST['subscriptions']));
// connect to database
// save record
}
?>
</body>
</html>
```

Now, try submitting the form without first selecting the required number of itemsfrom each multiple-choice control, and you will be presented with an error message.The options selected by a user from each multiple-choice control are submitted inthe form of a PHP array. Thus, it is convenient to use PHP's array functions—namely,the sizeof() function, which returns the number of elements in an array—to checkwhether the required number of options was selected.

Matching Patterns

Often, input validation requires more sophisticated tools than the primitive checksand tests shown in previous sections of this chapter. Fortunately, PHP comes withthese tools built in, with its support for regular expressions.Regular expressions (regex) are a powerful tool used in pattern-matching andsubstitution. Commonly associated with almost all *NIX-based tools, scriptinglanguages, and shell programs, a regular expression lets you build patterns usinga set of special characters. These patterns can then be compared with text in a file,data entered into an application, or input from a form filled up by users on a website. Depending on whether or not there's a match, appropriate action can be takenand appropriate program code executed. Regular expressions play an important rolein the decision-making routines of web applications,

and in complex find-and-replaceoperations.To see how regular expressions work, consider a form that requires the user toenter a name, password, and e-mail address. The application needs to enforce thefollowing constraints:

- The name may contain only uppercase (A–Z) or lowercase characters (a–z), with a minimum of three and a maximum of eight.
- The password may contain only lowercase characters (a–z) or integers (0–9), with a minimum of five and a maximum of eight.
- The e-mail address must conform to the standard *user@domain* format.

As you can see in the code listing that follows, these restrictions can beimplemented using regular expressions without adding too many extra lines of code:

```
<html>
<head>
<basefont face="Arial">
</head>
<body>
if (!$_POST['submit'])
{
// form not submitted
?>
<form action="<?=$_SERVER['PHP_SELF']?>" method="post">
Username:
<br />
<input type="text" name="username">
<p />
Password:
<br />
<input type="password" name="password">
<p />
Email address:
<br />
<input type="text" name="email">
<p />
<input type="submit" name="submit" value="Sign Up">
</form>
<?php
}
else
{
// form submitted
// validate "username", "password" and "email" fields
// using regular expressions
$username = (!isset($_POST['username']) || ↵
!ereg('^([a-zA-Z]){3,8}$', $_POST['username'])) ? ↵
die ('ERROR: Enter valid username') : ↵
mysql_escape_string(trim($_POST['username']));
$password = (!isset($_POST['password']) || ↵
!ereg('^([a-z0-9]){5,8}$', $_POST['password'])) ? ↵
die ('ERROR: Enter valid password') : ↵
```

```
mysql_escape_string(trim($_POST['password']));
$email = (!isset($_POST['email']) || ↲
!ereg('^([a-zA-Z0-9_-]+)([\.a-zA-Z0-9_-]+)@([a-zA-Z0-9_-]+) ↲
(\.[a-zA-Z0-9_-]+)+$', $_POST['email'])) ? ↲
die ('ERROR: Enter valid email address') : ↲
mysql_escape_string(trim($_POST['email']));
// connect to database
// save record
}
?>
</body>
</html>
```

This listing uses PHP's ereg() function to ensure that the user's input conformsto the constraints listed previously. This ereg() function requires two compulsoryarguments: the pattern to be matched, and the string to match it against.Coming to the regular expressions themselves, the first two are self-explanatory,especially if you have some familiarity with the syntax. Both of them list the allowedcharacters and also test the length of the string. The third and final pattern is a littlemore complex, because it uses numerous modifiers and special characters. Tounderstand it better, consider reading the article on regular expressions, or visitthe PHP manual page for the ereg() function, at http://www.php.net/manual/ en/function.ereg.php.

Listing Multiple Validation Errors at Once

In all the examples demonstrated so far, I have used the die() function toterminate script processing and display an error message if input validation fails.Now, while this is fine for forms with just a few fields, it doesn't make sense forforms that contain a large number of fields. For such forms, it is often more efficientto display a comprehensive list of errors at once, instead of displaying them one ata time, so that the user immediately has clear visibility of what (s)he did wrong. Forsome applications, you might even want to save the errors to a log file or databasefor future reference.

These varied requirements mean that it isn't enough for you only to catch errorsin userinput, but also present them in a manner that is efficient, extensible, and easyto understand. This next listing shows you how to validate all the data submitted bya user, prepare a list of errors encountered, and display them all at once.

```
<html>
<head>
<basefont face="Arial">
</head>
<body>
<?php
if (!$_POST['submit'])
{
// form not submitted
?>
<form action="<?=$_SERVER['PHP_SELF']?>" method="post">
Username (3-8 char):
<br />
<input type="text" name="username">
<p />
```

```
Password (5-8 char):
<br />
<input type="password" name="password">
<p />
Email address:
<br />
<input type="text" name="email">
<p />
Date of Birth:
<br />
Month <input type="text" name="month" size="2">
Day <input type="text" name="day" size="2">
Year <input type="text" name="year" size="4">
<p />
Hobbies (select at least <b>three</b>):
<br />
<input type="checkbox" name="hobbies[]" value="Sports">Sports
<input type="checkbox" name="hobbies[]" value="Reading">Reading
<input type="checkbox" name="hobbies[]" value="Travel">Travel
<input type="checkbox" name="hobbies[]" value="Television">Television
<input type="checkbox" name="hobbies[]" value="Cooking">Cooking
<p />
Subscriptions (Select at least <b>two</b>):
<br />
<select name="subscriptions[]" multiple>
<option value="General">General Newsletter</option>
<option value="Members">Members Newsletter</option>
<option value="Premium">Premium Newsletter</option>
</select>
<p />
<input type="submit" name="submit" value="Sign Up">
</form>
<?php
}
else
{
// array to store the error messages
$ERRORS = array();
// validate "username" field
$username = !ereg('^([a-zA-Z]){3,8}$', $_POST['username']) ? ↲
$ERRORS[] = 'Enter valid username' : ↲
mysql_escape_string(trim($_POST['username']));
// validate "password" field
$password = !ereg('^([a-z0-9]){5,8}$', $_POST['password']) ? ↲
$ERRORS[] = 'Enter valid password' : trim($_POST['password']);
// validate "email" field
$email = !ereg('^([a-zA-Z0-9_-]+)([\.a-zA-Z0-9_-]+)@([a-zA-Z0-9_-]+)(\ ↲
[a-zA-Z0-9_-]+)+$', $_POST['email']) ? ↲
$ERRORS[] ='Enter valid email address' : trim($_POST['email']);
```

```
// validate "date of birth" field
$dob = (!checkdate($_POST['month'], $_POST['day'], $_POST['year']) ? ↲
$ERRORS[] = 'Enter valid date of birth' : ↲
date("Y-m-d", mktime(0, 0, 0, $_POST['month'], $_POST['day'], ↲
$_POST['year'])));
// validate "hobbies" field
$hobbies = (sizeof($_POST['hobbies']) < 3) ? ↲
$ERRORS[] = 'Please select at least three hobbies' : ↲
implode(',', $_POST['hobbies']);
// validate "subscriptions" field
$subscriptions = (sizeof($_POST['subscriptions']) < 2) ? $ERRORS[] = ↲
'Please select at least two subscriptions' : ↲
implode(',', $_POST['subscriptions']);
// verify if there were any errors by checking
// the number of elements in the $ERRORS array
if(sizeof($ERRORS) > 0)
{
// format and display error list
echo "<ul>";
foreach ($ERRORS as $e)
{
echo "<li>$e</li>";
}
echo "</ul>";
die();
}
// no errors?
// connect to database
// save record
}
?>
</body>
</html>
```

Here, for every input test that fails, a new element is added to the global$ERRORS array. At the end of the tests, before connecting to the database, thisarray is checked. If it contains one or more elements, script processing stops andthe errors are displayed to the user as a bulleted list.

Formatting query output with Character-Numeric-Date and Time

That's where this chapter comes in. The focus of this chapter is messaging the output of your MySQL queries so it conforms to the expectations of your users and, thereby, becomes more readable and useful. Both PHP and the MySQL RDBMS come with a number of built-in functions to perform such output formatting.

Most of the important formatting options are

- Join multiple fields into a single string, using custom separators
- Make string or numeric data a uniform size with left/right padding
- Translate line breaks and special characters in text fields to their HTML equivalents
- Format numbers according to local or international currency conventions

- Use commas or other user-defined characters to make large numeric values more readable
- Truncate or round large floating-point values to one or two decimal places
- Display English-equivalent day and month names for UNIX timestamps or numeric date/time values
- Perform simple date arithmetic
- Break the results of a SELECT query into multiple "pages," and dynamically present links to move between pages

Formatting Character Data

A lot of your MySQL data is going to be stored as strings or text blocks, in CHAR,VARCHAR, or TEXT fields. It's essential that you know how to manipulate this stringdata and adjust it to fit the requirements of your application user interface. Both PHPand MySQL come equipped with numerous string manipulation functions (in fact,they overlap in functionality in many places), and the following sections discuss theimportant ones.

Concatenating String Values

You learned about string concatenation in PHP in Chapter 3. It's pretty simple—juststring together the variables you want to concatenate using the PHP concatenationoperation, a period (.). Concatenating fields from a MySQL result set is equallysimple—just assign the field values to PHP variables and concatenate the variablestogether in the normal manner.

To see how this works, consider the following table:

```
mysql>SELECT * FROM users;
+-----------+---------+----------+
| username | fname | lname |
+-----------+---------+----------+
| matt | Matthew | Johnson |
| har56 | Harry | Thompson |
| kellynoor | Kelly | Noor |
| jimbo2003 | Jim | Doe |
| x | Xavier | Belgudui |
+-----------+---------+----------+
5 rows in set (0.00 sec)
```

Now, assume you need to concatenate the first- and last-name fields into a single value (a common requirement). Here's how:

```
<html>
<head></head>
<body>
<?php
// open connection to MySQL server
$connection = mysql_connect('localhost', 'guest', 'pass') ↲
or die ('Unable to connect!');
// select database for use
mysql_select_db('db2') or die ('Unable to select database!');
// create and execute query
$query = 'SELECT fname, lname FROM users';
$result = mysql_query($query) ↲
or die ('Error in query: $query. ' . mysql_error());
```

```
// check if records were returned
if (mysql_num_rows($result) > 0)
{
// print HTML table
echo '<ul>';
// iterate over record set
// print each field
while($row = mysql_fetch_object($result))
{
// prints in format "last-name, first-name"
echo '<li>' . $row->lname . ', ' . $row->fname;
}
echo '</ul>';
}
else
{
// print error message
echo 'No rows found!';
}
// once processing is complete
// free result set
mysql_free_result($result);
// close connection to MySQL server
mysql_close($connection);
?>
</body>
</html>
```

Figure 15-1 illustrates what the output looks like.

There's another way to do this as well, though. MySQL comes with two built-infunctions—CONCAT() and CONCAT_WS()—which can be used to glue fieldstogether within the SQLquery itself. Take a look at this next snippet from theMySQL interactive client, which shows these functions in action:

```
mysql>SELECT CONCAT(fname, lname) FROM users ↵
WHERE username = 'matt';
+---------------------+
| CONCAT(fname, lname) |
+---------------------+
| MatthewJohnson |
+---------------------+
1 row in set (0.02 sec)

mysql>SELECT CONCAT_WS(', ', lname, fname) FROM users ↵
WHERE username = 'matt';
+-----------------------------+
| CONCAT_WS(', ', lname, fname) |
+-----------------------------+
| Johnson, Matthew |
```

```
+------------------------------+
1 row in set (0.00 sec)
```

Concatenating string values

Note the difference between the two functions: the CONCAT() function concatenates two or more fields, while the CONCAT_WS() function lets you specify a string separator between the concatenated field values. Obviously, the CONCAT_WS() function is used more often; it's also more forgiving of NULLs in your table (see the following Caution for more information).

Ensure that none of the fields you're trying to join with CONCAT() contain NULLs. This is because the function returns a NULL value if any of its input arguments are NULL. This quirk can produce unexpected results,damaging the carefully cultivated look of your output screens. To avoid this, check for NULL values prior to using the function, and ensure that your database and validation rules are rigid enough to prevent the entry of empty/NULL values into fields that aren't supposed to contain them (Chapter 14 has more information on how to do this).

The CONCAT_WS() function is more forgiving, simply ignoring NULL values if it encounters them.

Here's a rewrite of the previous script that uses these database-level functionsto perform the concatenation and achieve the same result:

```
<html>
<head></head>
<body>
<?php
// open connection to MySQL server
$connection = mysql_connect('localhost', 'guest', 'pass') ↲
or die ('Unable to connect!');
// select database for use
mysql_select_db('db2') or die ('Unable to select database!');
// create and execute query
$query = "SELECT CONCAT_WS(', ', lname, fname) AS name ↲
FROM users";
$result = mysql_query($query) ↲
or die ('Error in query: $query. ' . mysql_error());
// check if records were returned
if (mysql_num_rows($result) > 0)
{
// print HTML table
echo '<ul>';
// iterate over record set
// print each field
while($row = mysql_fetch_object($result))
{
// prints in format "last-name, first-name"
echo '<li>' . $row->name;
}
echo '</ul>';
}
else
```

```
{
// print error message
echo 'No rows found!';
}
// once processing is complete
// free result set
mysql_free_result($result);
// close connection to MySQL server
mysql_close($connection);
?>
</body>
</html>
```

Padding String Values

You read about the PHP trim() function, used to strip leading andtrailing white space from string values prior to testing them for validity or insertingthem into a database. However, PHP also comes with the str_pad() function,which does just the reverse: it pads strings to a specified length using either whitespace or a user-specified character sequence. This can come in handy if you needto artificially elongate string values for display or layout purposes.

Here's a table containing string values of differing lengths:

```
mysql>SELECT * FROM ingredients;
+----------------+
| name |
+----------------+
| cinnamon |
| ginger |
| red pepper |
| cloves |
| peas |
| tender coconut |
+----------------+
6 rows in set (0.00 sec)
```

And here's some PHP code that demonstrates padding them:

```
<html>
<head></head>
<body>
<pre>
<?php
// open connection to MySQL server
$connection = mysql_connect('localhost', 'guest', 'pass') ↲
or die ('Unable to connect!');
// select database for use
mysql_select_db('db2') or die ('Unable to select database!');
// create and execute query
$query = "SELECT name FROM ingredients";
$result = mysql_query($query) ↲
or die ('Error in query: $query. ' . mysql_error());
// check if records were returned
```

```
if (mysql_num_rows($result) > 0)
{
// iterate over record set
// print each field
while($row = mysql_fetch_object($result))
{
// prints " name"
echo str_pad($row->name, 30, ' ', STR_PAD_LEFT) . '<br />';
}
}
else
{
// print error message
echo 'No rows found!';
}
// once processing is complete
// free result set
mysql_free_result($result);
// close connection to MySQL server
mysql_close($connection);
?>
</pre>
</body>
</html>
```

Figure 15-2 illustrates what the output looks like.

The str_pad() function takes three parameters: the variable to be padded, thesize it should be padded to, and the character to use for padding. By default, thefunction pads the string on the right side. You can alter this default, however, bypassing one of the constants STR_PAD_LEFT or STR_PAD_BOTH to the functionas an optional fourth parameter.The PHP str_pad() function is functionally equivalent to MySQL's RPAD()and LPAD() functions, which pad a string from the right and left, respectively. Thefollowing snippets demonstrate how these functions work:

```
mysql>SELECT RPAD(name, 20,'_'), LPAD(name, 20, '_') ↲
FROM ingredients LIMIT 0,2;
+------------------------+------------------------+
| RPAD(name, 20,'_') | LPAD(name, 20, '_') |
+------------------------+------------------------+
| cinnamon_________ | _______cinnamon |
+------------------------+------------------------+
| ginger____________ | ___________ginger |
+------------------------+------------------------+
2 rows in set (0.00 sec)
```

A word of caution: if the total length specified in the RPAD() and LPAD()function call is less than the length of the field value, the value will be truncated.The next snippet illustratesthis:

```
mysql>SELECT RPAD(name, 5, '_') FROM ingredients ↲
WHERE name = 'cinnamon';
+-----------------------+
```

```
| RPAD(name, 5, '_') |
+------------------------+
| cinna |
+------------------------+
1 row in set (0.00 sec)
```

Padding string values

PHP's str_pad() function, however, does not truncate strings inequivalent situations.

Altering String Case

If you need case manipulation, just reach for PHP's string manipulation API again.Four useful functions are here: strtolower(), which converts all characters ina string to lowercase; strtoupper(), which converts all characters to uppercase;ucfirst(), which converts the first character of a string to uppercase, and theuseful ucwords(), which converts the first character of all the words in a stringto uppercase.

The following example demonstrates these functions, using them on thedifferent fields of the following table:

```
mysql>SELECT * FROM customers;
+-------+---------+----------------+----------+---------------------------+
| fname | lname | addr | city | email |
+-------+---------+----------------+----------+---------------------------+
| David | Johnson | 18 mcgoo place,
ray road | boston | David_Johnson@CORPMAIL.DOM |
| Flora | Bharti | 239/a harkrishna bldg,
j b marg | hyderabad| bharti@MyOwnCompany.in |
| joe | cool | 15 hill view,
east end road | yorktown | joecool@guess.it |
+-------+---------+----------------+----------+---------------------------+
3 rows in set (0.00 sec)
```

Here's the code that reformats all this data to a more consistent casing style:

```
<html>
<head></head>
<body>
<?php
// open connection to MySQL server
$connection = mysql_connect('localhost', 'guest', 'pass') ↲
or die ('Unable to connect!');
// select database for use
mysql_select_db('db2') or die ('Unable to select database!');
// create and execute query
$query = "SELECT * FROM customers";
$result = mysql_query($query) ↲
or die ('Error in query: $query. ' . mysql_error());
// check if records were returned
if (mysql_num_rows($result) > 0)
{
// iterate over record set
// print each field
echo '<table border=1 cellpadding=10>';
```

```
echo '<tr><td>Name</td><td>Mailing Address</td>↵
<td>Email Address</td></tr>';
while($row = mysql_fetch_object($result))
{
echo '<tr>';
echo '<td>' . ucfirst($row->fname) . ' ' . ↵
ucfirst($row->lname) . '</td>';
echo '<td>' . ucwords($row->addr) . '<br />' . ↵
strtoupper($row->city) . '</td>';
echo '<td>' . strtolower($row->email) . '</td>';
echo '</tr>';
}
echo '</table>';
}
else
{
// print error message
echo 'No rows found!';
}
// once processing is complete
// free result set
mysql_free_result($result);
// close connection to MySQL server
mysql_close($connection);
?>
</body>
</html>
```

Figure 15-3 shows an output sample.

If you want to, you could also do some of this in the query string itself, by usingMySQL's UCASE() and LCASE() functions. The following snippet illustrates this:

```
mysql>SELECT CONCAT_WS('\n', UCASE(addr), UCASE(city)) ↵
AS address, LCASE(email) AS email FROM customers;
+-------------------------------+--------------------------+
| address | email |
+-------------------------------+--------------------------+
| 18 MCGOO PLACE, RAY ROAD | |
| BOSTON | david_johnson@corpmail.dom |
| 239/A HARKRISHNA BLDG, J B MARG | |
| HYDERABAD | bharti@myowncompany.in |
```

Changing string case

```
| 15 HILL VIEW, EAST END ROAD | |
| YORKTOWN | joecool@guess.it |
+-------------------------------+--------------------------+
3 rows in set (0.11 sec)
```

MySQL does not offer functions to capitalize the first character of astring value. If you need to do this, use the PHP functions described previously.

Dealing with Special Characters

When it comes to displaying large text blocks on a web page, a PHP developer must grapple with a number of issues. Special characters need to be protected, white space and line breaks must be preserved, and potentially malicious HTML code must be defanged. PHP comes with a number of functions designed to perform just these tasks.

Repeat Business

MySQL also provides a REPEAT() function, which can be used to displaya string fieldmultiple times. Here's an example:

```
mysql>SELECT REPEAT('ho ', 5);
+------------------+
| REPEAT('ho ', 5) |
+------------------+
| ho ho ho ho ho |
+------------------+
1 row in set (0.00 sec)
```

PHP's equivalent function is the str_repeat() function.

To illustrate, consider a table containing large blocks of text data, like the following one:

```
mysql>SELECT id, data FROM newsdata LIMIT 0,1;
+----+-----------------------------------------------------------------+
| id | data |
+----+-----------------------------------------------------------------+
| 1 | Recently, I put together a Web site and the public actually liked |
my <html>&<javascript>. People... |
+----+-----------------------------------------------------------------+
1 row in set (0.00 sec)
```

Now, here's how you'd normally retrieve and display this information in a web page:

```
<html>
<head></head>
<body>
<font face="Arial" size="-1">
<?php
// open connection to MySQL server
$connection = mysql_connect('localhost', 'guest', 'pass') ↲
or die ('Unable to connect!');
// select database for use
mysql_select_db('db2') or die ('Unable to select database!');
// create and execute query
$query = "SELECT title, data FROM newsdata";
$result = mysql_query($query) ↲
or die ('Error in query: $query. ' . mysql_error());
// check if records were returned
if (mysql_num_rows($result) > 0)
{
// iterate over record set
while($row = mysql_fetch_object($result))
```

```
{
echo '<b>' . $row->title . '</b>';
echo '<p />';
echo $row->data;
echo '<p />';
}
}
else
{
// print error message
echo 'No rows found!';
}
// once processing is complete
// free result set
mysql_free_result($result);
// close connection to MySQL server
mysql_close($connection);
?>
</font>
</body>
</html>
```

Figure 15-4 illustrates what this looks like.

If you compare the output of the previous script with the original table data,you'll see numerous discrepancies: line breaks and white space are not correctlyrendered, and characters like <,>, and & are interpreted as HTML by the browserinstead of being displayed as is. The integrity of the original text block has, therefore,
been compromised.

To correct these discrepancies, alter the script so it looks like this:

```
<html>
<head></head>
<body>
<font face="Arial" size="-1">
<?php
// open connection to MySQL server
$connection = mysql_connect('localhost', 'guest', 'pass') ↲
or die ('Unable to connect!');
// select database for use
mysql_select_db('db2') or die ('Unable to select database!');
// create and execute query
$query = "SELECT title, data FROM newsdata";
$result = mysql_query($query) ↲
or die ('Error in query: $query. ' . mysql_error());
// check if records were returned
if (mysql_num_rows($result) > 0)
{
// iterate over record set
while($row = mysql_fetch_object($result))
{
```

```
echo '<b>' . $row->title . '</b>';
echo '<p />';
echo nl2br(wordwrap(htmlentities($row->data), 70));
                Printing a text block as is, resulting in display errors
echo '<p />';
}
}
else
{
// print error message
echo 'No rows found!';
}
// once processing is complete
// free result set
mysql_free_result($result);
// close connection to MySQL server
mysql_close($connection);
?>
</font>
</body>
</html>
```

Figure 15-5 illustrates the revised output.
The revised listing uses three new functions.

The htmlentities() function takes care of replacing special characters like ", &, <, and > with their corresponding HTML entity values. This function is useful to defang user-supplied HTML text and render it incapable of effecting the display or functionality of your web page. This function also translates these special characters and prevents them from being interpreted as HTML code by the browser.

Next, the wordwrap() function wraps text to the next line once it reaches a particular, user-defined size, by inserting the /n newline character at appropriate points in the text block (these are then converted into HTML line breaks by the next function). This can be used to set artificial boundaries on the width of your text display area, and to maintain the integrity of your page layout.

Finally, the nl2br() function automatically preserves newlines in a text block, by converting them to HTML
 elements. This makes it possible to reproduce the original formatting of the text when it is displayed. While wordwrap() is a useful way to restrict your text display areas to specific dimensions, it isn't the best. Using CSS width and height rules or constrained s to control the size of your text display areas is usually more appropriate.

PHP also comes with a strip_tags() function, which enables you to strip all the HTML and PHP tags out of a string, returning only the ASCII output. This can be useful if your application has a rigid "no HTML input" policy.Printing a text block, after correcting for special characters and line breaks

Formatting Numeric Data

Just as you can massage string values into a number of different shapes, so, too, can you format numeric data. Both PHP and MySQL come with a full set of functions to manipulate integer and floating-point numbers, and to format large numeric values for greater readability.

Using Decimal and Comma Separators

When it comes to formatting numeric values in PHP, there are only two functions: number_format() and sprintf(). Of these, the former is easier to understand and use, so let's begin with that function.

The number_format() function is used to display large numbers with comma and decimal separators. It can be used to control both the visibility and the appearance of the decimal digits, as well as the character used as the thousands separator.

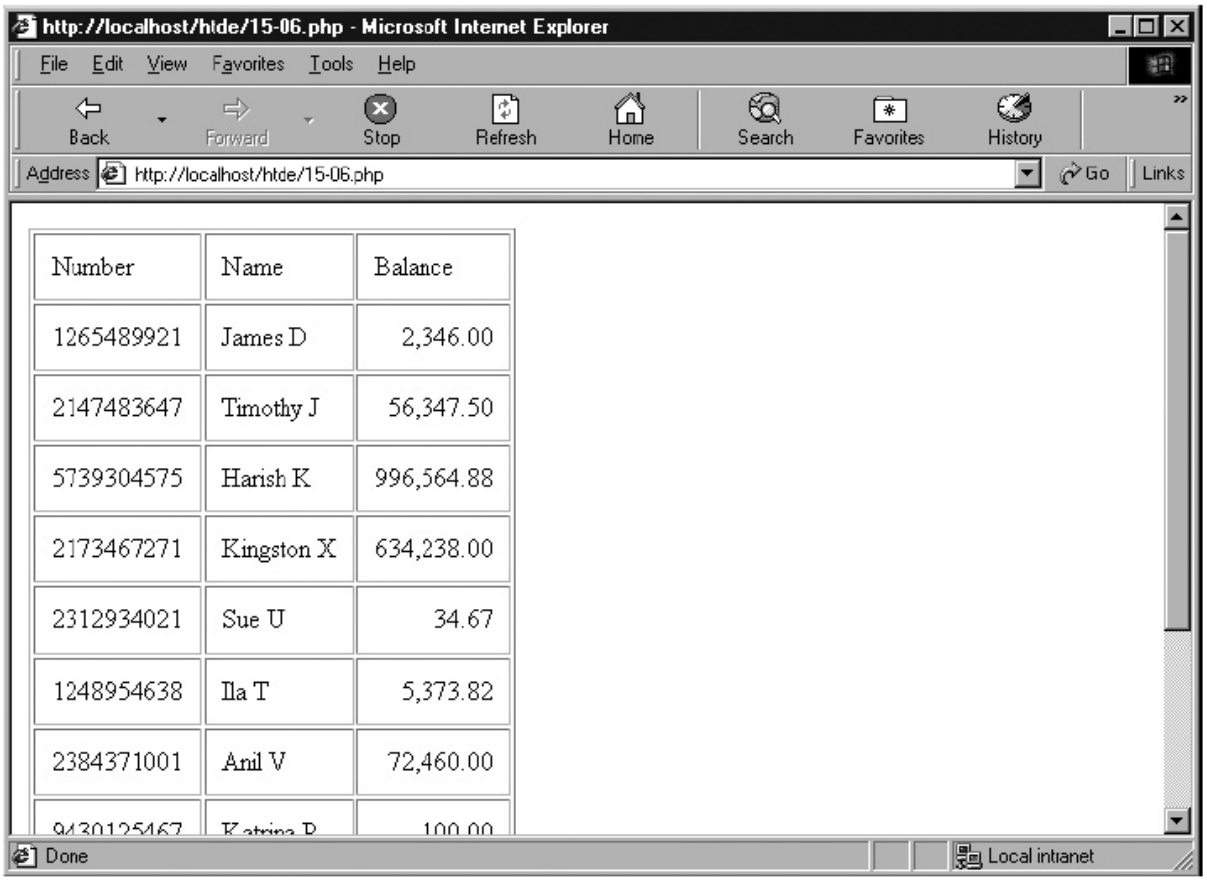
To see how this works, consider the following table:

```
mysql>SELECT accountNumber, accountName, ↲
accountBalance FROM accounts;
+--------------+-------------+----------------+
| accountNumber | accountName | accountBalance |
+--------------+-------------+----------------+
| 1265489921 | James D | 2346.00000 |
| 2147483647 | Timothy J | 56347.50000 |
| 5739304575 | Harish K | 996564.87500 |
| 2173467271 | Kingston X | 634238.00000 |
| 2312934021 | Sue U | 34.67000 |
| 1248954638 | Ila T | 5373.81982 |
| 2384371001 | Anil V | 72460.00000 |
| 9430125467 | Katrina P | 100.00000 |
| 1890192554 | Pooja B | 17337.11914 |
| 2388282010 | Sue U | 388883.12500 |
| 2374845291 | Jacob N | 18410.00000 |
+--------------+-------------+----------------+
11 rows in set (0.05 sec)
```

Here's a PHP script that displays this information on a web page, usingnumber_format() to display account balances with two decimal places andcommas as thousand separators:

```
<html>
<head></head>
<body>
<?php
// open connection to MySQL server
$connection = mysql_connect('localhost', 'guest', 'pass') ↲
or die ('Unable to connect!');
// select database for use
mysql_select_db('db2') or die ('Unable to select database!');
// create and execute query
$query = "SELECT accountNumber, accountName, accountBalance ↲
FROM accounts";
$result = mysql_query($query) ↲
or die ('Error in query: $query. ' . mysql_error());
// check if records were returned
if (mysql_num_rows($result) > 0)
{
echo '<table border=1 cellpadding=10>';
echo '<tr><td>Number</td><td>Name</td><td>Balance</td></tr>';
// iterate over record set
while($row = mysql_fetch_object($result))
```

```
{
echo '<tr>';
echo '<td>' . $row->accountNumber . '</td>';
echo '<td>' . $row->accountName . '</td>';
echo '<td align=right>' . ↲
number_format($row->accountBalance, 2, '.', ',') . '</td>';
echo '</tr>';
}
echo '</table>';
}
else
{
// print error message
echo 'No rows found!';
}
// once processing is complete
// free result set
mysql_free_result($result);
// close connection to MySQL server
mysql_close($connection);
?>
</body>
</html>
```



shows the output of this script. Notice how the use of a comma separator significantly increases the readability of the numbers.

Formatting numbers with the number_format() function

You've already used the echo() function extensively to display output.However, echo()doesn't let you format output in any significant manner, forexample, you can't write 1 as 00001.00. So, another common function used to performthis type of number formatting is the sprintf() function, which enables you todefine the format in which data is output.

Consider the following example:

```
<?php
// returns 1.6666666666667
print(5/3);
?>
```

As you might imagine, that's not very friendly. Ideally, you'd like to displayjust the significant digits of the result, so you'd use the sprintf() function, asin the following:

```
<?php
// returns 1.67
echo sprintf("%1.2f", (5/3));
?>
```

The PHP sprintf() function is similar to the sprintf() function thatC programmers are used to. To format the output, you need to use *field templates,* templates that represent the format you'd like to display. Common field template sare listed in Table

Template	What It Represents
%s	string
%d	decimal number
%x	hexadecimal number
%o	octal number
%f	float number

You can also combine these field templates with numbers that indicate the number of digits to display—for example, %1.2f implies that PHP should only display two digits after the decimal point. If you'd like the formatted string to have a minimum length, you can tell PHP which character to use for padding by prefixing it witha single quote (').

Here are a few more examples of sprintf() in action:

```
<?php
// returns 00003
echo sprintf("%05d", 3);
// returns $25.99
echo sprintf("$%2.2f", 25.99);
// returns ****56
printf("%'*6d", 56);
?>
```

To see a real-world example of sprintf() usage, consider the followingnumber-heavy MySQL table:

```
mysql>SELECT * FROM stocks;
+--------+-------------+------------+------------+------------+-----------+
| symbol | qty | buy | sell | high | low |
+--------+-------------+------------+------------+------------+-----------+
| HGTY | 17000.0000 | 289.9786 | 195.7474 | 315.7643 | 187.9540|
| HDYS | 5.8701 | 19000.2734 | 21759.6465 | 21759.6465 | 18639.2988|
```

```
| IWIK | 2174733.0000 | 868.0000 | 870.0000 | 891.0000 | 800.0000|
+--------+-------------+------------+------------+------------+-----------+
3 rows in set (0.00 sec)
```

Here's the PHP script that formats this mass of data into something more readable:

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

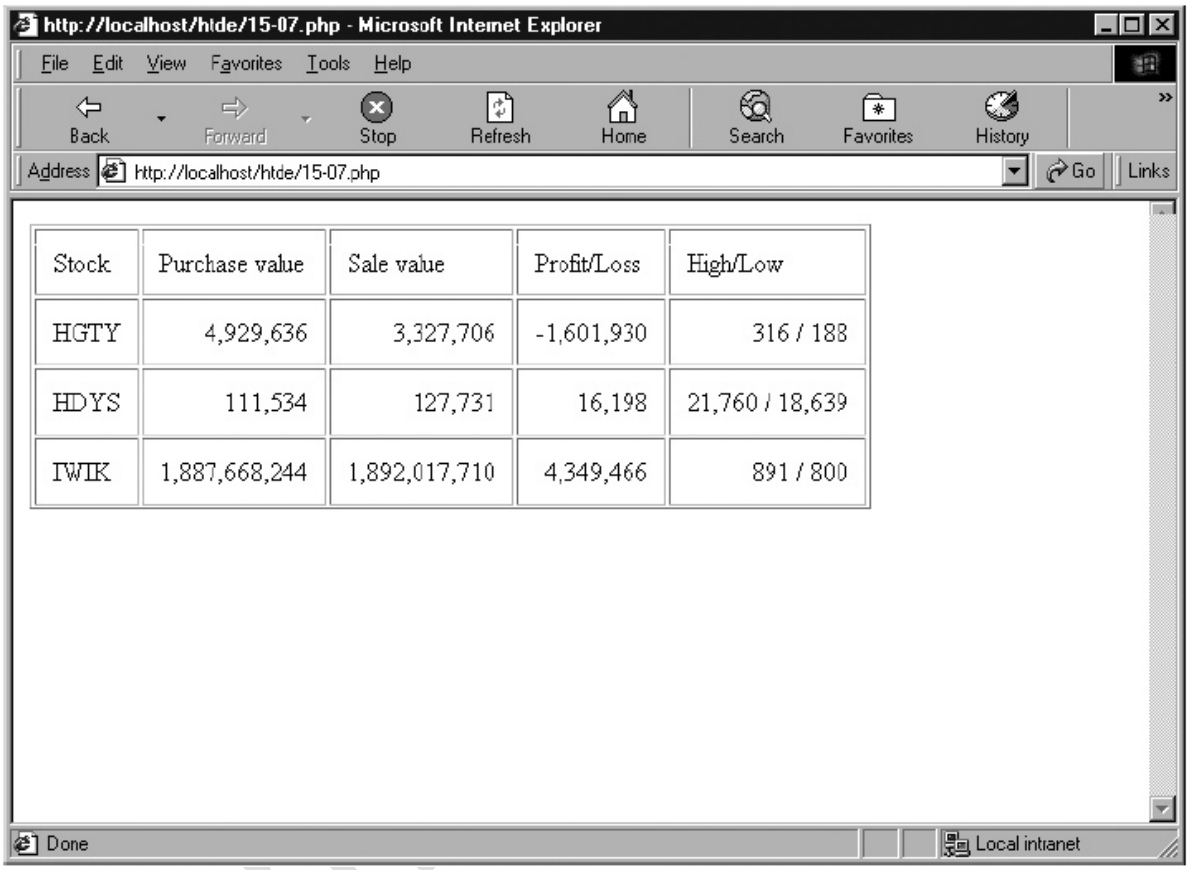
Target Selection

The sprintf() function returns the result of output formatting, while theprintf() function prints the result directly to the standard output device.

```
<?php
// open connection to MySQL server
$connection = mysql_connect('localhost', 'guest', 'pass') ↲
or die ('Unable to connect!');
// select database for use
mysql_select_db('db2') or die ('Unable to select database!');
// create and execute query
$query = "SELECT * FROM stocks";
$result = mysql_query($query) ↲
or die ('Error in query: $query. ' . mysql_error());
// check if records were returned
if (mysql_num_rows($result) > 0)
{
echo '<table border=1 cellpadding=10>';
echo '<tr><td>Stock</td><td>Purchase value</td>';
echo '<td>Sale value</td><td>Profit/Loss</td>';
echo '<td>High/Low</td></tr>';
// iterate over record set
// format and print numeric data
while($row = mysql_fetch_object($result))
{
echo '<tr>';
echo '<td>' . $row->symbol . '</td>';
printf('<td align=right>%s</td>', ↲
number_format($row->qty * $row->buy));
printf('<td align=right>%s</td>', ↲
number_format($row->qty * $row->sell));
printf('<td align=right>%s</td>', ↲
number_format($row->qty * ($row->sell - $row->buy)));
printf('<td align=right>%s / %s</td>', ↲
number_format($row->high), number_format($row->low));
echo '</tr>';
}
echo '</table>';
}
else
{
// print error message
echo 'No rows found!';
```

```
}
// once processing is complete
// free result set
mysql_free_result($result);
// close connection to MySQL server
mysql_close($connection);
?>
</body>
</html>
```

Figure shows the output of this script.



Stock	Purchase value	Sale value	Profit/Loss	High/Low
HGTY	4,929,636	3,327,706	-1,601,930	316 / 188
HDYS	111,534	127,731	16,198	21,760 / 18,639
IWIK	1,887,668,244	1,892,017,710	4,349,466	891 / 800

Formatting Currency Values

At this point, it's appropriate to mention PHP's money_format() function,introduced in PHP 4.3.0. This function is designed specifically for use with currency

Rounding Off

If you have a decimal value that you need to round up or down, you can doit using either PHP or MySQL. MySQL offers the CEIL() and FLOOR()functions, while PHP offers the round(), ceil(), and floor() functions.

Take a look at the following examples to see how these functions work:

```
mysql>SELECT CEIL(12.052),
FLOOR(12.052);
+--------------+--------------+
| ceil(12.052) | floor(12.052)|
+--------------+--------------+
| 13 | 12|
+--------------+--------------+
```

```
1 row in set (0.00 sec)
<?php
// returns 13
echo ceil(12.052);
// returns 12
echo floor(12.052);
// returns 12.1
// the second argument specifies ↲
// the number of decimals to round to
echo round(12.052, 1);
?>
```

values, and it formats numbers in accordance with local or international conventions for currency display.

The money_format() function is not available in the Windows version of PHP.

To see how this works, consider the following revision of a previous script, which formats account balances using American, Indian, and French conventions:

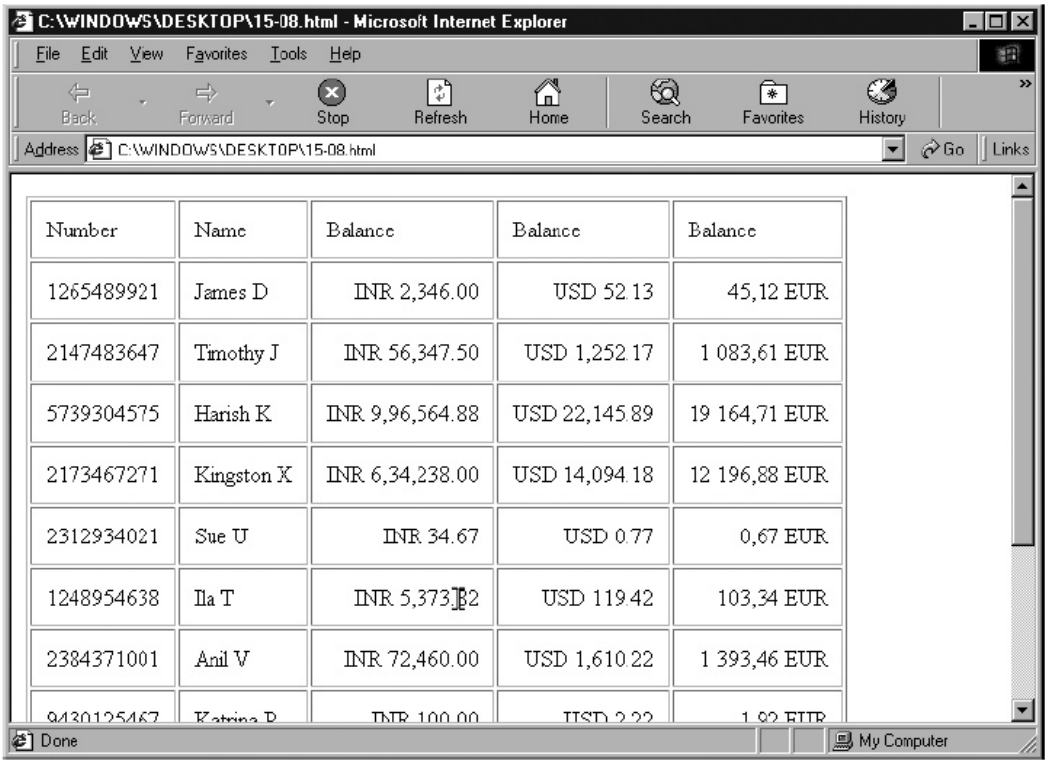
```
<html>
<head></head>
<body>
<?php
// open connection to MySQL server
$connection = mysql_connect('localhost', 'guest', 'pass') ↲
or die ('Unable to connect!');
```

Formatting numbers with the printf() function

```
// select database for use
mysql_select_db('db2') or die ('Unable to select database!');
// create and execute query
$query = "SELECT accountNumber, accountName, accountBalance ↲
FROM accounts";
$result = mysql_query($query) ↲
or die ('Error in query: $query. ' . mysql_error());
// check if records were returned
if (mysql_num_rows($result) > 0)
{
echo '<table border=1 cellpadding=10>';
echo '<tr><td>Number</td><td>Name</td><td>Balance</td>
<td>Balance</td><td>Balance</td></tr>';
// iterate over record set
while($row = mysql_fetch_object($result))
{
echo '<tr>';
echo '<td>' . $row->accountNumber . '</td>';
echo '<td>' . $row->accountName . '</td>';
// display in Indian rupees
setlocale(LC_MONETARY, 'en_IN');
echo '<td align=right>' . ↲
money_format('%i', $row->accountBalance) . '</td>';
```

```
// display in US dollars (convert using 1 USD = 45 INR)
setlocale(LC_MONETARY, 'en_US');
echo '<td align=right>' . ↲
money_format('%i', $row->accountBalance/45) . '</td>';
// display in euros (convert using 1 EUR = 52 INR)
setlocale(LC_MONETARY, 'fr_FR');
echo '<td align=right>' . ↲
money_format('%i', $row->accountBalance/52) . '</td>';
echo '</tr>';
}
echo '</table>';
}
else
{
// print error message
echo 'No rows found!';
}
// once processing is complete
// free result set
mysql_free_result($result);
// close connection to MySQL server
mysql_close($connection);
?>
</body>
</html>
```

Figure demonstrates what the output looks like.



Formatting numbers with the money_format() function

Here, the money_format() function formats numeric values as per international currency conventions, using the appropriate separators. As the output illustrates, the French locale uses commas instead of decimals and spaces instead of commas, while the Indian locale differs from the American locale in its placement of thousand separators. Locale information is set with PHP's setlocale() function, and numerous adjustments can be made to the alignment and precision of the final currency value using sprintf()-type field templates (you can obtain a complete list of these from **http://www.php.net/manual/en/function.money-format.php**).
To display the national currency symbol instead of the three-letter international currency code, replace the %i symbol in the call to money_format() with %n.

Formatting Dates and Times

You can use PHP's mktime() function to obtain a UNIXtimestamp for any arbitrary date/time value. However, because the timestamp returnedby mktime() does not resemble traditional date/time displays, it is usually necessaryto format this timestamp, so it isunderstandable to humans. This is particularly true inweb applications, where dates and times are frequently displayed in human-readable,rather than machine-readable, form. To this end, PHP offers the date() function,which accepts two arguments: one or more *format specifiers,* which indicates howthe timestamp should be formatted, and the timestamp itself (optional; PHP assumesthe current time if this second argument is not provided).

To see a few examples of the date() function in action, create and run thefollowing script:

```
<?php
// retrieve current date and time
// prints a date and time like "09:18 pm 19 Jun 2004"
echo date("h:i a d M Y", mktime());
// returns just the date "27 April 2003"
echo date("d F Y", mktime(0, 0, 0, 04, 27, 2003));
// returns the time in 24-hr format "21:18"
echo date("H:i", mktime());
?>
```

Table 15-2 lists some of the more useful format specifiers recognized by thedate() function.
Let's see an example of this in action. Consider the following database table,which holds a list of users and their birth dates:

```
mysql>SELECT * FROM birthdays;
+-------+------------+
| name | dob |
+-------+------------+
| raoul | 1978-06-04 |
| luis | 1970-11-17 |
| larry | 1971-08-19 |
| moe | 1992-01-23 |
+-------+------------+
4 rows in set (0.00 sec)
```

Specifier What It Means
d Day of the month; numeric
D Day of the week; short string
F Month of the year; long string
h Hour; numeric 12-hour format

H Hour; numeric 24-hour format
i Minute; numeric
l Day of the week; long string
L Boolean indicating whether it is a leap year
m Month of the year; numeric
M Month of the year; short string
s Seconds; numeric
T Timezone
Y Year; numeric
z Day of the year; numeric

Common Format Specifiers Supported by the date() Function

Now, create and run a PHP script to retrieve these dates and format them into more readable values:

```
<html>
<head></head>
<body>
<?php
// open connection to MySQL server
$connection = mysql_connect('localhost', 'guest', 'pass') ↲
or die ('Unable to connect!');
// select database for use
mysql_select_db('db2') or die ('Unable to select database!');
// create and execute query
$query = 'SELECT name, UNIX_TIMESTAMP(dob) AS dob ↲
FROM birthdays';
$result = mysql_query($query) ↲
or die ('Error in query: $query. ' . mysql_error());
// check if records were returned
if (mysql_num_rows($result) > 0)
{
// print HTML table
echo '<table border=1 cellpadding=10>';
// iterate over record set
// print each field
while($row = mysql_fetch_object($result))
{
echo '<tr>';
echo "<td>$row->name</td><td>" . ↲
date("d M Y", $row->dob) . "</td>";
echo '</tr>';
}
echo '</table>';
}
else
{
// print error message
echo 'No rows found!';
}
// once processing is complete
```

```
// free result set
mysql_free_result($result);
// close connection to MySQL server
mysql_close($connection);
?>
</body>
</html>
```

Figure 15-9 demonstrates the output.

MySQL isn't far behind either: the RDBMS comes with powerful DATE_FORMAT() and TIME_FORMAT() functions to manipulate the display of dateand time values until they're exactly the way you want them. As with the PHPdate() function, format specifiers are used to control the appearance of theoutput.

Table 15-3 demonstrates the specifiers supported by the DATE_FORMAT()and TIME_FORMAT() functions.

Here are some examples demonstrating these in action:

```
mysql>SELECT DATE_FORMAT(NOW(), '%W, %D %M %Y %r');
+----------------------------------------+
| DATE_FORMAT(NOW(), '%W, %D %M %Y %r') |
+----------------------------------------+
| Thursday, 18th November 2004 12:07:55 PM |
+----------------------------------------+
1 row in set (0.22 sec)
```

Just in Time

The MySQL UNIX_TIMESTAMP() function converts a MySQL-compliantdate or time value into a UNIX timestamp suitable for use with the PHPdate() function.

```
mysql>SELECT DATE_FORMAT(19980317, '%d/%m/%Y');
+----------------------------------+
| DATE_FORMAT(19980317, '%d/%m/%Y') |
+----------------------------------+
| 17/03/1998 |
+----------------------------------+
1 row in set (0.00 sec)
mysql>SELECT DATE_FORMAT("20011215101030", ↵
"%H%i hrs on %a %d %M %y");
+-----------------------------------------------------+
| DATE_FORMAT("20011215101030", "%H%i hrs on %a %d %M %y") |
+-----------------------------------------------------+
| 1010 hrs on Sat 15 December 01 |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

Formatting dates with the date() function

```
mysql>SELECT TIME_FORMAT(19690609140256, '%h:%i %p');
+---------------------------------------+
| TIME_FORMAT(19690609140256, '%h:%i %p') |
+---------------------------------------+
| 02:02 PM |
+---------------------------------------+
1 row in set (0.00 sec)
```

Using the DATE_FORMAT() function, you can perform date formattingwithin your SQL query itself, without needing PHP's date() function. Thisnext script revisits the previous PHP listing, moving the formatting task to thedatabase layer:

```
<html>
<head></head>
<body>
Symbol What It Means
%a Short weekday name (Sun, Mon . . .)
%b Short month name (Jan, Feb . . .)
%d Day of the month
%H Hour (01, 02 . . .)
%I Minute (00, 01 . . .)
%j Day of the year (001, 002 . . .)
%m 2-digit month (00, 01 . . .)
%M Long month name (January, February . . . .)
%p AM/PM
%r Time in 12-hour format
%S Second (00, 01 . . .)
%T Time in 24-hour format
%w Day of the week (0,1 . . .)
%W Long weekday name (Sunday, Monday . . .)
%Y 4-digit year
                MySQL Date/Time Formatting Codes
// open connection to MySQL server
$connection = mysql_connect('localhost', 'guest', 'pass') ↲
or die ('Unable to connect!');
// select database for use
mysql_select_db('db2') or die ('Unable to select database!');
// create and execute query
$query = "SELECT name, DATE_FORMAT(dob, '%d %b %Y') ↲
AS dob FROM birthdays";
$result = mysql_query($query) ↲
or die ('Error in query: $query. ' . mysql_error());
// check if records were returned
if (mysql_num_rows($result) > 0)
{
// print HTML table
echo '<table border=1 cellpadding=10>';
// iterate over record set
// print each field
while($row = mysql_fetch_object($result))
{
echo '<tr>';
echo "<td>$row->name</td><td>$row->dob</td>";
echo '</tr>';
}
echo '</table>';
}
else
```

```
{
// print error message
echo 'No rows found!';
}
// once processing is complete
// free result set
mysql_free_result($result);
// close connection to MySQL server
mysql_close($connection);
?>
</body>
</html>
```

Calculating Your Age with MySQL

MySQL comes with a comprehensive date/time manipulation API that lets youperform complex date arithmetic and extraction. While the list of availablefunctions is too large to list, two of the more interesting ones are thePERIOD_DIFF() and TO_DAYS() functions,which return the difference,in months and days, respectively, between two date values.

To see how this works, consider the following variant of the previous listing,which lists the current age of each user in the table, given their date of birth:

```
mysql>SELECT name, dob, ↲
ROUND(PERIOD_DIFF(DATE_FORMAT(NOW(), '%Y%m'), ↲
DATE_FORMAT(dob, '%Y%m')) / 12, 1) AS age ↲
FROM birthdays;
+-------+------------+------+
| name | dob | age |
+-------+------------+------+
| raoul | 1978-06-04 | 26.4 |
| luis | 1970-11-17 | 34.0 |
| larry | 1971-08-19 | 33.2 |
| moe | 1992-01-23 | 12.8 |
+-------+------------+------+
4 rows in set (0.06 sec)
mysql>SELECT name, dob, ↲
(TO_DAYS(NOW()) - TO_DAYS(dob)) / 365 AS age ↲
FROM birthdays;
+-------+------------+------+
| name | dob | age |
+-------+------------+------+
| raoul | 1978-06-04 | 26.48 |
| luis | 1970-11-17 | 34.03 |
| larry | 1971-08-19 | 33.28 |
| moe | 1992-01-23 | 12.83 |
+-------+------------+------+
4 rows in set (0.00 sec)
```

There’s also an entire family of functions designed to extract each componentof a timestamp separately. Take a look at Table 15-4, which has a list, and the examplesfollowing it to see how these work.

Here are some examples of these in action:

mysql>**SELECT DAYOFMONTH(NOW()), DAYOFYEAR('1979-01-02');**

```
+------------------+-----------------------+
| DAYOFMONTH(NOW()) | DAYOFYEAR('1979-01-02') |
+------------------+-----------------------+
| 23 | 2 |
+------------------+-----------------------+
```

1 row in set (0.00 sec)

mysql>**SELECT DAYNAME(NOW()), MONTHNAME(NOW()), YEAR(NOW());**

```
+---------------+-----------------+-------------+
| DAYNAME(NOW()) | MONTHNAME(NOW()) | YEAR(NOW()) |
+---------------+-----------------+-------------+
| Tuesday | November | 2004 |
+---------------+-----------------+-------------+
```

1 row in set (0.00 sec)

Function What It Does

DAYOFWEEK() Returns a number (1 to 7) representing the day of the week for a date

DAYOFMONTH() Returns the day component (1 to 31) of a date

DAYOFYEAR() Returns a number (1 to 366) representing the day of the year for a date

DAYNAME() Returns the weekday name for a date

HOUR() Returns the hour component (0–23) of a time

MINUTE() Returns the minute component (0–59) of a time

MONTH() Returns the month component (1 to 12) for a date

MONTHNAME() Returns the month name for a date

QUARTER() Returns the quarter (1–2) in which a date falls

WEEK() Returns the week number (0–53) for a date

YEAR() Returns the year component (1000–9999) of a date

More MySQL Date Functions

mysql>**SELECT HOUR(NOW()), MINUTE('14:36');**

```
+-------------+-----------------+
| HOUR(NOW()) | MINUTE('14:36') |
+-------------+-----------------+
| 21 | 36 |
+-------------+-----------------+
```

1 row in set (0.05 sec)

Read more about these functions at **http://dev.mysql.com/doc/mysql/en/**

Date_and_time_functions.html.

Paginating Large Result Sets

In previous sections of this chapter, you've seen how to message and reformatindividual records so they meet your display requirements. In this concludingsegment, it's time to step back and understand how to better present the entire setof records returned by an SQL query.

It's not uncommon for query result sets to contain hundreds or even thousandsof records. In such cases, it's usually not user friendly to display the entire resultset on a single HTML page, as doing so forces the user to scroll up and downendlessly to view the results. This is where *pagination*—the act of breaking uplarge record collections into smaller subsets and displaying them one page ata time—can help. By breaking the large mass of data into smaller, more easilynavigable pages, you increase the usability of your application, and you also avoidoverwhelming the user with mountains of data at once.

Writing PHP code to paginate a MySQL result set is fairly simple and groundedin common sense. First, you decide how many results you want to display at atime, say, ten. Next, you COUNT() how many records exist in the result set. Nowyou can extract the first ten records using a LIMIT clause, and provide a link toenable the user to select the next ten records. On each “page,” the last record identifierserves as the starting point for the next page of records. This continues until all therecords in the collection have been processed.

Once you understand the underlying principle, writing the code to implement itis simple. Take a look at the next listing, which illustrates this:

```
<html>
<head>
<basefont face="Arial">
</head>
<body>
<?php
// number of records to be displayed per page
$records_per_page = 10;
// look for starting marker
// if not available, assume 0
(!$_GET['start']) ? $start = 0 : $start = $_GET['start'];
// open connection to MySQL server
$connection = mysql_connect('localhost', 'guest', 'pass') ↲
or die ('Unable to connect!');
// select database for use
mysql_select_db('db2') or die ('Unable to select database!');
// create and execute query to count records
$query = "SELECT COUNT(*) FROM books WHERE rating > 3";
$result = mysql_query($query) ↲
or die ('Error in query: $query. ' . mysql_error());
// get total number of records
$row = mysql_fetch_row($result);
$total_records = $row[0];
// if records exist
if (($total_records > 0) && ($start < $total_records))
{
// create and execute query to get batch of records
$query = "SELECT title, author, DATE_FORMAT(date, '%d %M %Y') ↲
AS date FROM books WHERE rating > 3 LIMIT $start, $records_per_page";
$result = mysql_query($query) ↲
or die ('Error in query: $query. ' . mysql_error());
// iterate over record set
// print data
echo '<table border=1 cellpadding=10>';
while($row = mysql_fetch_object($result))
{
echo '<tr>';
echo "<td>$row->title</td>";
echo "<td>$row->author</td>";
echo "<td>$row->date</td>";
echo '</tr>';
```

```
}
echo '</table>';
// set up the previous page link
// this should appear on all pages except the first page
// the start point for the previous page will be
// the start point for this page
// less the number of records per page
if ($start >= $records_per_page)
{
echo "<a href=" . $_SERVER['PHP_SELF'] . ↵
"?start=" . ($start-$records_per_page) . ">Previous ↵
Page</a>      ";
}
// set up the "next page" link
// this should appear on all pages except the last page
// the start point for the next page
// will be the end point for this page
if ($start+$records_per_page < $total_records && $start >= 0)
{
echo "<a href=" . $_SERVER['PHP_SELF'] . ↵
"?start=" . ($start+$records_per_page) . ">Next Page</a>";
}
}
?>
</body>
</html>
```

In this listing, the key variable is $records_per_page, which controls the number of records displayed at one time. This value is used as the upper boundary in the SELECT query's LIMIT clause to restrict the number of records returned by the query (the lower boundary is the number of the last record displayed, received from the previous instance of the page through the URL GET method).

For each batch of records, further calculations are performed to assess if "next page" and "previous page" links should be displayed. The presence of these links is heavily dependent on the interaction between the total number of records in the result set, and the number of records to be displayed at any one time.

Sample Database applications

By building a real-world application that retrieves data from a MySQL database to create a dynamic PHP-based web site. This application is more challenging than the examples you've seen in previous chapters. Once you complete this exercise, however, you will have practical, hands-on knowledge of how to use PHP and MySQL together to build usable web applications.

Understanding Requirements

The application here is a news publishing system for a business web site, either on an intranet or the public Internet. It's intended to provide the organization's administrative and press personnel with a way to post news items, press releases, and articles on the web site, and to easily maintain (view, edit, and delete) this information. A MySQL database stores this

information, with PHP taking care of retrieving and manipulating the information through a web browser.

This application has two pieces: the “public” piece, which consists of the code that displays the latest news and press releases to the site’s visitors, and the “private” piece, which consists of the administration interface for individual editors within the PR department to publish new content to the web site. Both these pieces interact with the MySQL database (which contains the actual news stories and press releases) using the MySQL API built into PHP.

With this in mind, it should be clear that this application must support the following tasks.

- It must be able to display a list of all news items in the database (or the mostrecent ones), and enable users to view the complete contents of each.
- It must let administrators add new items and press releases to the database.
- It must enable administrators to edit existing releases, to make correctionsor update them with new information.
- It must permit the removal of older, out-of-date releases, and news itemsfrom the database.

With these requirements in mind, it’s time to design the database.

Designing the Database

Because the content for the application is stored in a MySQL table, it’s important to define exactly what constitutes a press release. If you think about it, you’ll see that a *press release* or article can typically be broken down into three subsections: a title, a main body containing the text of the press release or news item, and an information section with the publication date and name of the contact person.

To begin, create a database to store this information, and select it for use:

```
mysql>CREATE DATABASE news;
Query OK, 1 row affected (0.16 sec)
mysql>USE news;
Database changed
```

Next, create a table to hold press releases and news:

```
mysql>CREATE TABLE news (
->id SMALLINT(5) unsigned NOT NULL auto_increment,
->title TEXT NOT NULL,
->content TEXT NOT NULL,
->contact VARCHAR(255),
->timestamp DATETIME DEFAULT '0000-00-00 00:00:00'
->NOT NULL,
->PRIMARY KEY (id)
->);
Query OK, 0 rows affected (0.05 sec)
```

As you can see, this maps right into the information provided previously. The table has one field for every element of a press release.

To get things rolling, populate this table with a couple of dummy records, like the following ones:

```
mysql>INSERT INTO news (id, title, content, contact, timestamp) VALUES ( '1',
'Megalomaniacs Inc. Is Born', 'EARTH -- A new star was born today on the planet third closest
to the sun. Megalomaniacs Inc., a venture of WeWantItAll Corp., today threw open its doors
```

for business in the ritzy Jefferson Square business district. Created with the sole goal of colonizing every single planet in the known Universe (and beyond), Megalomaniacs Inc. hopes to quickly acquire a monopoly over the vast tracts of uncharted real estate in space. Speaking at a press conference, Megalomaniacs Inc. CEO warned reporters that Megalomaniacs Inc. would "take everything it could, and then some". ', 'Peter Paul (peter@megalo.mania)', '2003- 12-11 17:29:25');
Query OK, 1 row affected (0.01 sec)
mysql>**INSERT INTO news (id, title, content, contact, timestamp) VALUES ('2', 'Megalomaniacs Inc. Expands To Mars', 'MARS -- As part of its business strategy of "expand and swallow", Megalomaniacs Inc. today announced that it had successfully sent a team of corporate raiders to Mars, in an effort to persuade the inhabitants of that**
planet to surrender their planet for colonization. Megalomaniacs Inc. COO today said that the move was a "friendly overture", but that a failure to comply with the company\'s
colonization plans would result in a "swift and sure eviction of those little green guys". ', 'Tim Jr. (tim@megalo.mania)', '2004-08-30 12:13:48');
Query OK, 1 row affected (0.07 sec)

Listing and Displaying News Items

You'll remember from the requirements discussion a couple of pages back, that this development effort can broadly be split into two parts. One part consists of the scripts that retrieve the list of newest items from the database and display this list to the user. The other part consists of administrative tools that enable editors to manage this list, enter new information, and edit or delete existing information. Because the first part is simpler, let's get that out of the way first. Two scripts are involved here: *list.php,* which retrieves a list of the five newest entries in the database; and *story.php,* which displays the full text for the selected story.

Listing News Items

Create *list.php* first:

```
<html>
<head>
<basefont face="Verdana">
</head>
<body>
<!-- standard page header begins -->
<p> <p>
<table width="100%" cellspacing="0" cellpadding="5">
<tr>
<td></td>
</tr>
<tr>
<td bgcolor="Navy"><font size="-1" color="White">
<b>Megalomaniacs Inc : Press Releases</b></font>
</td>
</tr>
</table>
<!-- standard page header ends -->
<ul>
```

```
<?php
// includes
include('../conf.php');
include('../functions.php');
// open database connection
$connection = mysql_connect($host, $user, $pass) ↲
or die ('Unable to connect!');
// select database
mysql_select_db($db) or die ('Unable to select database!');
// generate and execute query
$query = "SELECT id, title, timestamp FROM news ↲
ORDER BY timestamp DESC LIMIT 0, 5";
$result = mysql_query($query) ↲
or die ("Error in query: $query. " . mysql_error());
// if records present
if (mysql_num_rows($result) > 0)
{
// iterate through resultset
// print article titles
while($row = mysql_fetch_object($result))
{
?>
<li><font size="-1"><b><a href="story.php?id= ↲
<?php echo $row->id; ?>"><?php echo $row->title; ?></a></b></font>
<br>
<font size="-2"><?php echo formatDate($row->timestamp); ?>
</font>
<p>
<?php
}
}
// if no records present
// display message
else
{
?>
<font size="-1">No press releases currently available</font>
<?php
}
// close database connection
mysql_close($connection);
?>
</ul>
<!-- standard page footer begins -->
<p>
<table width="100%" cellspacing="0" cellpadding="5">
<tr>
<td align="center"><font size="-2">
All rights reserved. Visit Melonfire
```

```
<a href="http://www.melonfire.com/community/columns/trog/">
here</a> for more.</td>
</tr>
</table>
<!-- standard page footer ends -->
</body>
</html>
```

This script connects to the database, retrieves a set of records, and formatsthem for display in a web browser. Pay special attention to the SELECT query that retrievesthe records from the MySQL table: it contains a DESC clause to order the items inthe order of most recentfirst, and a LIMIT clause to restrict the result set to fiveitems only.

The formatDate() function used in the previous code listing is a userdefinedfunction that turns a MySQL timestamp into a human-friendly date string The functionis defined in the*functions.php* file and looks like this:

```
<?php
// format MySQL DATETIME value into a more readable string
function formatDate($val)
{
$arr = explode('-', $val);
return date('d M Y', mktime(0,0,0, $arr[1], $arr[2], $arr[0]));
}
?>
```

Also necessary is to include some code that tells the script what to do if norecords are returned by the query (this could happen when the application isinstalled for the first time, and no records are present in the database). Without thiscode, the generated page would be completely empty—not a nice thing to show tousers, especially on a potentially high-traffic page. The solution is to use an if()loop to check if any records were returned by the query and display a neat littlemessage if none were returned.Here's a fragment thatoutlines how this would work:

```
<?php
// if records present
if (mysql_num_rows($result) > 0)
{
// iterate through resultset
// print article titles
}
// if no records present
else
{
// display error message
}
?>
```

Figure shows what it looks like when you view this script through a browser. As a developer, it's important to think through all possible situations and write code that handles each one intelligently. The possibility of an empty database doesn't even occur to many novice developers—and this can lead to embarrassing situations if you're demonstrating the application to your boss . . . or worse, the customer!

A list of available news items

The Configuration File

In case you're wondering, the MySQL hostname, the username, and the passwordused by the mysql_connect() function are all variables sourced from theconfiguration file *conf.php*. This file has been include()-d at the top of eachscript and it looks like this:

```
<?php
// database configuration
$host = 'localhost';
$user = 'newuser';
$pass = 'newspass';
$db = 'news';
// default contact person
$def_contact = 'Johnny Doe (jd@megalo.mania)';
?>
```

Extracting this configuration information into a separate file makes it easierto update the application in case the database username or password changes.Updating a single file is far easier than updating multiple scripts, each with thevalues hard-wired into it.

Displaying Story Content

You'll notice, from the previous code listing, that every press release title is linkedto *story.php* via its unique ID. The *story.php* script uses this ID to connect to thedatabase and retrieve the full text of the release. Here is what it looks like:

```
<html>
<head></head>
<body>
<!-- standard page header -->
<?php
// includes
include('../conf.php');
include('../functions.php');
// check for record ID
if ((!isset($_GET['id']) || trim($_GET['id']) == ''))
{
die('Missing record ID!');
}
// open database connection
$connection = mysql_connect($host, $user, $pass) ↲
or die ('Unable to connect!');
// select database
mysql_select_db($db) or die ('Unable to select database!');
// generate and execute query
$id = $_GET['id'];
$query = "SELECT title, content, contact, timestamp FROM news ↲
WHERE id = '$id'";
$result = mysql_query($query) ↲
or die ("Error in query: $query. " . mysql_error());
// get resultset as object
$row = mysql_fetch_object($result);
// print details
```

```
if ($row)
{
?>
<p>
<b><?php echo $row->title; ?></b>
<p>
<font size="-1"><?php echo nl2br($row->content); ?></font>
<p>
<font size="-2">This release was published on
<?php echo formatDate($row->timestamp); ?>.
For more information, please contact <?php echo $row->contact; ?>
</font>
<?php
}
else
{
?>
<p>
<font size="-1">That release could not be located in
our database.</font>
<?php
}
// close database connection
mysql_close($connection);
?>
<!-- standard page footer -->
</body>
</html>
```

Again, extremely simple—connect, use the ID to get the full text for thecorresponding item, and display it. Figure illustrates what it looks like.At this point, you have a primitive publishing system that can be used toprovide users of a web site with news, press releases, and other information.