Chapter 1: Introduction

Introduction

When you start to create web pages, you usually reach a point where you want or need to do more, and more usually refers to providing server-side functionality, such as storing or retrieving information. A simple application is to allow users to add comments to a web page. This requires that text provided by the user is stored in a database, and the text associated with a page is displayed along with the rest of the page content.

This is only a very high-level outline of what we want to do, and we'll see more detailed descriptions later.

This exercise assumes that you're familiar with DHTML, i.e. HTML, CSS and JavaScript, although the level is not high. It introduces the use of a database (MySQL) and a web server (Apache) and the server-side programming language PHP.

This document is divided into a number of parts, each of which is a logical step in getting to the point where your server page works. If you follow the steps slowly and carefully, asking yourself *why?* at each stage, you'll learn a lot. This document is being created for *CoderDojo*, so if you're participating in that movement, please ask questions of your mentors if you're unclear about anything. In any case there is a wealth of information on the web describing how to do just about anything in PHP and MySQL and Apache, so you should be able to find answers online.

Chapter 2: Installation

Required software

These instructions are for Windows 7, but if you are using Linux or Mac, the steps are similar. We want to use a web server and a database, and we don't want to have to pay for them. The obvious choices are Apache and MySQL, and we'll use the versions included in XAMPP since that's packaged up with everything you need. These two are servers, i.e. they serve up information and/or functions.

Apache is a web server, i.e. it serves up web pages across the internet. It can use the HTML pages you have created up to now, but it also has the ability to process dynamic web pages written in other languages like PHP. It identifies these from the file extension, so using the correct file extension is very important. Apache assumes that files that end with *.php* are PHP files and those that end with *.html* are HTML files. PHP is a programming language which we'll introduce a little later in this document.

If you want this to work with a smaller computer such as Raspberry Pi, you can use a smaller server like **Lighttpd** instead. This functions pretty much the same as Apache, but it isn't included in the XAMPP package.

MySQL is a relational database system (RDBMS). This means that it is a system that can be used to store and retrieve information in a structured way. And it's relational because it contains links between data items, i.e. relationships. For example, the database might contain information about sales, and each sales record might have a relationship to a product and its description, and also to a customer and her information. We'll cover more about the structure of data in a database later.

To install everything you need, do the following:

- If it isn't already installed, download and install the Microsoft Visual C++ 2008 redistributable from this URL:

  `http://www.microsoft.com/en-us/download/confirmation.aspx?id=5582`

- Download XAMPP from http://www.apachefriends.org - the installer is probably the easiest to work with so it's best to download this. Launch the downloaded file, at

  ```
  xampp-win32-1.8.1-VC9-installer.exe
  ```

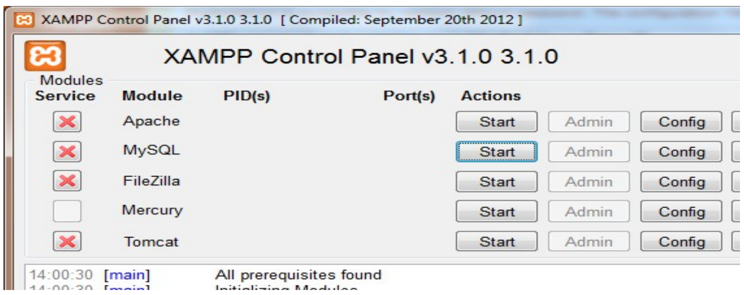
(or the version you have downloaded). Accept the defaults and complete the installation.

Starting the servers

Now we'll set up the servers. Launch the XAMPP control panel – run:

```
\xampp\xampp-control.exe
```

to start the control panel. We want both the MySQL and Apache servers, so click on the “start” buttons beside each of these:



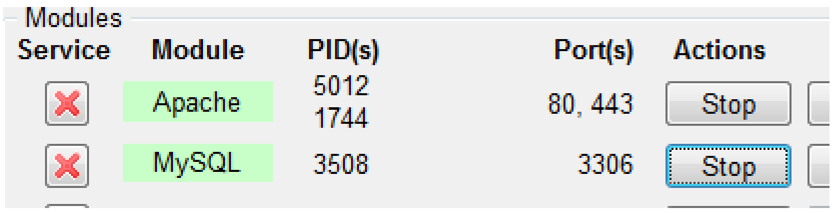
XAMPP Control Panel v3.1.0 3.1.0 [Compiled: September 20th 2012]

XAMPP Control Panel v3.1.0 3.1.0

Modules Service	Module	PID(s)	Port(s)	Actions		
☒	Apache			Start	Admin	Config
☒	MySQL			Start	Admin	Config
☒	FileZilla			Start	Admin	Config
	Mercury			Start	Admin	Config
☒	Tomcat			Start	Admin	Config

```
14:00:30  [main]        All prerequisites found
```

This will result in the servers being started and highlighted in green:



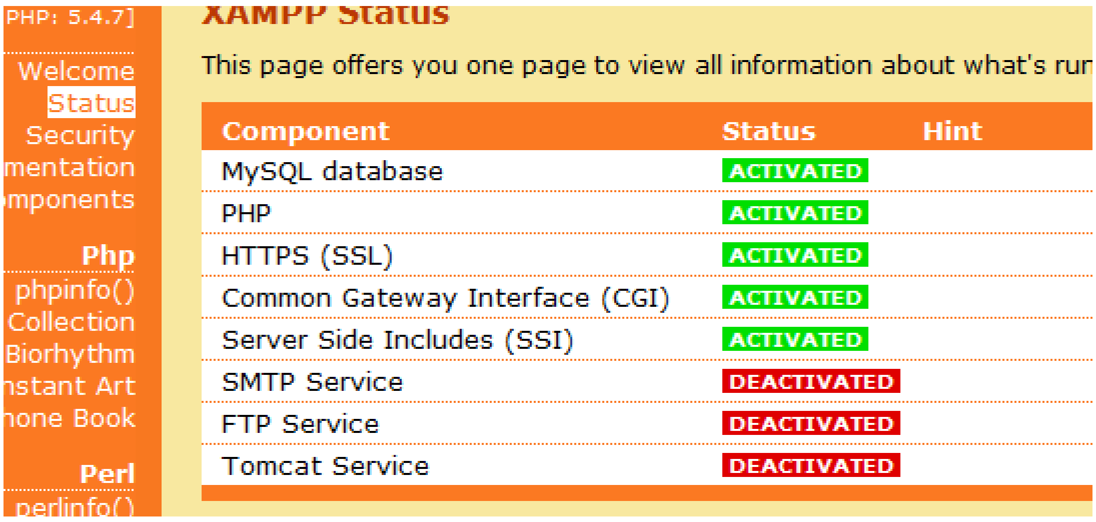
Modules Service	Module	PID(s)	Port(s)	Actions
☒	Apache	5012 1744	80, 443	Stop
☒	MySQL	3508	3306	Stop

Server configuration

Now that your servers are up and running, we need to make sure they run correctly and securely. We can access the server using the local address – launch your browser and type the following into the address bar:

```
http://localhost
```

This will display the welcome screen: it's actually redirected to http://localhost/xampp which is the root of the XAMPP application. Click on the *Status* link on the left, and it should display this:

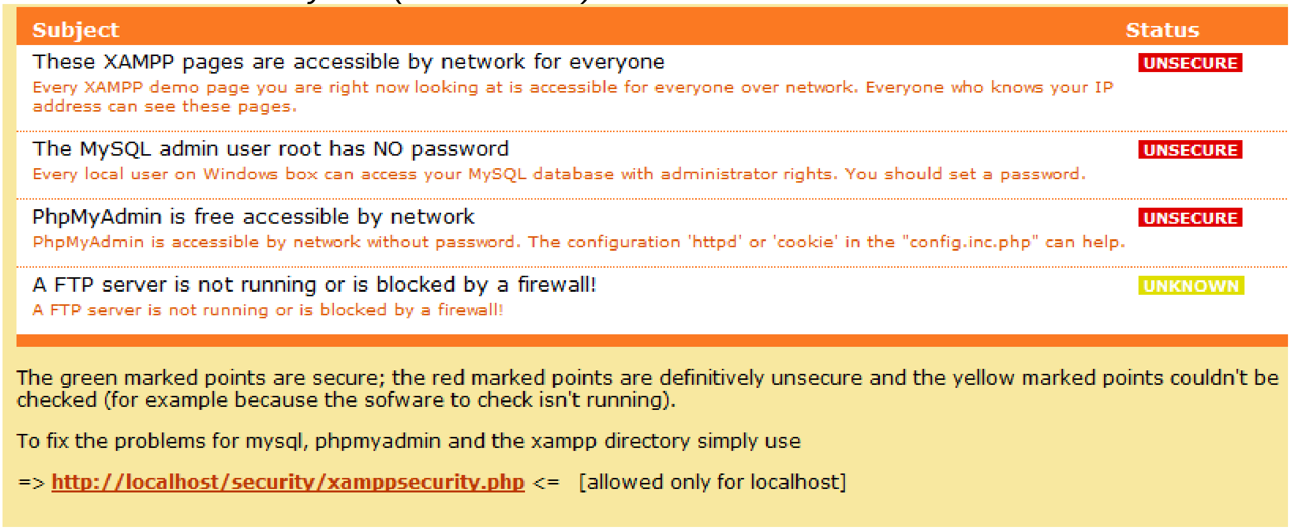


XAMPP Status

This page offers you one page to view all information about what's run

Component	Status	Hint
MySQL database	ACTIVATED	
PHP	ACTIVATED	
HTTPS (SSL)	ACTIVATED	
Common Gateway Interface (CGI)	ACTIVATED	
Server Side Includes (SSI)	ACTIVATED	
SMTP Service	DEACTIVATED	
FTP Service	DEACTIVATED	
Tomcat Service	DEACTIVATED	

This shows that the database has been started as well as some other features of the server, just what we wanted. Next we want to make sure that the system is secure, so click on the *Security* link (under *Status)* which will show:



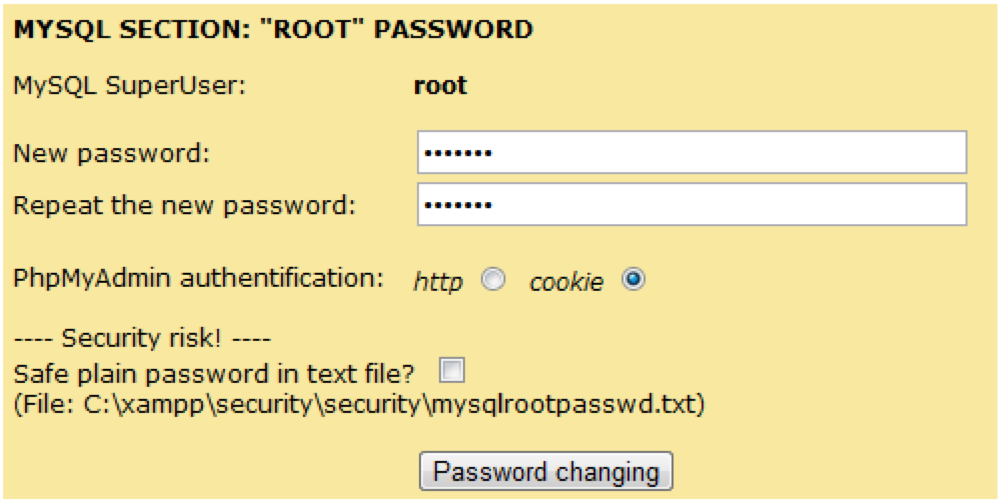
Subject	Status
These XAMPP pages are accessible by network for everyone Every XAMPP demo page you are right now looking at is accessible for everyone over network. Everyone who knows your IP address can see these pages.	UNSECURE
The MySQL admin user root has NO password Every local user on Windows box can access your MySQL database with administrator rights. You should set a password.	UNSECURE
PhpMyAdmin is free accessible by network PhpMyAdmin is accessible by network without password. The configuration 'httpd' or 'cookie' in the "config.inc.php" can help.	UNSECURE
A FTP server is not running or is blocked by a firewall! A FTP server is not running or is blocked by a firewall!	UNKNOWN

The green marked points are secure; the red marked points are definitively unsecure and the yellow marked points couldn't be checked (for example because the sofware to check isn't running).

To fix the problems for mysql, phpmyadmin and the xampp directory simply use

=> http://localhost/security/xamppsecurity.php <= [allowed only for localhost]

This is clearly not what we want, where there are three major security issues. The link below the table will take us to where we can correct these problems. First set the MySQL password and make sure that the authentication is set to *cookie*. It is not safe to save the password, so do not check the box, but it is important to note the password or you will not be able to access your database if you forget it. Click the button to make the change:

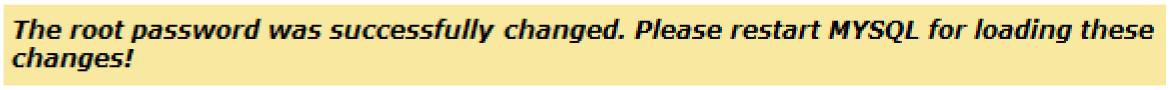


MYSQL SECTION: "ROOT" PASSWORD

MySQL SuperUser:	**root**
New password:	•••••••
Repeat the new password:	•••••••
PhpMyAdmin authentification:	*http* ○ [✓ *cookie*]

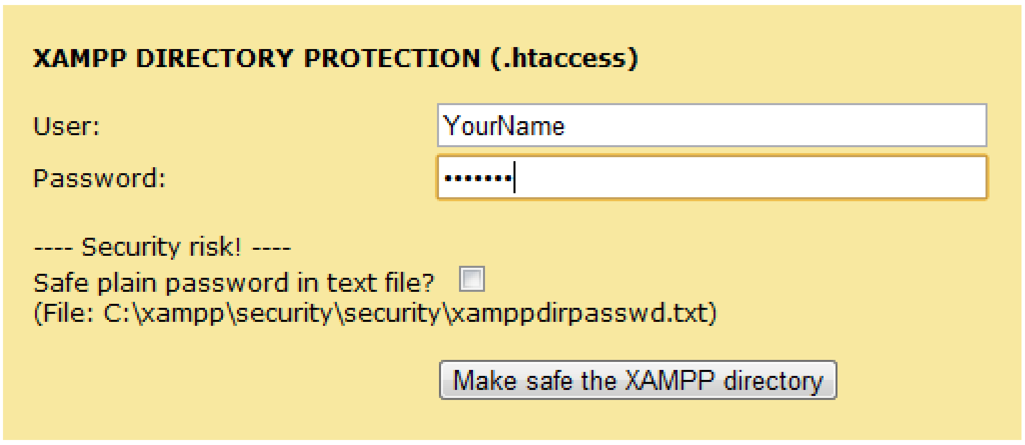
---- Security risk! ----
Safe plain password in text file? ☐
(File: C:\xampp\security\security\mysqlrootpasswd.txt)

The page is displayed again with the message:



The root password was successfully changed. Please restart MYSQL for loading these changes!

Now, change the access by specifying the username and password. Again, don't check the box to save the information, but it's important to remember the credentials (your username and password):

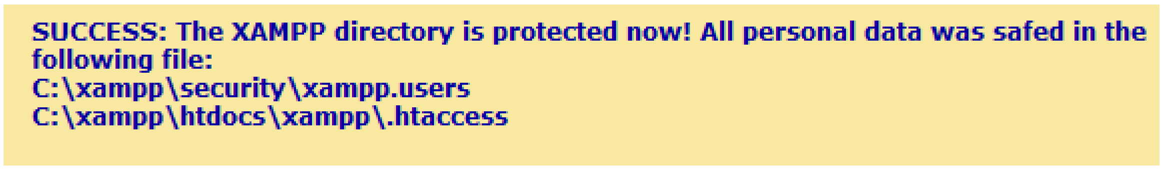

XAMPP DIRECTORY PROTECTION (.htaccess)

User: YourName

Password: •••••••

---- Security risk! ----
Safe plain password in text file? ☐
(File: C:\xampp\security\security\xamppdirpasswd.txt)

When you click the button the following message is displayed:

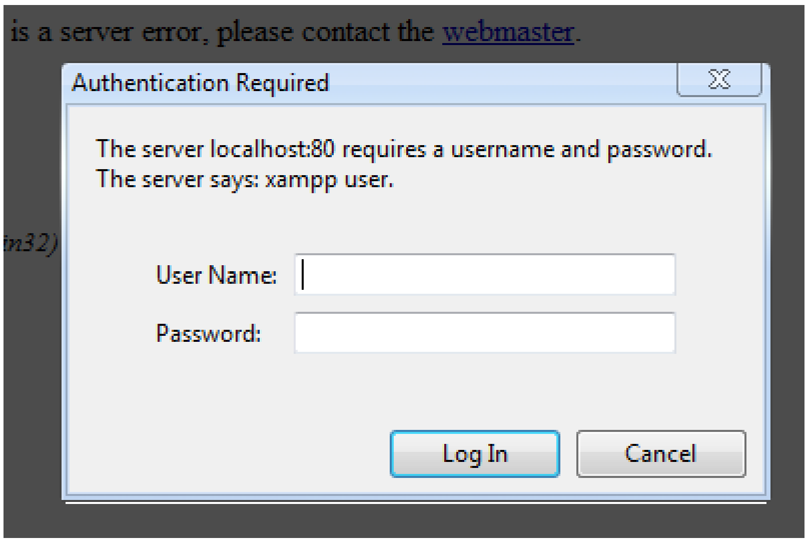

**SUCCESS: The XAMPP directory is protected now! All personal data was safed in the following file:
C:\xampp\security\xampp.users
C:\xampp\htdocs\xampp\.htaccess**

Now, restart your servers to make sure that everything is OK. Click on the S*top* buttons for both servers and then click on *Start* again to restart them.

Point your browser at the welcome page again:

http://localhost

and it will challenge you with the username and password you entered above:


is a server error, please contact the webmaster.

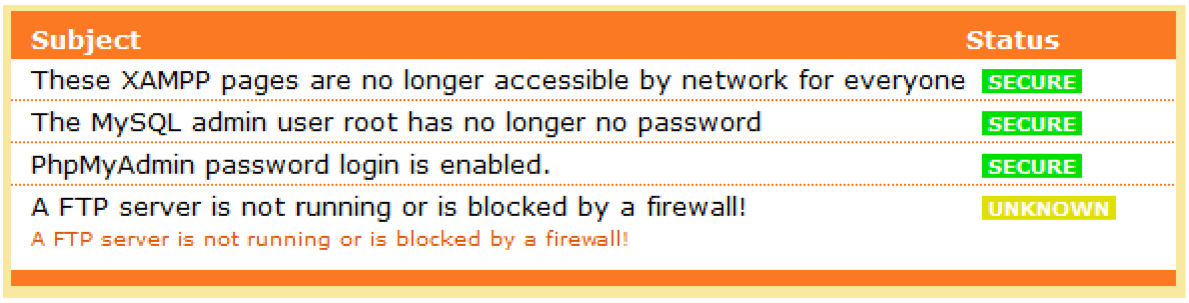
Authentication Required

The server localhost:80 requires a username and password.
The server says: xampp user.

User Name:

Password:

Provide your credentials and you are taken to the Xampp welcome page. Click on the *Security* link on the left, and you should see:



Subject	Status
These XAMPP pages are no longer accessible by network for everyone	SECURE
The MySQL admin user root has no longer no password	SECURE
PhpMyAdmin password login is enabled.	SECURE
A FTP server is not running or is blocked by a firewall! A FTP server is not running or is blocked by a firewall!	UNKNOWN

As you can see, it's all secure now. At this point, the system is all ready to use and we can start to create the application.

Chapter 3: The database

Defining the database

The comments data need to be kept somewhere, and the natural place to store them is a database. When we consider the various pieces of information we need to store, we get a list like:

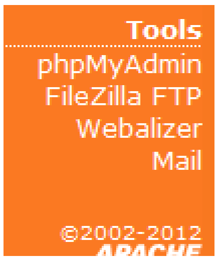
- The text of the comment
- The date and time the comment was entered
- The web page with which the comment is associated
- The name of the comment's author
- The internet address of the author
- An id to ensure the comment is unique

Each of these items is called a *field*, and this collection of fields is called a *record*. Now, each comment will be in a record, and all comments are gathered together in a *table*. Another way of stating this is:

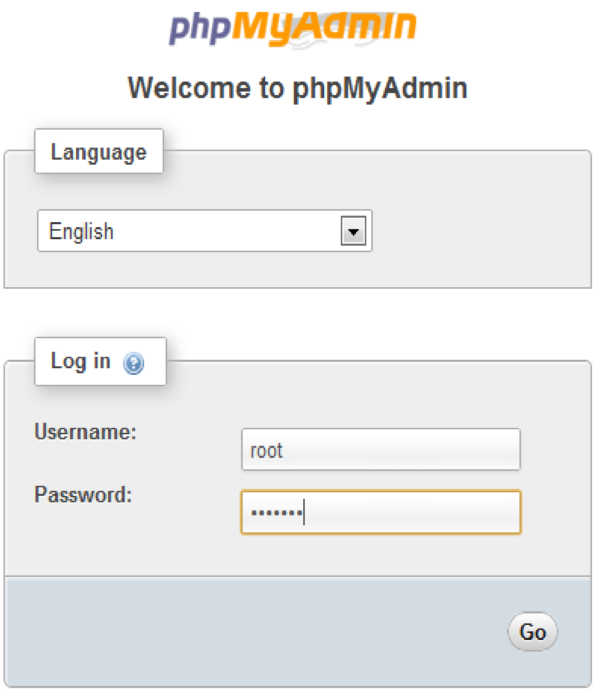
- A *database* contains a number of *tables*
- A *table* contains a number of *records.*
- A *record* is a collection of *fields*

So, in the database, we will store these items together in a table of comments.

To create the database and table, click on the phpMyAdmin link:



This will launch the database administration tool. You have to log into the admin system using the username *root* and the password you provided above, and press *Go*:



Language: English

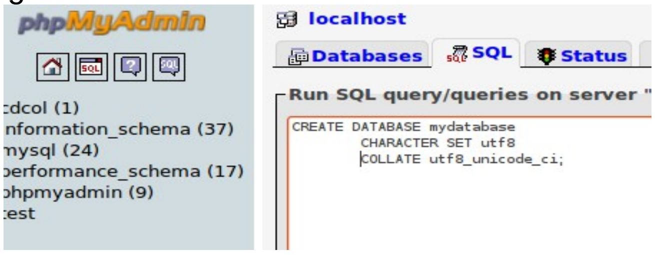
Username: root

Password: •••••••

The admin system allows you to work with MySQL in a simple fashion, but we'll use a special database programming language called Structured Query Language (SQL). The advantage of doing this is that you can see exactly what we're doing. If you're looking for complete information, you can download a PDF document containing the MySQL reference manual from:

http://dev.mysql.com/doc/

Our first step is to create a database where we'll store the data. Open the SQL tab in the phpMyAdmin page:



```
CREATE DATABASE mydatabase
        CHARACTER SET utf8
        COLLATE utf8_unicode_ci;
```

Now enter the following SQL to create the database:

```
CREATE DATABASE mydatabase
       CHARACTER SET utf8
       COLLATE utf8_unicode_ci;
```

Click *Go* and the database is created.

This SQL statement says that we want to create a database called mydatabase and it describes that it should use a particular character set encoding, and a particular collation.

If you're not familiar with the concepts of internationalization, using the UTF-8 character set is a good idea since this is capable of encoding all languages unlike e.g. ASCII which is limited to the English character set.

Similarly, using the Unicode collation algorithm (or sorting algorithm) means that sorting items from the database will probably be in the right order for most countries.

Once the database has been created, we can create the comments table. The SQL to create this table is below. There are two parts – first we have to tell the system that we want to create the table, but we have to also ensure that we're using our database:

```
USE mydatabase;

CREATE TABLE comments (
   id          INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,
   comment     TEXT NOT NULL,
   user        VARCHAR(255) NOT NULL,
   page        VARCHAR(255) NOT NULL,
   createdate TIMESTAMP DEFAULT CURRENT_TIMESTAMP)
CHARACTER SET utf8
COLLATE utf8_unicode_ci;
```

The USE statement simply says that we're referring to that particular database, and the CREATE TABLE statement creates the *comments* table in the *mydatabase* database. When we create a table, we're describing the record contents, so we have to specify the fields that make up that record. As you can see, we list the fields by giving each a name, and then specifying the data type. There are many different data types, and we could potentially use different types for the same field. However, here we use the following types:

- `INTEGER` – numeric data, can be negative, but no decimals
- `TEXT` – pretty much what it says, i.e. it contains a block of text
- `VARCHAR` – can contain anything, and in this case it could easily be TEXT
- `TIMESTAMP` – date and time

Each field also has some attributes - let's look at some of these:

- NOT NULL: when creating a record in the table, that field must have a value.
- AUTO_INCREMENT: when creating a new record, the value of this field is a sequential number that's incremented with each new record.
- PRIMARY KEY: is a unique value that identifies the record.
- VARCHAR: this describes the field as having a generic content
- TIMESTAMP says that we want to use this to store when the comment was created, and its value should be the current time when the comment is stored.

We could add other items to the table, such as a flag for marking the comment as approved by a moderator, a field to contain the IP address of the author of the comment, etc. but we'll just stick to the above for this exercise.

Chapter 4: The web application

The browser client

At this point, we have our database, so the next step is to create the web application.

The first part is the part that displays in the browser, and this could be a blog or something completely different. What is important is that it allows visitors to add comments, and the rest of the page is pretty much irrelevant for the purpose of this exercise.

Creating the web application involves adding it to the documents that Apache knows about. We'll create the application area by adding a new folder under Apache's documents where we'll store everything associated with the appplication. All of the web pages that Apache serves up are in the *htdocs* directory.

```
mkdir \xampp\htdocs\comments
```

Next we'll create a web page that has two parts that are of interest:

- the area where existing comments are displayed, and
- the area where users can add comments

```
<!DOCTYPE HTML>
<html>
<head>
   <title>First page of comments application</title>
   <meta charset="utf-8">
   <link rel="stylesheet" href="comments.css">
</head>
<body class="defaults">
   <h1 class="header">My First Page</h1>
   <div id="pagecontent" class="content">
      Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed
      diam nonummy nibh euismod tincidunt ut laoreet dolore magna
      aliquam erat volutpat. Ut wisi enim ad minim veniam, quis
      nostrud exerci tation ullamcorper suscipit lobortis nisl ut
   </div>
   <div id="comments" class="comments">
        [COMMENTS DISPLAYED HERE]
   </div>
   <div id="newcomments" class="newcomments">
        [COMMENTS INPUT HERE]
   </div>
</body>
</html>
```

As you can see, this contains the basic layout of the page, with references to an extra file containing styles. The sections we're interested in are the two *<div>* elements with the ids *comments* and *newcomments*.

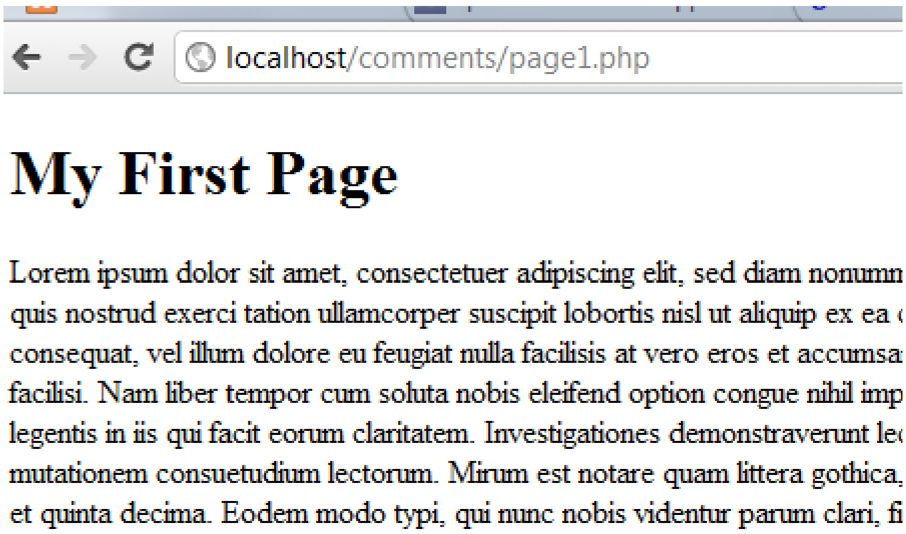
Copy the content above and paste it into a file called

```
C:\xampp\htdocs\comments\page1.php
```

You can then load it in your browser using the address

    http://localhost/comments/page1.php

And it should display the rather boring page as



localhost/comments/page1.php

My First Page

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonumm
quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea
consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsa
facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imp
legentis in iis qui facit eorum claritatem. Investigationes demonstraverunt le
mutationem consuetudium lectorum. Mirum est notare quam littera gothica,
et quinta decima. Eodem modo typi, qui nunc nobis videntur parum clari, fi

We can add CSS styles to make the page look nice, so create the stylesheet with the following content, which decorates the style classes in various parts of the page:

```
.header {
        width: 100%;
        background-color: lightgrey;
}
.content {
        margin-left: 50px;
        margin-right: 50px;
        margin-bottom: 20px;
}
.comments {
        margin-left: 50px;
        margin-right: 50px;
        margin-bottom: 20px;
        border-top: 3px groove lightgrey;
        border-bottom: 3px groove lightgrey;
}
.newcomments {
        margin-left: 50px;
        margin-right: 50px;
        margin-bottom: 20px;
}
```

and save it to

```
C:\xampp\htdocs\comments\comments.css
```

Reload the page and you'll see that it looks a lot better.

Adding server functions

This web page so far can display the content directly in the browser, but we need to add the piece that runs on the server. As mentioned earlier, we are going to write the server-side code in PHP, but it can actually be written in many other programming languages. Java is very popular, as are Perl and C, and Python is becoming more common. For Microsoft servers, C# is also a popular choice.

As mentioned above, Apache knows to use PHP by the file extension: instead of naming our HTML file *page1.html*, we'll call it *page1.php* and the server will automatically treat it as PHP. So far, the file contains no PHP, but it will still work just fine.

Adding PHP code to the HTML is as simple as adding the PHP start and end markers with the PHP code in between:

```
    <?php PHP code goes here ?>
```

This can be placed anywhere in the file, and there can be many of these on a single page, manipulating the page's content to perform a particular function. What these parts do is allow you to include PHP code that is run on the server, before it is sent to the browser. In fact, you never see the PHP code in the browser, with only the DHTML being passed through to the client.

Now, to send data back to the server, we need to use a Form, which is written like this:

```
    <form action="page2.php" method="post">
      <div class="entry">
         E-mail address: <input type="text" id="email" class="email" />
      </div>
      <div class="entry">
         Comment: <textarea id="commenttext" class="commenttext"></textarea>
      </div>
      <div class="entry">
         <input type="submit" value="Add" />
      </div>
      <input type="hidden" id="ipaddr" value="9.9.9.9" />
    </form>
```

All this looks fine, but we need a few missing items. First, the form's action is a fixed value and it's considered good practise to allow PHP to identify its own address as the action:

```
    <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post">
```

This is your first piece of PHP so it's important to explain it thoroughly:

```
    <?php
        echo $_SERVER['PHP_SELF'];
    ?>
```

As mentioned above, the *<?pvp* and *?>* mark the start and end of the PHP. The rest of the code is an echo statement, i.e. it says that it should display whatever follows, writing it into the HTML as it's being sent to the browser. The item being displayed is a server variable: the *$_SERVER* variable is actually an array of various server items that can be used here. This is an associative array, i.e. it can be addressed by providing a value that refers to the value associated with that value: *PHP_SELF* is a special name, so this says to return the name of the current web page, i.e. *page1.php*.

Also, the user's internet address is a fixed value which is downright silly. We need a way to extract that from the interaction with the server, and the *$_SERVER* set of variables provides a mechanism to achieve what we need – we reference *REMOTE_ADDR,* the address of the remote client, here :

```
    <input type="hidden" id="ipaddr" value="<?php echo $_SERVER['REMOTE_ADDR']; ?>" />
```

The PHP code is very simple, comprising *echo* (which simply displays something) followed by a variable. The variable in both cases is the PHP system variable $_SERVER, and the code extracts the page address and the IP address from these. This variable is described well in the PHP reference which you can find at

    http://php.net/manual/en/reserved.variables.server.php

I mentioned the possible addition of a field to store this value earlier, and this is a good way to achieve this.

We also have some new CSS styles which you can add to the *comments.css* file:

```
.entry {
        vertical-align: top;
        text-align: center;
}
.email {
        width: 80%;
        margin-left: 10px;
        margin-right: 50px;
        margin-bottom: 10px;
}
.commenttext {
        width: 83%;
        margin-left: 10px;
        margin-right: 50px;
        margin-bottom: 10px;
}
```

Now reload the page, and if you look at the page source in your browser, you will see the form content as:

```
<form action="/comments/page1.php" method="post">
   <div class="entry">
      E-mail address: <input type="text" id="email" class="email" />
   </div>
   <div class="entry">
      Comment: <textarea id="commenttext" class="commenttext"></textarea>
   </div>
   <div class="entry">
      <input type="submit" value="Add" />
   </div>
   <input type="hidden" id="ipaddr" value="127.0.0.1" />
</form>
```

As you can see, the *action* and the *value* parameters have been replaced by what the PHP code generated. What has actually happened here is that the PHP code is executed on the server, and the result is delivered to the browser, along with the rest of the HTML page.

This illustrates how PHP works: it runs on the server and not at the browser. This means that it has access to all the functionality available to the server, which is out of reach of the browser. That way, someone can connect to the server from far away yet retrieve the exact same results as someone directly connected.

Accessing the database

Now that we can send a comment to the server, there are two tasks we have to complete:

- display the comments for the page that are already in the server, and
- store a new comment sent by the page.

Of course, before we read anything from the database or write to it, we have to tell the server about the database we want. We use the username and password to connect to the database. Currently we only have the user *root* and her password, but this is probably not the safest since root can do anything with the database. However, as an extra challenge, you can read up on how to add a new user to the database who has limited rights to only read and write the *comments* database.

If you look at other PHP pages, you may see that it uses functions with names that start with *mysql_*. This is the old way to do things, so we won't use that mechanism. Instead we will use *PHP Data Objects*, referred to as *PDO* which will make doing things with a database much easier.

When the page starts, we need to connect to the database by creating a PDO i.e. at the top of the page, we add:

```
<?php
   // Link to the database server
   $db = new PDO('mysql:host=localhost;dbname=mydatabase;charset=UTF-8',
                 'root', 'Me32316xy');

   // If an error occurs, we want to catch it as an "exception"
   $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
?>
```

The parameters to this are all in a long confusing string, so let's see what it contains:

- *mysql* This indicates that the database is a MySQL database
- *host=* This says that the database is on localhost, i.e. the local machine. If the database is on another machine, you give its address here.
- *dbname=* This gives the database name, mydatabase
- *charset=* This says to use Unicode as UTF-8.

We also have to give the username and password.

Then we tell PHP that any time we encounter an error, it should cause an *exception.* This allows us to manage what happens when a problem occurs, rather than displaying a complicated message on the screen.

Displaying the comments

We want to display the comments that are already in the database, so we need to read these from the database and format them so that they look correct on the screen:

```
   // If this is in response to the "Add" button, insert the new record
   if ($_SERVER['REQUEST_METHOD'] === 'POST') {
      $author  = $_REQUEST['email'];
      $comment = $_REQUEST['commenttext'];
      try {
         $stmt = $db->prepare("INSERT INTO comments(user, comment, page)
VALUES(:field1,:field2,:field3)");
         $stmt->execute(array(':field1' => $author, ':field2' => $comment, ':field3' =>
$_SERVER['PHP_SELF']));
      } catch(PDOException $ex) {
         echo "An Error occured when adding a comment: " . $ex.getMessage();
```

```
        }
    }
```

First note that strings are concatenated, i.e. joined to each other, using the dot-operator. The first thing we do is format the query and substitute strings. You should always construct a query this way since it helps protect against SQL-injection attacks. The *mysql_real_escape_string* function makes sure than anything nasty that might be embedded in the string, is made harmless.

Let's look at the query used to retrieve the comments from the database. It's a SELECT which means that it identifies the records it wants:

```
SELECT * FROM 'comments'
```

means to select all the fields in the records from the *comments* table. Next is the WHERE clause, which tells which particular records we want:

```
WHERE page=$_SERVER['PHP_SELF'])
```

which means that we're interested in the records where the page is the same as the current page. Finally there's an ORDER BY clause which indicates the order in which the records are returned:

```
ORDER BY createdate
```

which says that we want the records to be in the order of the date in which the comments were entered.

Once the *mysql_query* function is run, the relevant records have been selected and we then read them one by one by means of the *mysql_fetch_object* function, and display them formatted for the page.

Of course if no records are found, the mysql_query call returns null, and we can display a text that indicates that the page has no comments.

Adding comments to the database

When the user inputs a comment and clicks the *Add* button, the form is submitted to the same page, as indicated by the *action* parameter, as a POST. This can be captured and processed on the server by the following code which builds on the code above that we used to open the database:

```
<?php
   $dbLink = mysql_connect('localhost', 'root', 'me32316');
   if (!$dbLink) {
       die('Could not connect to the database server: ' . mysql_error());
   }
   $db = mysql_select_db('mydatabase', $dbLink);
   if (!$db) {
       die('Could not connect to the database: ' . mysql_error());
   }
   if ($_SERVER['REQUEST_METHOD'] === 'POST') {
      $author  = $_REQUEST['email'];
      $comment = $_REQUEST['commenttext'];
      $insert = sprintf("INSERT INTO comments (user, comment, page) ".
                        "VALUES ('%s', '%s', '%s')",
                        mysql_real_escape_string($author),
                        mysql_real_escape_string($comment),
                        mysql_real_escape_string($_SERVER['PHP_SELF']));
      mysql_query($insert) or die('Cannot add comment: '.mysql_error());
   }
?>
```

What's new here is the test to see if this is a POST. If it is, we want to process it as a new comment added to the page. Note the use of 3 equals signs: this means that the values must be the same, and they must also be of the same type, in this case a string. Again, we use the $_SERVER array, this time to identify the request method, and since the form on the page says to use POST, that's what we check for.

Note that when PHP receives a request from the browser, it fills out three arrays:

- $_POST contains the parameters when the request is a POST
- $_GET contains the parameters when the request is a GET
- $_REQUEST contains the parameters, irrespective of whether it's a GET or POST.

We use the generic $_REQUEST array to retrieve the values of the two fields on the page that we want to store, i.e. the user's E-mail address and the content of the comment.

Creating a record in the database involves using the INSERT SQL statement:

```
INSERT INTO table (field1, field2, ...) VALUES ('value for field1', ...)
```

Note that once again we use the *mysql_real_escape_string* to make sure that there's nothing nasty in the strings from the browser when we build up the actual SQL statement. Also as before we use the *mysql_query* function to execute the SQL statement.

You'll see that the mysql_query is followed by *or die(message)*: this is a good way of stopping the page when you run into a fatal error. What it actually means is that if the first part returns null, this is interpreted as *false*, so it executes the next part of the *or*. If the first part is successful, it returns non-null value which is considered *true*, so the next part is not executed.

You may have noticed that we have created the record using only three fields. However, the ID field is automatically incremented to the next value, and the createdate field is automatically set to the current date and time, so there's no need to specify them explicitly.

Chapter 5: Putting it all together

The completed page

We have now created a very simple page with some simple server-side processing. The completed page looks like:

```
<?php
   // Link to the database server
   $dbLink = mysql_connect('localhost', 'root', 'me32316');
   if (!$dbLink) {
       die('Could not connect to the database server: ' . mysql_error());
   }

   // Let's use our database
   $db = mysql_select_db('mydatabase', $dbLink);
   if (!$db) {
       die('Could not connect to the database: ' . mysql_error());
   }

   // If this is in response to the "Add" button, insert the new record
   if ($_SERVER['REQUEST_METHOD'] === 'POST') {
      $author  = $_REQUEST['email'];
      $comment = $_REQUEST['commenttext'];
      $insert = sprintf("INSERT INTO comments (user, comment, page) ".
                        "VALUES ('%s', '%s', '%s')",
                        mysql_real_escape_string($author),
                        mysql_real_escape_string($comment),
                        mysql_real_escape_string($_SERVER['PHP_SELF']));
      mysql_query($insert) or die('Cannot add comment: '.mysql_error());
   }
?>

<!DOCTYPE HTML>
<html>
<head>
   <title>First page of comments application</title>
   <meta charset="utf-8">
   <link rel="stylesheet" href="comments.css">
</head>
<body class="defaults">
   <h1 class="header">My First Page</h1>
   <div id="pagecontent" class="content">
      Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed
      diam nonummy nibh euismod tincidunt ut laoreet dolore magna
      aliquam erat volutpat. Ut wisi enim ad minim veniam, quis
      nostrud exerci tation ullamcorper suscipit lobortis nisl ut
      aliquip ex ea commodo consequat.  Duis autem vel eum iriure
      dolor in hendrerit in vulputate velit esse molestie consequat,
      vel illum dolore eu feugiat nulla facilisis at vero eros et
      accumsan et iusto odio dignissim qui blandit praesent luptatum
      zzril delenit augue duis dolore te feugait nulla facilisi.
      Nam liber tempor cum soluta nobis eleifend option congue
      nihil imperdiet doming id quod mazim placerat facer possim
      assum. Typi non habent claritatem insitam; est usus legentis
      in iis qui facit eorum claritatem. Investigationes
      demonstraverunt lectores legere me lius quod ii legunt saepius.
      Claritas est etiam processus dynamicus, qui sequitur
      mutationem consuetudium lectorum. Mirum est notare quam
      littera gothica, quam nunc putamus parum claram, anteposuerit
      litterarum formas humanitatis per seacula quarta decima et
      quinta decima. Eodem modo typi, qui nunc nobis videntur parum
      clari, fiant sollemnes in futurum.
   </div>

   <div id="comments" class="comments">
   <?php
      // Format the query
      $query = sprintf("SELECT * FROM comments WHERE page='%s' ".
                       "ORDER BY createdate",
                       mysql_real_escape_string($_SERVER['PHP_SELF']));
```

```
         // Select the records
         $resource = mysql_query($query);
         if ($resource) {
            while ($record = mysql_fetch_object($resource)) {
               echo '<div class="commenttitle">Comment by '.$record->user.
                    ' on '.$record->createdate.'</div>';
               echo '<div class="commenttext">'.$record->comment.'</div>';
            }
         } else {
            // No records found
            echo 'No comments';
         }
      ?>
      </div>

      <div id="newcomments" class="newcomments">
         <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
            <div class="entry">
               E-mail address: <input type="text" id="email" name="email"
                                                          class="email" />
            </div>
            <div class="entry">
               Comment: <textarea id="commenttext" name="commenttext"
                                           class="commenttext"></textarea>
            </div>
            <div class="entry">
               <input type="submit" value="Add" />
            </div>
            <input type="hidden" id="ipaddr" name="ipaddr"
                           value="<?php echo $_SERVER['REMOTE_ADDR']; ?>" />
         </form>
      </div>
   </body>
   </html>

   <!-- Close the database -->
   <?php mysql_close($dbLink); ?>
```

And the stylesheet contains the following:

```
    .header {
            width: 100%;
            background-color: lightgrey;
    }
    .content {
            margin-left: 50px;
            margin-right: 50px;
            margin-bottom: 20px;
    }
    .comments {
            margin-left: 50px;
            margin-right: 50px;
            margin-bottom: 20px;
            border-top: 3px groove lightgrey;
            border-bottom: 3px groove lightgrey;
    }
    .newcomments {
            margin-left: 50px;
            margin-right: 50px;
            margin-bottom: 20px;
    }
    .entry {
            vertical-align: top;
            text-align: center;
    }
    .commenttitle {
            text-align: left;
    }
    .commenttext {
            text-align: left;
```

```
                margin-left: 50px;
                margin-bottom: 20px;
        }
        .email {
                width: 80%;
                margin-left: 10px;
                margin-right: 50px;
                margin-bottom: 10px;
        }
        .commenttext {
                width: 83%;
                margin-left: 10px;
                margin-right: 50px;
                margin-bottom: 10px;
        }
```

What next?

Of course, this doesn't contain any validation for the input fields – it's possible to create a comment with no E-mail and/or no comment text. You should add JavaScript to validate this at the browser and again at the server. You could also extend it to manage the user's IP address – if you get repeated SPAM from a particular address, you could block that address from posting comments. Another possible extension is the use of *Captcha* to make sure that the poster is a human rather than another computer.

Also, there is a serious security hole where the content of the database are displayed directly on the page, permitting XSS (cross-site scripting) attacks. How would you protect against this?

I'd like to extend this with a parallel version that explains how to do this in other languages, such as Java, Perl and Python, depending on how useful people find this.

```
<!DOCTYPE HTML>
<html>
<head>
   <title>First page of comments application</title>
<?php
   // Link to the database server
   $db = new PDO('mysql:host=localhost;dbname=mydatabase;charset=UTF-8', 'root', 'Me32316xy');

   // If an error occurs, we want to catch it as an "exception"
   $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

   // If this is in response to the "Add" button, insert the new record
   if ($_SERVER['REQUEST_METHOD'] === 'POST') {
      $author  = $_REQUEST['email'];
      $comment = $_REQUEST['commenttext'];
      try {
         $stmt = $db->prepare("INSERT INTO comments(user, comment, page)
VALUES(:field1,:field2,:field3)");
         $stmt->execute(array(':field1' => $author, ':field2' => $comment, ':field3' =>
$_SERVER['PHP_SELF']));
      } catch(PDOException $ex) {
         echo "An Error occured when adding a comment: " . $ex.getMessage();
      }
   }
?>

   <meta charset="utf-8">
   <link rel="stylesheet" href="comments.css">
   <script src="comments.js"></script>
</head>
<body class="defaults">
   <h1 class="header">My First Page</h1>
   <div id="pagecontent" class="content">
      Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed
      diam nonummy nibh euismod tincidunt ut laoreet dolore magna
      aliquam erat volutpat. Ut wisi enim ad minim veniam, quis
      nostrud exerci tation ullamcorper suscipit lobortis nisl ut
      aliquip ex ea commodo consequat.  Duis autem vel eum iriure
      dolor in hendrerit in vulputate velit esse molestie consequat,
      vel illum dolore eu feugiat nulla facilisis at vero eros et
      accumsan et iusto odio dignissim qui blandit praesent luptatum
      zzril delenit augue duis dolore te feugait nulla facilisi.
      Nam liber tempor cum soluta nobis eleifend option congue
      nihil imperdiet doming id quod mazim placerat facer possim
      assum. Typi non habent claritatem insitam; est usus legentis
      in iis qui facit eorum claritatem. Investigationes
      demonstraverunt lectores legere me lius quod ii legunt saepius.
      Claritas est etiam processus dynamicus, qui sequitur
      mutationem consuetudium lectorum. Mirum est notare quam
      littera gothica, quam nunc putamus parum claram, anteposuerit
      litterarum formas humanitatis per seacula quarta decima et
      quinta decima. Eodem modo typi, qui nunc nobis videntur parum
      clari, fiant sollemnes in futurum.
   </div>

   <div id="comments" class="comments">
   <?php
      try {
         foreach ($db->query('SELECT * FROM comments WHERE page="' . $_SERVER['PHP_SELF'] . '" ORDER BY
createdate') as $row) {
            echo '<div class="commenttitle">Comment by ' . $row['user'] .
                 ' on ' . $row['createdate'] . '</div>';
            echo '<div class="commenttext">' . $row['comment'] . '</div>';
         }
         $db = null;
      } catch(PDOException $ex) {
         echo "An Error occured when reading comments: " . $ex->getMessage();
      }
   ?>
   </div>

   <div id="newcomments" class="newcomments">
      <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
         <div class="entry">
            E-mail address: <input type="text" id="email" name="email"
                                          class="email" />
         </div>
         <div class="entry">
            Comment: <textarea id="commenttext" name="commenttext"
                                   class="commenttext"></textarea>
         </div>
         <div class="entry">
            <input type="submit" value="Add" />
         </div>
         <input type="hidden" id="ipaddr" name="ipaddr"
                    value="<?php echo $_SERVER['REMOTE_ADDR']; ?>" />
```

```
        </form>
    </div>
</body>
</html>
```