Introduction to LAMP technology:

Explore the open source Web development platform

Skill Level: Intermediate

Jono Bacon (jono@jonobacon.org)
Writer, Open Source consultant

03 May 2005

This tutorial explores the Linux-Apache-MySQL-PHP, or LAMP, Web development framework and shows how that framework can help you build applications to solve common business problems. The tutorial begins with an exploration of the LAMP architecture, then introduces fundamental PHP concepts. After a solid grounding of PHP, the tutorial explains MySQL support, with coverage focusing on database concepts and how to access MySQL from PHP. All of these techniques are discussed within the context of a real-world customer management example.

Section 1. Before you start

About this tutorial

Within any business, you must be able to access information, process it, and make it available anywhere. This requirement is particularly important in the always-on world in which users expect to be able to pull up any information required from any location.

This tutorial explores the Linux-Apache-MySQL-PHP, or LAMP, Web development framework and shows how that framework can help you build applications to solve common business problems. The tutorial begins with an exploration of the LAMP architecture, then introduces fundamental PHP concepts. After a solid grounding of PHP, the tutorial explains MySQL support, with coverage focusing on database

concepts and how to access MySQL from PHP. All of these techniques are discussed within the context of a real-world customer management example.

Prerequisites

To complete this tutorial, you need the following tools:

- A working LAMP system (The easiest way to get this system up and running is through the XAMPP Apache distribution, which contains all the components necessary.
- phpMyAdmin (included with XAMPP)
- A Web browser (Mozilla Firefox is recommended)
- A text editor

Section 2. Plug into the LAMP revolution

The data organization challenge

Suppose that you work for a mid-sized organization that has a real problem managing its organizational data. You've managed all the information about customers, products, transactions, and meetings using a random system of scribbled notes that have quickly piled up on your desk. You know that you need to manage your information better, but you don't have a lot of time to dedicate to solving the problem, and you don't have time to learn a huge programming language and toolset. You need a solution quickly.

Fortunately, a great portfolio of tools is available to help you build an application that can solve your problem. This tutorial shows how to use the LAMP portfolio to solve basic data-management problems and provides the fundamental skills you need to store your data in a database and display it on a Web page.

The LAMP platform consists of four components that are structured in a layered way. Each layer provides a critical part of the entire software stack:

- **Linux.** Linux is the lowest-level layer and provides the operating system. Linux actually runs each of the other components. You are not specifically limited to Linux, however; you can easily run each of the other

components on Microsoft®; Windows®;, Mac OS X, or UNIX® if you need to.

- **Apache.** The next layer is Apache, the Web server. Apache provides the mechanics for getting a Web page to a user. Apache is a stable, mission-critical-capable server, and it runs more than 65 percent of all Web sites on the Internet. The PHP component actually sits inside Apache, and you use Apache and PHP together to create your dynamic pages.
- **MySQL.** MySQL provides the data-storage side of the LAMP system. With MySQL, you have access to a very capable database suitable for running large and complex sites. Within your Web application, all your data, products, accounts, and other types of information will reside in this database in a format that you can easily query with the SQL language.
- **PHP.** PHP is a simple and efficient programming language that provides the glue for all the other parts of the LAMP system. You use PHP to write dynamic content capable of accessing the data in the MySQL database and some of the features that Linux provides.

Get started

To start, you will build a customer information tool. Using the tools in the LAMP portfolio, you will build an application that provides a means to store customer data in a database, then you can display and manage this data on a Web page.

When you develop an application, it is important to remember that you use HTML only to lay out your pages: You cannot use HTML to perform dynamic processes such as pulling data out of a database. To provide this dynamic functionality, you have to drop PHP code into the parts of the page that you want to be dynamic. For example, later in this tutorial, you will add some customer information derived from a database to a specific part of a Web page. You will move your cursor to the section of the Web page in which you want the information to display and write the PHP code to pull the information from the database and display it. When you roll this PHP code into your pages, you add the code between <?php and ?> symbols.

Create index.php

Start by creating a new page called index.php. Then, insert the following code:

```
<html>
<body>
<h1>A simple example</h1>
This is normal HTML
```

```
<?php
  echo "This is PHP!";
?>

</body>
</html>
```

In this example, you can see a special PHP block inside some HTML tags. Within the <?php and ?> symbols, you can also see your first PHP command, `echo`. This command simply displays the text within double quotation marks in the browser. Also note that every PHP command ends with a semicolon (;). When `echo` sends text going to the browser, you can also use HTML tags within the double quotation marks. For example, you can make PHP bold by using the `<strong>` HTML tag:

```
echo "This is <strong>PHP</strong>";
```

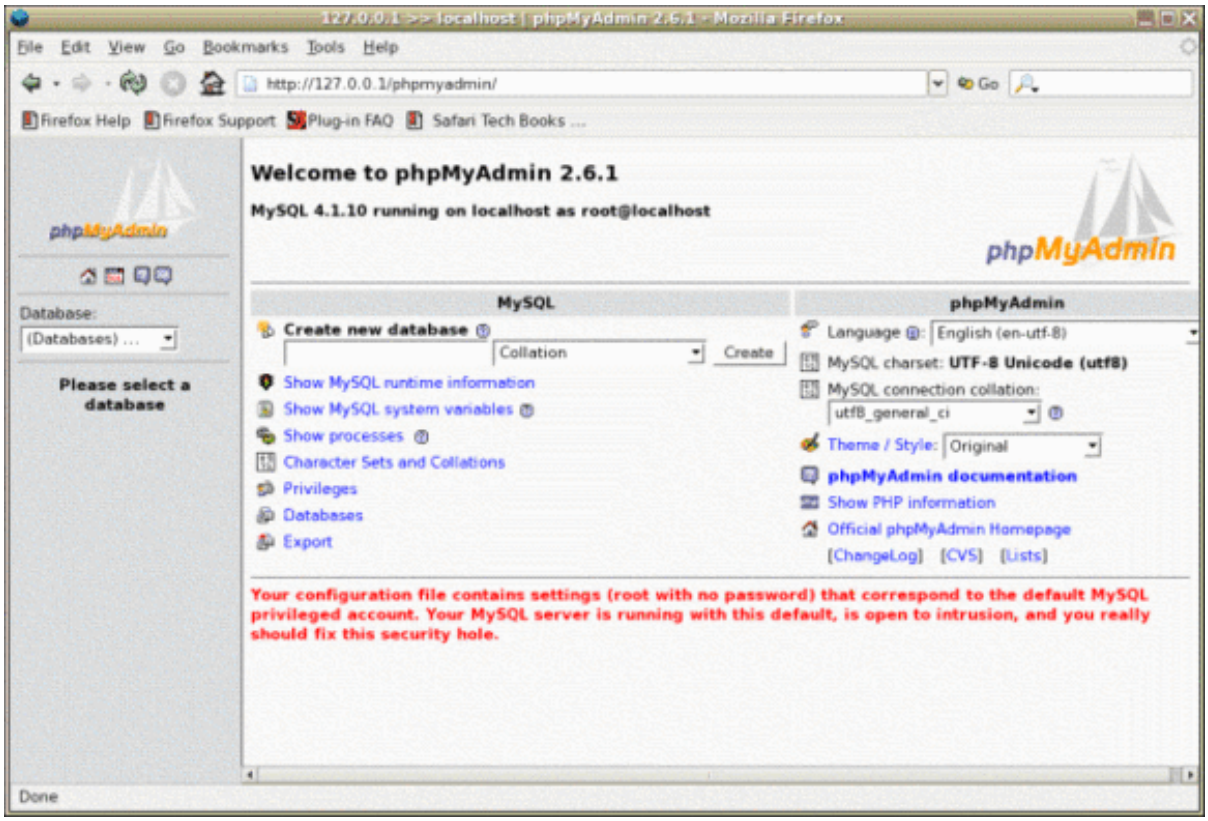
Section 3. Create the MySQL database

The phpMyAdmin client

Before you can create your Web application, you must first create the database where your customer information will reside. You will use this database to store the data, display it, and more. Ushering MySQL support into your application requires a few steps. First, you have to create a database as well as the data to go into it, which is where the phpMyAdmin client comes in.

The phpMyAdmin client provides a Web interface (see Figure 1) through which you can manage every aspect of a MySQL database, including managing users, creating databases, adding tables, inserting data, and creating relationships. SQL controls virtually every aspect of the MySQL database.

Figure 1. The phpMyAdmin Web interface



Create the MySQL database

When you have loaded and logged in to phpMyAdmin, perform the following steps to create your database:

1. Type `customer` in the **Create new database** text box to create a database of that name, then click **Create**. The new database appears in the left panel.
2. Create a new table by entering `customerlist` as the name of the table and 3 as the number of fields.

The design window appears in which you can configure the three fields in your table. Each field has several options across a row. In this table, you need to create the following fields:

- **id.** This field provides a unique ID for each record in the table. It is always a good idea to have a unique field that you can use to identify a record. The easiest way to achieve this is by creating a numeric field called ID and ensuring that there is a different number for each record. MySQL can

help you with this by automating the addition of a unique number.

- **forename.** This field contains a forename for a customer in the table.
- **surname.** This field contains the surname. I recommend keeping the first and last names separate, because the more abstract your data is, the greater the flexibility of your database. (For more information about data abstraction and referential integrity, see Resources.)

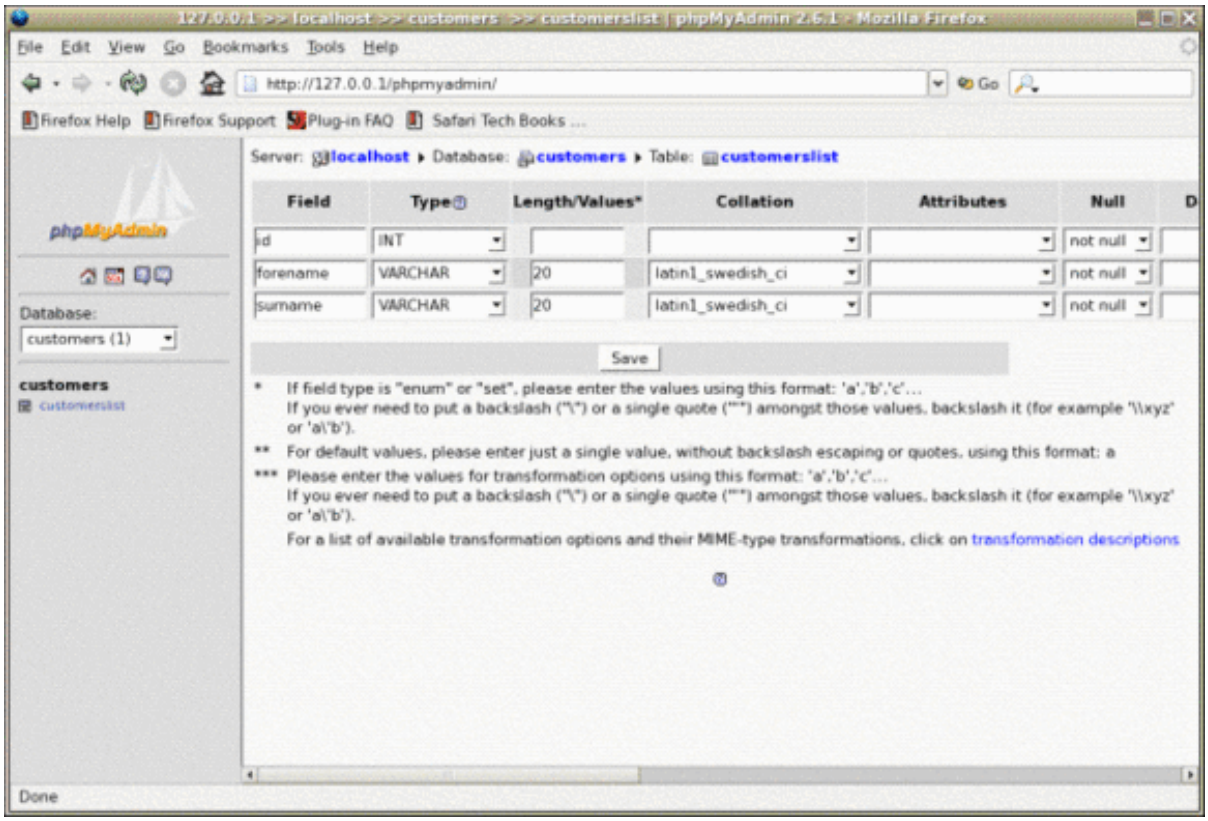
Fill out the fields

Filling out these fields in phpMyAdmin is simple. Simply add the following information, starting with the top row:

- In the first row, type `id` in the **Field** box, and set the **Type** to `INT`. Select **auto_increment** from the **Extra** drop-down list. (The auto_increment option sets MySQL to update the id field for you and adds a new number into the field each time you add a record.) Finally, select the **PRIMARY KEY** option at the end of the row to prevent MySQL from allowing a duplicate value in this field.
- In the second row, type `forename` in the **Field** box, set the **Type** to `VARCHAR`, and set the **Length** to `20`. In this field, you are setting the data type to that of a variable character to ensure that the field will be no longer than 20 characters but will take up only the right amount of memory as data is added.
- The third row is identical to the second row, except it has `surname` in the **Field** box.

Click **Save** to create your table (see Figure 2).

Figure 2. Create your MySQL table



View the SQL output

An important point to note about phpMyAdmin is that whenever you perform any interactions in the interface, the SQL that was executed to perform the action is displayed to you. This behavior is a fantastic method of learning the nuts and bolts of SQL, so always make a point of observing this output to see what SQL command was used to perform your request. In the case of the table you just created, you get the following SQL:

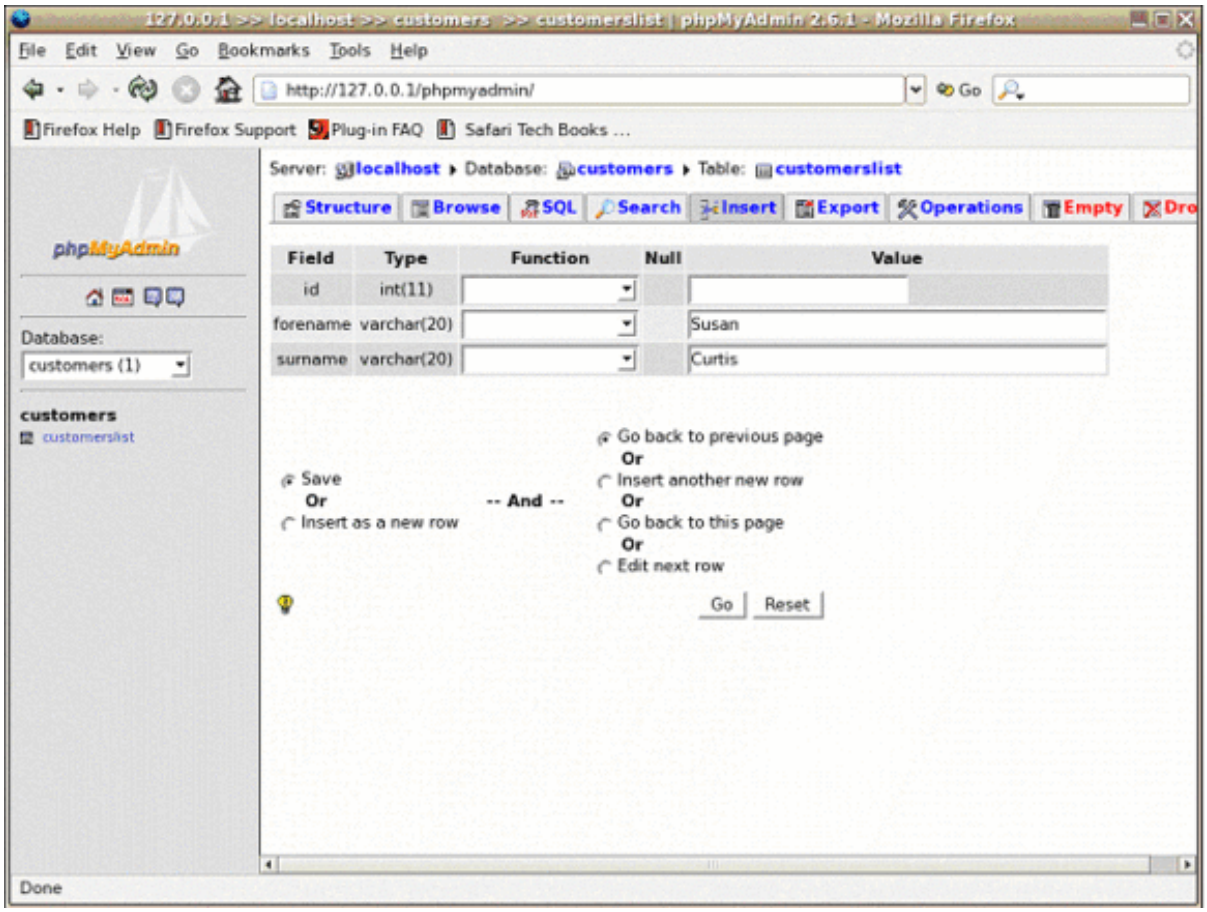
```
CREATE TABLE `customerslist` (
`id` INT NOT NULL AUTO_INCREMENT ,
`forename` VARCHAR( 20 ) NOT NULL ,
`surname` VARCHAR( 20 ) NOT NULL ,
PRIMARY KEY ( `id` )
);
```

Add data to the table

Now you have to add some data to the table you've created. To do so, click the Insert tab. The phpMyAdmin interface displays two forms in which you can add two records (see Figure 3). You don't actually need to add two records, but having two

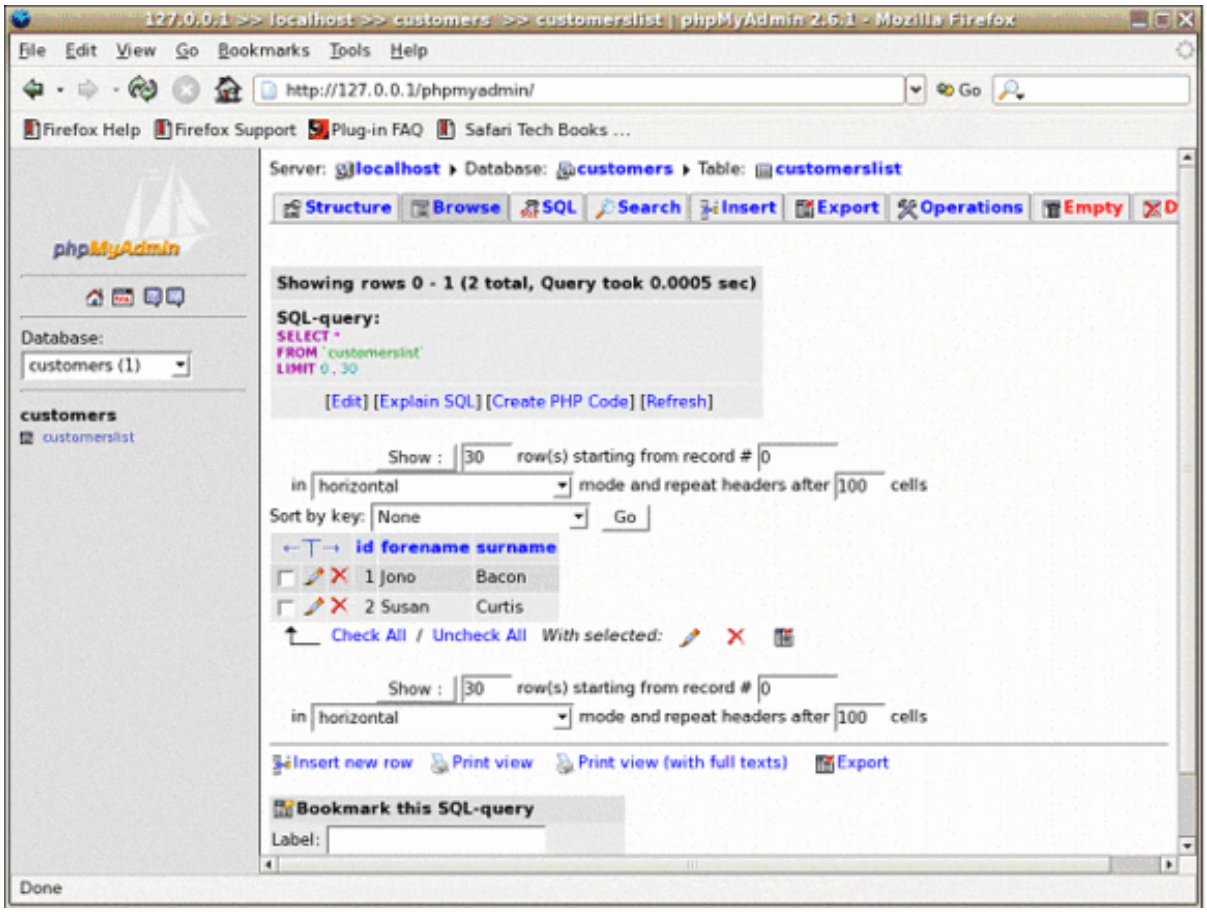
forms is handy when you need to add a lot of data. Fill in the **forename** and **surname** fields, and remember not to add anything to the **id** field; the auto_increment option ensures that this field is filled for you. Add several records to make your database appear full of customers.

Figure 3. Insert records in the database table



When you have added the records, click the Browse tab to see the contents of your table, which looks similar to Figure 4.

Figure 4. Browse database records



Section 4. Access the database through PHP

Connect to MySQL in PHP

With a table created, you're set to roll up your collaborative sleeves and get MySQL hooked into PHP. The first step in performing this process is to make a connection to the MySQL server. When you have made the connection, you're ready to interact with the server.

To begin, create four following variables to contain the details about where the database is and how to connect to it. These variables are `db_host`, `db_username`, `db_password`, and `db_database`. Note that variables in PHP begin with a dollar sign ($):

```
$db_host =
"localhost";
$db_username
= "bob";
$db_password
=
"sum65for!";
$db_database
=
"customers";
```

These four variables contain important information about your database connection. The first variable, `$db_host`, indicates where the MySQL server is located. (This location is probably localhost, unless you are running the server on a separate networked machine.) The `$db_username` and `$db_password` variables contain the authentication details for the connection (in this case, a user called *bob* with *sum65for!* for a password). Finally, you need to indicate which database on the MySQL server you want to deal with; `$db_database` specifies this database as the customers database.

Perform the connection

Remember that at this point, you have not actually connected to the database; you have merely created some variables with the relevant information. To connect to the database, run the following two lines of code:

```
$db = mysql_connect($db_host, $db_username, $db_password);
mysql_select_db($db_database, $db);
```

The first line actually creates the connection. It uses the `mysql_connect` function to pass the host, username, and password information from the variables to the server. The result of this connection is stored in the `$db` variable. In the second line of code, you then use the `mysql_select_db` function to indicate which database on the server you want to use. To do this, pass the `$db_database` variable, which contains your chosen database, and indicate the connection that you want to choose that database from (`$db`). When you have entered these two lines, you should have a successful connection.

Perform a query

With a successful connection created, the next step is actually to ask the database for some information so that you can use it in a useful way. In this project, you want to pull the customer information from the database and display it on the screen.

The first step in performing a query is to create the SQL. Underneath the database

connection code that you just wrote, add the following line:

```
$sql = "SELECT * FROM customers;";
```

In this line, you select all values from the customers table. You can read this command from left to right as such: Select (`SELECT`) all values (`*`) from (`FROM`) the customers table (`customers`) and then end the query (`;`). Again, at this point in the script, the query has not actually been executed: You have only put the SQL query into a variable. To actually send the query to the database, use the following command:

```
$result = mysql_query($sql);
```

In this command, you use the `mysql_query` function to send the `$sql` variable to the server. The result of this query (a collection of information containing the results from your query) is then made available in the `$result` variable. The actual data that is stored in the `$result` variable is called a *Record Set,* and it provides a container full of information with your results.

Key-value pairs and arrays

When you refer to data from a record set, you reference it as a *key-value pair.* This concept is important, and I'm going to digress slightly to discuss these key-value pairs before you pull the data out of the database.

The idea of a key-value pair is that you have a series of keys with which a value is associated. As an example, you may want to store a list of favorite choices and associate the `blue` value with the `colour` key and the `10` value with the `number` key. Using these values, if you look up `colour` in the array, you will want to return `blue`. You can set up this and any other type of array, with the `array()` function:

```
$array = array("colour" => "blue", "number" => 10);
```

In this line of code, you set the `colour` element to the `blue` value and the `number` element to the `10` value. You can now refer to the `colour` key with this line of code:

```
echo $array['colour'];
```

When you run this line of code, you see how `blue` is outputted, which provides a convenient method of storing information about specific things in a key-value pair on your page. This setup makes arrays suitable for storing details such as configuration

information and preferences.

If you want to iterate through the array again with `foreach`, use the following code:

```
foreach($array as $key => $value) {
  echo $key . " contains: " . $value . "<br />";
}
```

In this example, you name the variables to access the data as `$key` for the key and `$value` for the value. You then glue all this information together in the `echo` statement, with which you concatenate the key, some text saying *contains* the value, and then a `<br />` tag that acts like a carriage return to place each item on a new line.

Pull customers out of the database

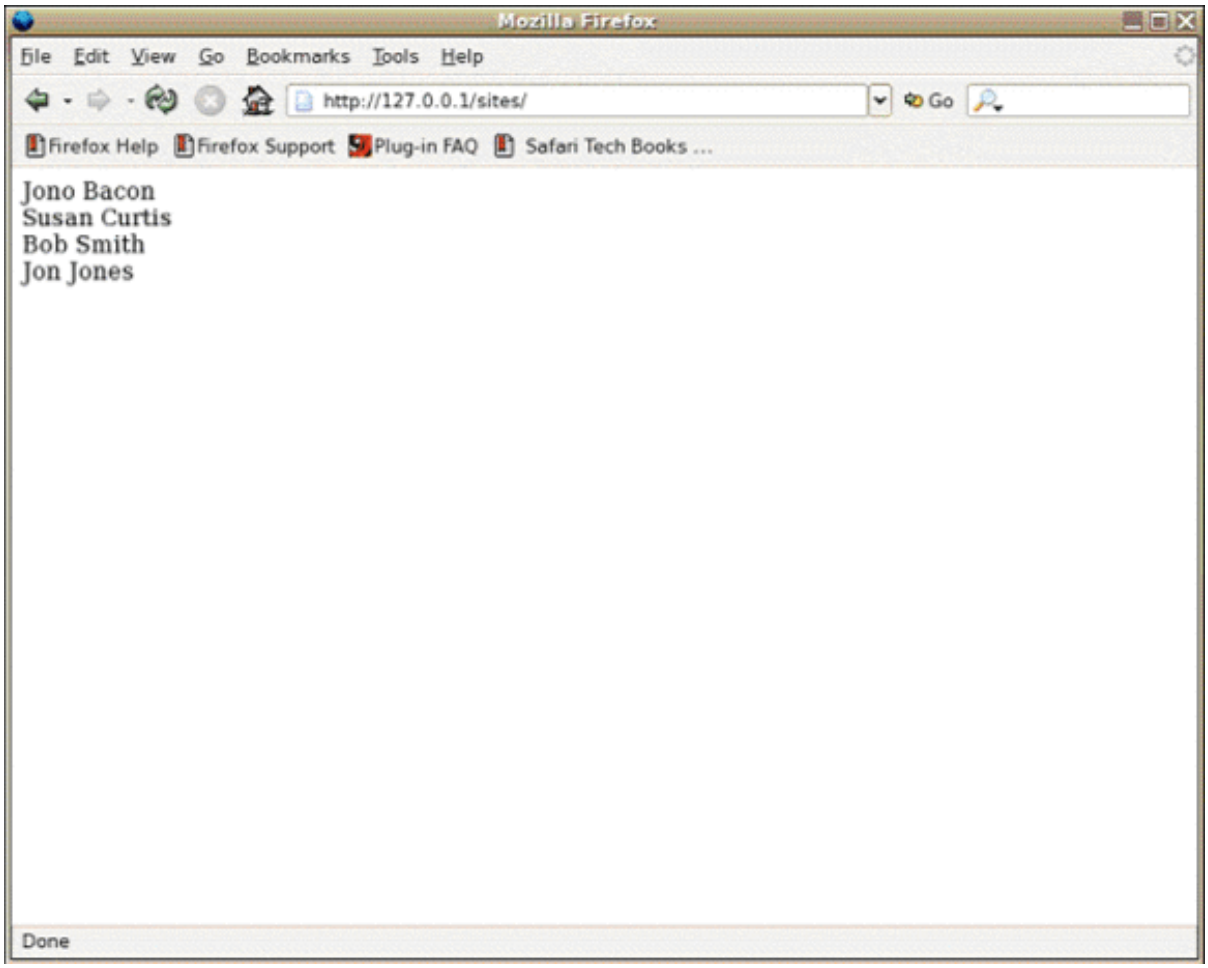
So, how does all this array theory relate to getting the customer information out of the database? You use an array to store the details from each row, and store the data in a series of key-value pairs. This way, if you want to get the contents of the **forename** field, you look up the `forename` key in the array.

To iterate through each row in the record set, use the following example:

```
while($row = mysql_fetch_assoc($result)) {
  echo $row['forename'] . " " . $row['surname'];
}
```

You should see the output that Figure 5 shows for your records.

Figure 5. Display the contents of the customers table



The purpose of a `while` loop is to simply repeat the code between the `{` and `}` brackets over and over again while the value in the parentheses next to the `while` statement is true. The condition in the parentheses in this case is `$row = mysql_fetch_assoc($result)`. This line takes the result set in `$result` and puts it in the `mysql_fetch_assoc` function to pull out the first row and place it into an associative array called `$row`. When you place this line in a `while()` loop condition, the loop will loop through each row in the record set and place each row in the `$row` associative array. This functionality provides a convenient system of accessing all of the data in the record set with the same array name (`$row`).

The code inside the `while()` loop simply displays the contents of each row on the page by referencing the **field** name in the `$row` array. As such, `$row['forename']` displays the contents of the **forename** field from each row in the database. In this particular example, you can see how you concatenate the **forename** field, then a single white space, and then the surname, which should result in the names being displayed from your table.

Concatenation

One of the most fundamental concepts when you code in PHP -- and the one concept that trips most new PHP users up -- is that of concatenation. This technique refers to gluing together the contents of a variable and other information by using the `.` symbol. In the following example, you can see how the `.` glues together some plain text and the `$name` variable:

```
$name = "Bob Smith";
echo "Your name is " . $name;
```

In the previous example, it is recommended that you concatenate the string and the variable as suggested, but you don't actually have to do this. You could write it as the following line:

```
echo "Your name is $name";
```

This line may be simpler to understand, but concatenation creates cleaner-looking code. In addition, most text editors cannot apply proper syntax highlighting to non-concatenated code, and you are also required to use concatenation for arrays.

Fine-tune your SQL

In the first example of connecting to MySQL, the SQL was fairly simple. SQL is incredibly flexible, and it is likely that you will want to search for specific records and possibly order the data you get back in different ways. You can do all this within SQL, so your PHP scripts can remain largely unchanged.

A great place to play with much of this is within phpMyAdmin. As mentioned earlier, whenever you interact with phpMyAdmin, the SQL that is generated for each interaction is displayed. Within phpMyAdmin, you can also run arbitrary SQL statements to explore the language and try different queries. To do so, click the small SQL window icon in the left bar of the phpMyAdmin interface. A new window appears (if you are using Firefox, make sure that the popup blocker does not block the window) in which you can enter your SQL queries. When you have added a query, click **Go**. The main interface window updates with results of your query.

Perform different query types

Even with your simple customers table, you can perform several different types of query to bring back information in different ways. As an example, if you only want

the rows that match a specific customer ID, try this query:

```
SELECT * FROM customers WHERE id = 1;
```

This query brings back the record with the user id `1`. (This is why it is so important to have a unique Primary Key for each table.) You can also try this query:

```
SELECT * FROM customers WHERE surname = "Curtis";
```

This query brings back all users with *Curtis* in the **surname** field. If you have a large customers database, it is likely that you might want to order how this data comes back. To do so, you can use the **ORDER BY** field:

```
SELECT * FROM customers WHERE surname = "Smith" ORDER BY forename ASC;
```

In this query, you ask for all records that have the surname *Smith,* but you order the results back in alphabetical order for the **forename** field. The `ASC` part refers to ascending order, and you could have the order reversed by using `DESC`, instead.

Section 5. Create the Web application

Display customer information: Create the link

A frequent mechanism used on the Web is a *drill-down.* This technique displays a list of summarized links and then gives the user a chance to click a link and go to another page to view more information about whatever they clicked on. In this project, you will create a drill-down of customer names: When you click a customer name, you will view more details about the user.

To achieve this, you first need to add some extra fields to the customers table to display on the linked page. Add fields such as *description, address, telephone,* and whatever other kinds of information you need.

Display customer information: Pull information from the database

You must now ensure that you pull all the data from the database. Set your SQL line to the following:

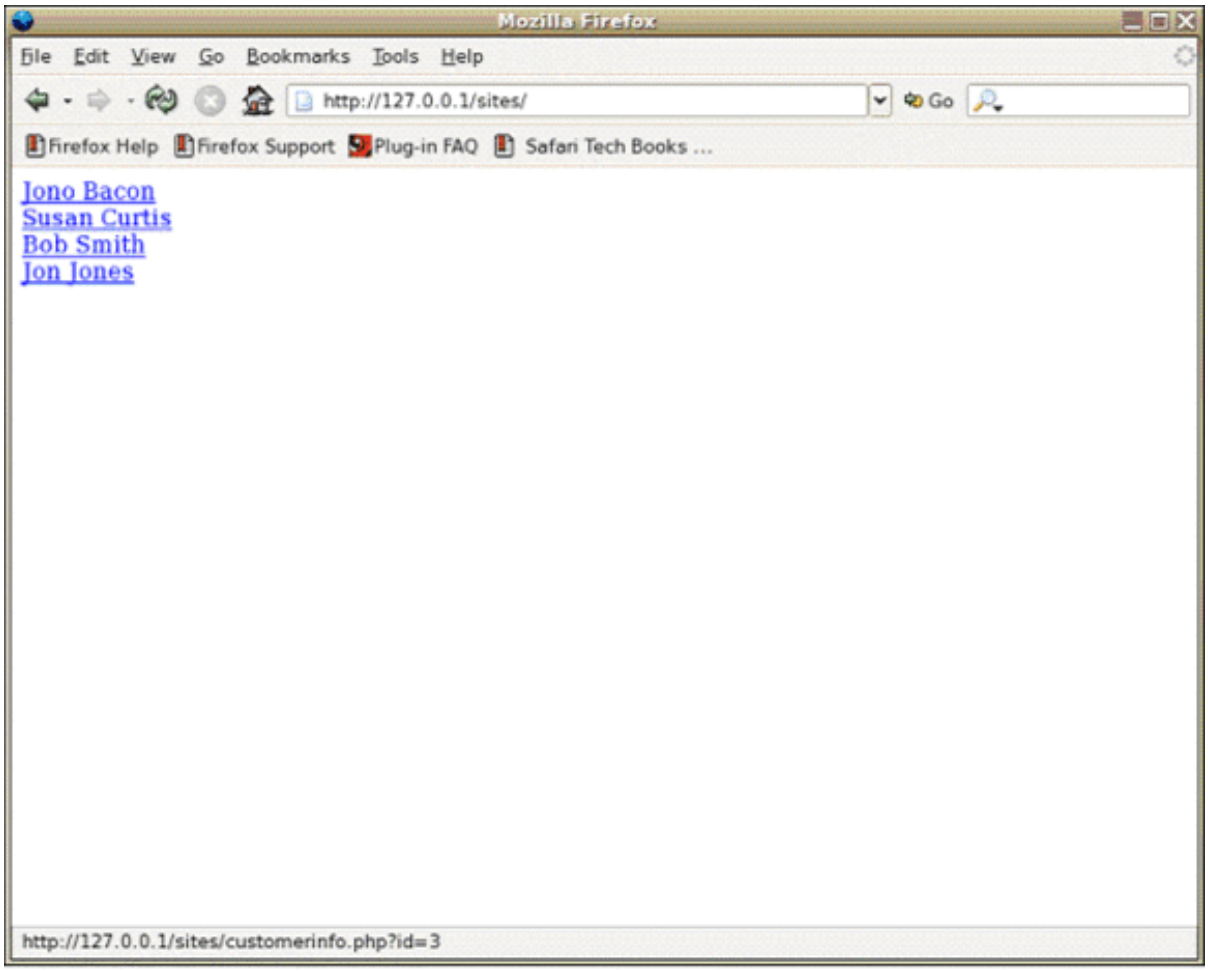
```
$sql =
"SELECT *
FROM
customers;";
```

Next, adjust the contents of your `while` loop and change it into a link that concatenates the ID of the record into the link. This should result in a link such as http://localhost/customerinfo.php?id=1:

```
echo "<a href='customerinfo.php?cust_id="
      . $row['id'] ."'>"
      . $row['forename'] . " "
      . $row['surname']
      . "</a>";
```

This line dynamically creates a link and appends the right ID for each record. If you hover your mouse over the links, you should see the dynamic ID in the status bar (see Figure 6).

Figure 6. Create links for the records complete with a GET variable



Display customer information: Create the information page

You now need to create the information page. Create a new page called productinfo.php, and first include the database connection information:

```
$db_host = "localhost";
$db_username = "bob";
$db_password = "sum65for!";
$db_database = "customers";

$db = mysql_connect($db_host, $db_username, $db_password);
mysql_select_db($db_database, $db);
```

Next, read in the value from the address bar and put it into a separate variable:

```
$theid = $_GET['cust_id'];
```

In this line, you access the variable at the end of the URL by using `$_GET`. This feature in PHP is called a *Superglobal,* and you can use it to get `GET` variables (that is, variables that appear at the end of the URL). You would then ideally want to run the variable through some validation checks to ensure that people are not trying to exploit the variable, but I'm going to keep the script simple for the purposes of this tutorial.

Display customer information: Pull specific records

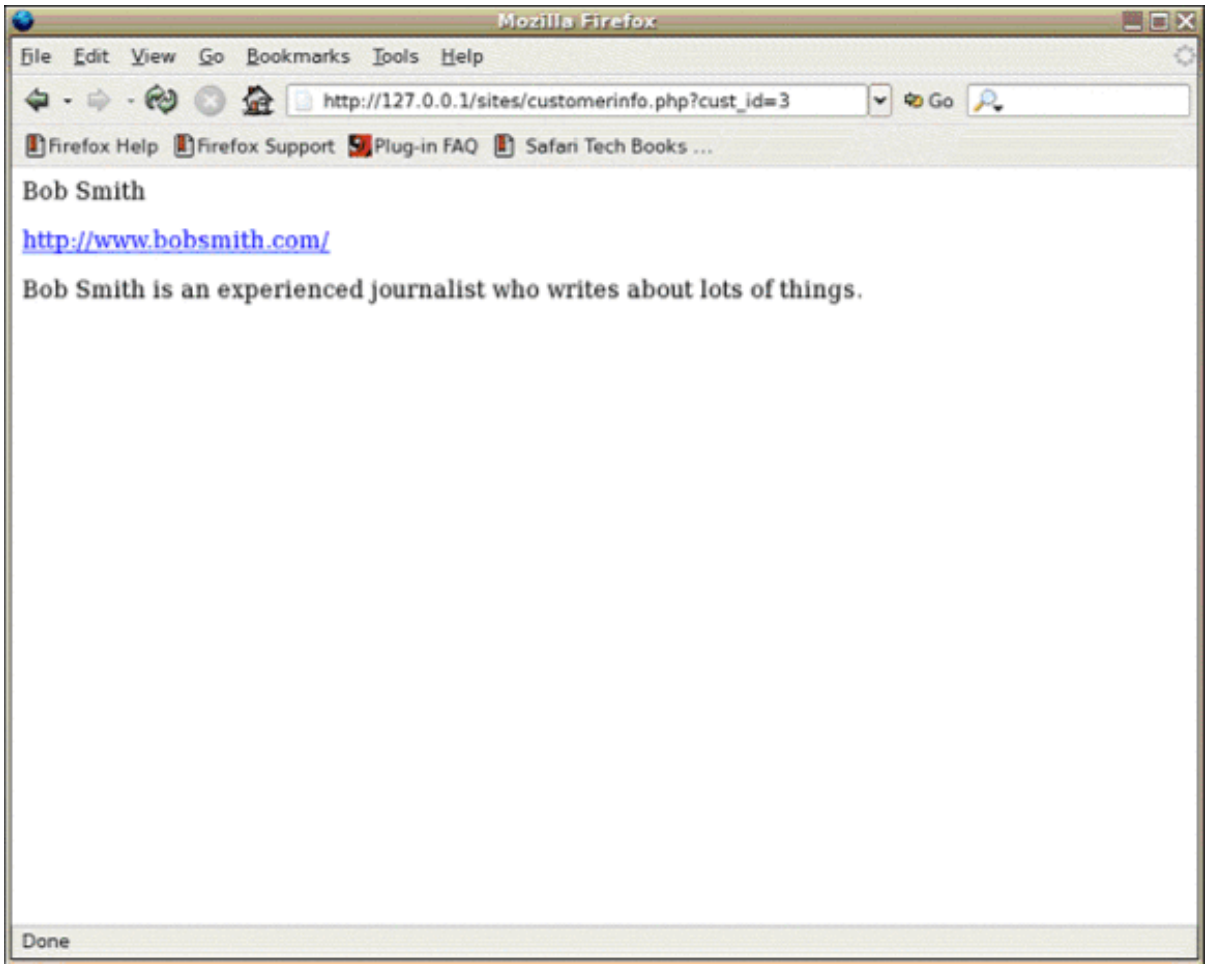
Now, pull out the specific record from the table that has the ID now stored in the `$theid` variable. Do this with the following code:

```
$sql = "SELECT * FROM customers WHERE id = " . $theid . ";";
$result = mysql_query($sql);
```

This code will perform the query, and you can now present the information by simply echoing out the results (see Figure 7):

```
echo $row['forename'] . "<br />";
echo $row['surname'] . "<br />";
echo $row['description'] . "<br />";
```

Figure 7. Display the complete record information when the user clicks a record



Section 6. Summary

Summary

This tutorial provides a simple application that will get you started with the core principles in PHP and MySQL programming. LAMP is a huge subject, and I have only scraped the surface that is the LAMP system. The LAMP system is wonderfully flexible. Although I only covered a subset of the aspects involved in LAMP development, the concepts discussed here can give you a solid foundation that will prepare you to meander through the huge selection of LAMP tutorials and documentation scattered across the Internet.

Resources

- Participate in the discussion forum for this content.
- The XAMPP platform XAMPP platform provides an easily installable LAMP setup, complete with additional tools and libraries.
- The PHP Web site is an essential resource when programming, and you might keep the PHP manual open in your browser while you are developing.
- The MySQL manual and Web site also offers a critically important resource when you work with your database and SQL.
- To learn more about Cascading Stylesheets (CSS), take a look at the CSS Zen Garden -- a great resource for learning CSS.
- A useful resource for a range of Web technologies, particularly for beginners, is the W3 Schools Web site.
- For information and resources on the core Web development technologies, take a look at the official W3C site; this is where the technologies are invented and managed.
- Find hundreds more Web architecture resources on the developerWorks Web architecture zone.
- Read Ian Gilfillan's *Database Journal* tutorial Referential Integrity in MySQL for more information about how to store and access data in MySQL databases.

About the author

Jono Bacon

Jono Bacon is a professional writer and Open Source consultant. Working at the government-funded OpenAdvantage center in the United Kingdom, he spends most of his time consulting with people on how Open Source can be useful for them. Jono is also an established writer who writes for a range of IT magazines and Web sites. He is the co-author of *Linux Desktop Hacks.*