Server Side development with PHP

MICC / University of Florence
Daniele Pezzatini

Outline

1. Introduction
2. PHP installation
3. Variables, Loops and Functions
4. Passing data through pages: GET and POST method
5. Cookies and Sessions
6. PHP and MySQL database
7. Security best practices
8. Resources

Example: Create a simple CRUD application - The shopping list

** slide inspired by http://www.w3schools.com/php/*

Introduction

What is PHP?

- PHP stands for PHP: Hypertext Preprocessor
- PHP is a server-side scripting language, like ASP
- PHP scripts are executed on the server
- PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.)
- PHP is an open source software
- PHP is free to download and use

http://www.php.net/

Structure of PHP file

PHP pages contain HTML with embedded code that does "something" (in this case, output "Hi, I'm a PHP script!"). The PHP code is enclosed in special start and end processing instructions **<?php** and **?>** that allow you to jump into and out of "PHP mode."

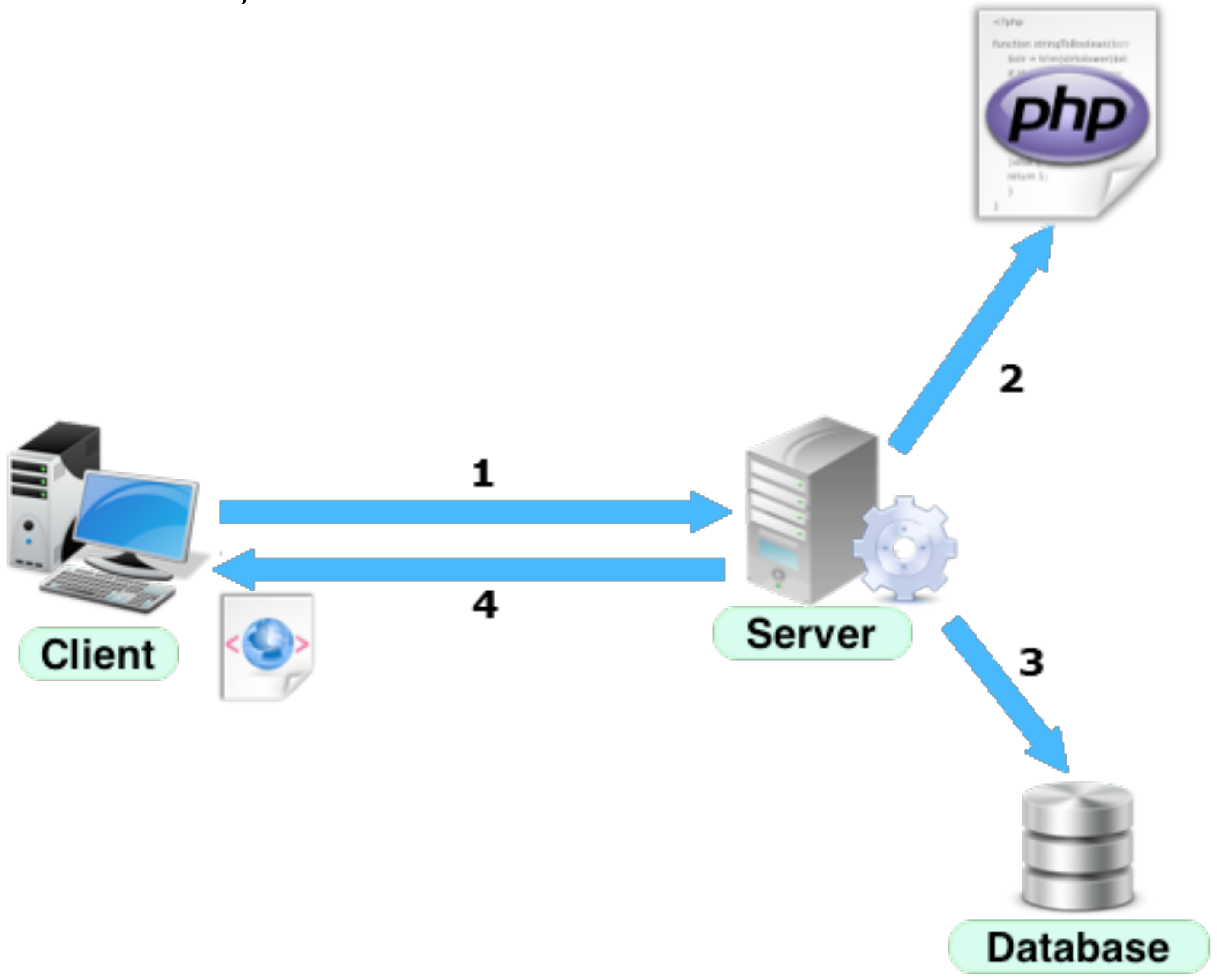
PHP files have a file extension of **.php**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <title>Example</title>
    </head>
    <body>

        <?php
            echo "Hi, I'm a PHP script!";
        ?>

    </body>
</html>
```

Server Side developing

What distinguishes PHP from something like client-side JavaScript is that the code is executed on the server, generating HTML which is then sent to the client.

1) The server receive from a client a request of a PHP file
2) The PHP file is executed by a **web server** (e.g. Apache).
3) If needed, data from a database are read.
4) The results, typically an HTML document, is then send to the client.



Installation

For developing and testing purpose, download and install on of this free all-in-one packages

- WAMP (WIN)
- EasyPHP (WIN)
- MAMP (Mac)
- xampp (WIN)

Linux users: search online for “**LAMP install**”. LAMP stands for **L**inux **A**pache **M**ySQL and **P**HP

Basic PHP Syntax

A PHP script always starts with **<?php** and ends with **?>**. A PHP script can be placed anywhere in the document.

On servers with shorthand-support, you can start a PHP script with <? and end with ?>.

For maximum compatibility, it’s recommend to use the standard form (<?php) rather than the shorthand form.

Below, we have an example of a simple PHP script that sends the text "Hello World" back to the browser:

```
<html>
  <body>

  <?php
    echo "Hello World";
  ?>

  </body>
</html>
```

Each code line in PHP must end with a semicolon. The semicolon is a separator and is used to distinguish one set of instructions from another.

Comments in PHP

In PHP, we use **//** to make a one-line comment or **/*** and ***/** to make a comment block:

```
<html>
<body>

<?php
//This is a comment

/*
This is
a comment
block
*/
?>

</body>
</html>
```

PHP Variables

Rules for PHP variable names:

- Variables in PHP starts with a $ sign, followed by the name of the variable
- The variable name must begin with a letter or the underscore character
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- A variable name should not contain spaces
- Variable names are case sensitive (y and Y are two different variables)

```php
<?php
$txt="Hello World!";
$x=16;
?>
```

In PHP, a variable does not need to be declared before adding a value to it.
In the example above, notice that we did not have to tell PHP which data type the variable is.
PHP automatically converts the variable to the correct data type, depending on its value.
In a strongly typed programming language, you have to declare (define) the type and name of the variable before using it.

String Variables in PHP

String variables are used for values that contain characters.
Below, the PHP script assigns the text "Hello World" to a string variable called $txt:

```
<?php
$txt="Hello World";
echo $txt;
?>
```

The output of the code above will be:

```
Hello World
```

The Concatenation Operator

The concatenation operator (.) is used to put two string values together.
To concatenate two string variables together, use the concatenation operator:

```
<?php
$txt1="Hello World!";
$txt2="What a nice day!";
echo $txt1 . " " . $txt2;
?>
```

The output of the code above will be:

```
Hello World! What a nice day!
```

Using quotes

Choosing whether to use double quotes or single quotes is not just a personal preference. There is an important difference in the way that PHP handles them:

- Anything between **single quotes** is treated literaly as **text**;
- **Double quotes** act as a signal to **process** variables and special characters.

```
$name = 'Dolly ';
echo 'Hello, $name';
```

The example above will output: `Hello, $name`

```
$name = 'Dolly ';
echo “Hello, $name”;
```

The example above will output. `Hello,Dolly`

String functions in PHP

The **strlen()** function is used to return the length of a string.
Let's find the length of a string:

```
<?php
echo strlen("Hello world!");
?>
```

The output of the code above will be `12`

The **join**() function returns a string from the elements of an array.

```
<?php
$arr = array('Hello','World!','Beautiful','Day!');
echo join(" ",$arr);
?>
```

The output of the code above will be: `Hello World! Beautiful Day!`

For a complete reference of all string functions: http://php.net/manual/en/ref.strings.php.

IF/ELSE statements

Conditional statements are used to perform different actions based on different conditions.

```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri") {
  echo "Have a nice weekend!";
}
else {
  echo "Have a nice day!";
}
?>

</body>
</html>
```

SWITCH statements

Use the switch statement to select one of many blocks of code to be executed.

```
<?php
$x=1;
switch ($x)
{
case 1:
  echo "Number 1";
  break;
case 2:
  echo "Number 2";
  break;
case 3:
  echo "Number 3";
  break;
default:
  echo "No number between 1 and 3";
}
?>
```

PHP Arrays

An array can hold all your variable values under a single name. And you can access the values by referring to the array name.

Each element in the array has its own index so that it can be easily accessed.

In PHP, there are three kind of arrays:

- **Numeric array** - An array with a numeric index
- **Associative array** - An array where each ID key is associated with a value
- **Multidimensional array** - An array containing one or more arrays

Numeric Arrays

A numeric array stores each array element with a numeric index.

There are two methods to create a numeric array.

In the following example the index are automatically assigned (the index starts at 0):

```
$cars=array("Saab","Volvo","BMW","Toyota");
```

In the following example we assign the index manually:

```
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
```

In the following example you access the variable values by referring to the array name and index:

```
<?php
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
echo $cars[0] . " and " . $cars[1] . " are Swedish cars.";
?>
```

The code above will output: **`Saab and Volvo are Swedish cars`**.

Associative Arrays

An associative array, each ID key is associated with a value.

When storing data about specific named values, a numerical array is not always the best way to do it.

With associative arrays we can use the values as keys and assign values to them.

In this example we use an array to assign ages to the different persons:

```
$ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34);
```

This example shows a different way of creating the same array:

```
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";
```

The ID keys can be used in a script:

```
<?php
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";

echo "Peter is " . $ages['Peter'] . " years old.";
?>
```

The code above will output: `Peter is 32 years old.`

Multidimensional Arrays

In a multidimensional array, each element in the main array can also be an array. And each element in the sub-array can be an array, and so on.

In this example we create a multidimensional array, with automatically assigned ID keys. On the right it's shown how the array would look like this if written to the output using **print_r()**:

```
$families = array
  (
  "Griffin"=>array
    (
    "Peter",
    "Lois",
    "Megan"
    ),
  "Quagmire"=>array
    (
    "Glenn"
    ),
  "Brown"=>array
    (
    "Cleveland",
    "Loretta",
    "Junior"
    )
 );
```

```
Array
(
[Griffin] => Array
  (
  [0] => Peter
  [1] => Lois
  [2] => Megan
  )
[Quagmire] => Array
  (
  [0] => Glenn
  )
[Brown] => Array
  (
  [0] => Cleveland
  [1] => Loretta
  [2] => Junior
  )
)
```

Lets try displaying a single value from the array above:

```
echo "Is " . $families['Griffin'][2] . " a part of the Griffin family?";
```

The code above will output: `Is Megan a part of the Griffin family?`

PHP Loops

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task like this.

In PHP, we have the following looping statements:

- **while** - loops through a block of code while a specified condition is true
- **do...while** - loops through a block of code once, and then repeats the loop as long as a specified condition is true
- **for** - loops through a block of code a specified number of times
- **foreach** - loops through a block of code for each element in an array

Example: This **for** loop will starts with i=1 and will continue to run as long as i is less than, or equal to 5. i will increase by 1 each time the loop runs:

```
<?php
for ($i=1; $i<=5; $i++)
  {
  echo "The number is " . $i . "<br />";
  }
?>
```

Output:

```
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
```

Passing data - GET method

The predefined **$_GET** variable is used to collect values in a form with method="get"

Information sent from a form with the GET method is visible to everyone (it will be displayed in the browser's address bar) and has limits on the amount of information to send.

```
<form action="welcome.php" method="get">
  Name: <input type="text" name="fname" />
  Age: <input type="text" name="age" />
  <input type="submit" />
</form>
```

When the user clicks the "Submit" button, the URL sent to the server could look something like this:

`http://www.example.com/welcome.php?fname=Peter&age=37`

The "welcome.php" file can now use the $_GET variable to collect form data (the names of the form fields will automatically be the keys in the $_GET array):

```
Welcome <?php echo $_GET["fname"]; ?>.<br />
You are <?php echo $_GET["age"]; ?> years old!
```

When using method="get" in HTML forms, all variable names and values are displayed in the URL.

This method should **not be used when sending passwords** or other sensitive information.
However, because the variables are displayed in the URL, it is possible to **bookmark the page.** This can be useful in some cases.

The get method is not suitable for very large variable values. It should not be used with values exceeding 2000 characters.

Passing data - POST method

The predefined **$_POST** variable is used to collect values from a form sent with method="post".

Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.

Note: However, there is an 8 Mb max size for the POST method, by default (can be changed by setting the post_max_size in the php.ini file).

```
<form action="welcome.php" method="post">
  Name: <input type="text" name="fname" />
  Age: <input type="text" name="age" />
  <input type="submit" />
</form>
```

When the user clicks the "Submit" button, the URL will look like this:

```
http://www.example.com/welcome.php
```

The "welcome.php" file can now use the $_POST variable to collect form data (the names of the form fields will automatically be the keys in the $_POST array):

```
Welcome <?php echo $_POST["fname"]; ?>!<br />
You are <?php echo $_POST["age"]; ?> years old.
```

However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

Cookies

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.

A cookie is often used to identify a user.

With PHP, you can both create and retrieve cookie values.

The **setcookie()** function is used to set a cookie. The function must appear BEFORE the <html> tag.

```
<?php
setcookie("user", "Jhon Doe", time()+3600);
?>

<html>
.....
```

Cookies

The PHP **$_COOKIE** variable is used to retrieve a cookie value.

In the example below, we retrieve the value of the cookie named "user" and display it on a page:

```
<?php

// Print a cookie
echo $_COOKIE["user"];

?>
```

In the following example we use the **isset()** function to find out if a cookie has been set:

```
<?php
if (isset($_COOKIE["user"]))
  echo "Welcome " . $_COOKIE["user"] . "!<br />";
else
  echo "Welcome guest!<br />";
?>
```

When **deleting** a cookie you should assure that the expiration date is in the past.

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

PHP Sessions

When you are working 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 one problem: the web server does not know who you are and what you do because the HTTP protocol doesn't **maintain state**.

A PHP session solves this problem by allowing you to **store user information on the server** for later use (i.e. username, shopping items, etc). However, session information is temporary and will be deleted after the user has left the website. If you need a permanent storage you may want to store the data in a database.

Sessions work by creating a unique id (UID) for each visitor and store variables based on this UID. The UID is either stored in a cookie or is propagated in the URL.

Starting a PHP Session

Before you can store user information in your PHP session, you must first start up the session. The **session_start()** function must appear BEFORE the <html> tag:

```
<?php session_start(); ?>

<html>
<body>
..
```

PHP Sessions

The correct way to store and retrieve session variables is to use the PHP **$_SESSION** variable:

```
<?php
session_start();
// store session data
$_SESSION['views']++;
?>

<html>
<body>

<?php
//retrieve session data
echo "Pageviews=". $_SESSION['views'];
?>

</body>
</html>
```

PHP Sessions

If you wish to delete some session data, you can use the unset() or the session_destroy() function.

The **unset**() function is used to free the specified session variable:

```
<?php
unset($_SESSION['views']);
?>
```

You can also completely destroy the session by calling the **session_destroy**() function. It will reset your session and you will lose all your stored session data.

```
<?php
session_destroy();
?>
```

PHP and MySQL

MySQL is the world's most used relational database management system (RDBMS) that runs as a server providing multi-user access to a number of databases.

The MySQL development project has made its source code available under the terms of the GNU General Public License, as well as under a variety of proprietary agreements.

SQL (Structured Query Language) is a language designed for interacting with RDBMS like MySQL, Oracle, Sqlite etc.

official website http://mysql.com/

PHP and MySQL - Connection

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

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

The **mysql_select_db()** function sets the active MySQL database.

```
$con = mysql_connect( $servername, $username, $password);
$db_selected = mysql_select_db("test_db", $con);
```

PHP and MySQL - Read records

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

We use the mysql_fetch_array() function to return the first row from the recordset as an array. Each call to mysql_fetch_array() returns the next row in the recordset. The while loop loops through all the records in the result set.

```
<?php
$con = mysql_connect("localhost","peter","abc123") or die('Could not connect');


mysql_select_db("my_db", $con);

$result = mysql_query("SELECT * FROM Persons");

while($row = mysql_fetch_array($result))
  {
  echo $row['FirstName'] . " " . $row['LastName'];
  }

?>
```

PHP and MySQL - Writing records

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

The **mysql_affected_rows()** function returns the number of affected rows in the previous MySQL operation. It can be useful to check if the operation ware concluded successfully.

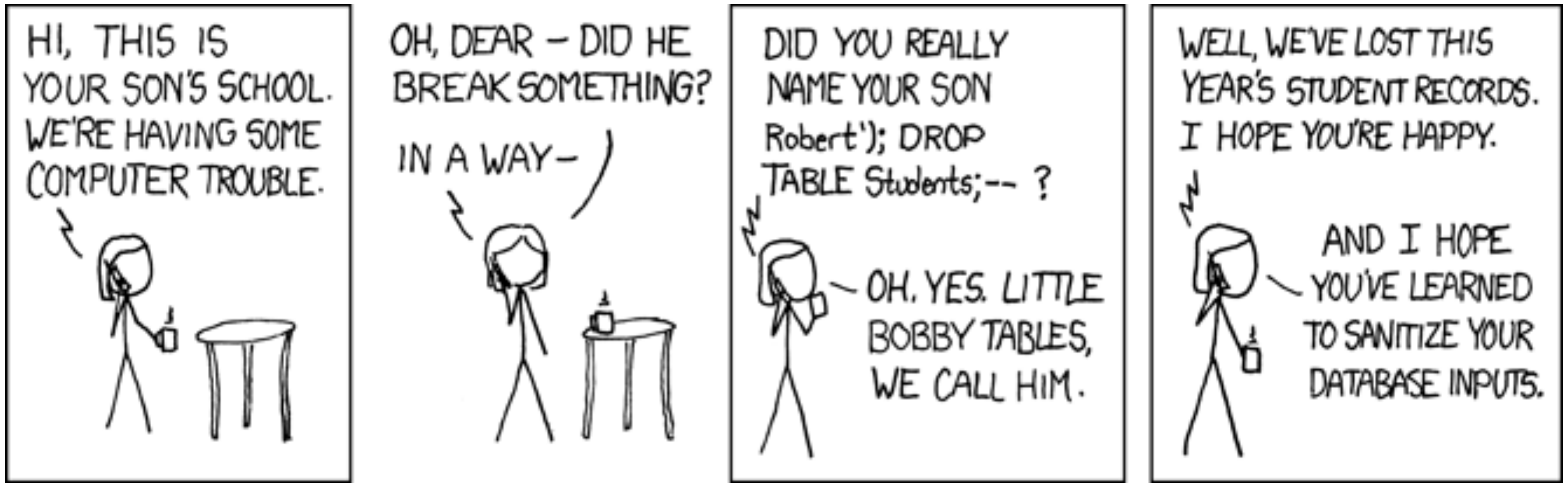
```
$con = mysql_connect("localhost","root","root") or die('Could not connect’);


mysql_select_db("my_db", $con);

$sql="INSERT INTO Persons (FirstName, LastName, Age)
      VALUES('”.$_POST[firstname].”','”.$_POST[lastname].”','”.$_POST[age].”')";

mysql_query($sql);

if ( mysql_affected_rows())
  echo "OK";
else
  echo "Error";
```

A full list of MySQL functions can be found here: http://php.net/manual/en/ref.mysql.php

PHP Security

There are some very simple measures you can take to protect your application from potential abuse. It's very important to **sanitize external data** (user inputs, cookies,external web servers etc.) before processing them or using as database inputs.

Assume everything is dirty until proven clean. Filtering all data from external sources is probably the most important security measure you can take. This can be as easy as running some simple built-in functions on your variables.


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

PHP Security

EXAMPLE

The school apparently stores the names of their students in a table called Students. When a new student arrives, the school inserts his/her name into this table. The code doing the insertion might look as follows:

```
$sql = " INSERT INTO Students (Name) VALUES ('" . $studentName . "'); ";
mysql_query($sql);
```

The first line creates a string containing an SQL INSERT statement. The content of the $studentName variable is glued into the SQL statement. The second line sends the resulting SQL statement to the database. The pitfall of this code is that outside data, in this case the content of $studentName, becomes part of the SQL statement.

First let's see what the SQL statement looks like if we insert a student named John:

```
INSERT INTO Students (Name) VALUES ('John');
```

This does exactly what we want: it inserts John into the Students table.

Now we insert little Bobby Tables, by setting $studentName to ***Robert'); DROP TABLE Students;--***. The SQL statement becomes:

```
INSERT INTO Students (Name) VALUES ('Robert'); DROP TABLE Students;--');
```

This inserts Robert into the Students table. However, the INSERT statement is now followed by a DROP TABLE statement which removes the entire Students table.

PHP Security - Filters

A PHP filter is used to validate and filter data coming from insecure sources.

The PHP filter extension is designed to make data filtering easier and quicker.

Input filtering is one of the most important application security issues.

To filter a variable, use the following filter functions:

```
filter_var( var, filter, options)
```

PHP Security - Filters

A PHP filter is used to validate and filter data coming from insecure sources.

Input filtering is one of the most important application security issues.

To filter a variable, use the following filter functions:

```
filter_var( var, filter, options)
```

There are two kinds of filters:

Validating filters:

- Are used to validate user input
- Strict format rules (like URL or E-Mail validating)
- Returns the expected type on success or FALSE on failure

Sanitizing filters:

- Are used to allow or disallow specified characters in a string
- No data format rules
- Always return the string

PHP Security - Sanitize Input

First we confirm that the input data we are looking for exists.

Then we sanitize the input data using the **filter_var()** function.

In the example below, the input variable "email" is sent to the PHP page:

```
<?php
if( isset ($_GET["email"]) {
  $email = filter_var($_GET["email"],FILTER_SANITIZE_EMAIL);
}
else {
  echo("Input type does not exist");
}
?>
```

The FILTER_SANITIZE_EMAIL filter removes all illegal e-mail characters from a string.

This filter allows all letters, digits and $-_.+!*'{}|^~[]`#%/?@&=

E.G. Filtering **`some(one)@exa\\mple.com`** will output **`someone@example.com`**

PHP Security - Validate Input

After sanitizing, we have to verify that the value is in a correct format.

In the example we use the **filter_var()** function for validating an email.

```
<?php

    if (filter_var($email, FILTER_VALIDATE_EMAIL)){
         echo "Email is valid";
    }
    else{
         echo "Email is not valid";
    }
}

?>
```

The FILTER_VALIDATE_EMAIL filter validates value as an e-mail address.

E.G. Filtering **`someone@example_com`** will produce the output “Email is not valid”

Filtering **`someone@example.com`** will produce the output “Email is valid”

PHP Security - SQL escaping

The **mysql_real_escape_string()** function escapes special characters in a string for use in an SQL statement

This function returns the escaped string on success, or FALSE on failure.

```
<?php

$user = mysql_real_escape_string($user);

$pwd = mysql_real_escape_string($pwd);

$sql = "SELECT * FROM users WHERE user='" . $user . "' AND password='" . $pwd . "'"

?>
```

If you don't execute the **mysql_real_escape_string()** function, using `' OR ''='` as password value will produce the following SQL statement:

```
SELECT * FROM users WHERE user='john' AND password='' OR ''=''
```

This means that anyone could log in without a valid password!

Resources

Officials
http://php.net
http://www.mysql.com/
http://www.w3schools.com/php/

Video tutorial
http://blog.themeforest.net/screencasts/diving-into-php-video-series/

Security
http://us3.php.net/manual/en/security.php
http://www.ultramegatech.com/2009/08/5-basic-php-security-tips/
http://coding.smashingmagazine.com/2010/10/18/common-security-mistakes-in-web-applications/

Object Oriented Programming
http://php.net/manual/en/language.oop5.php
http://net.tutsplus.com/tutorials/php/object-oriented-php-for-beginners/

Create a simple CRUD application - The shopping list

CRUD application

In computer programming **create, read, update and delete** (CRUD) are the four basic functions of persistent storage.

We are going to implement a simple shopping list application in order to show some simple best practices in PHP programming

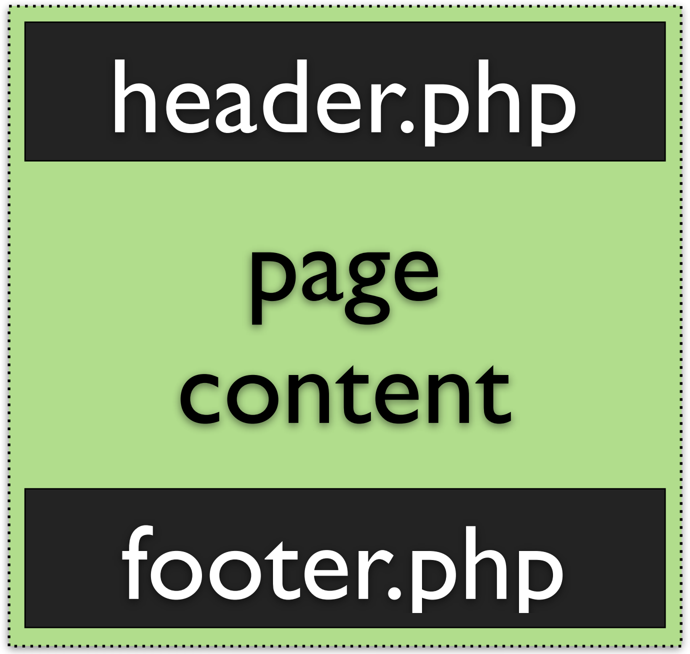
Full code available here: http://www.micc.unifi.it/pezzatini/downloads/crud_ppm.zip

Page structure

All the code that is common to all the pages could be moved in external files.

In our case we will create an **header.php** and a **footer.php** files that must be included in all our pages.

All our pages will have the following structure:



```
<?php
include('header.php');

// insert all the page code here

include('footer.php');

?>
```

Header and footer

In **header.php** we open the HTML head and body tags.
This file will be included at the beginning of every file.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
    <title>Shopping list</title>
    <link rel="stylesheet" type="text/css" href="style.css" />
</head>

<body>
 <div id="main">
        <h2> Shopping List </h2>
```

In **footer.php** we close the HTML body tag and add some copyrights information.
This file will be included at the end of every file.

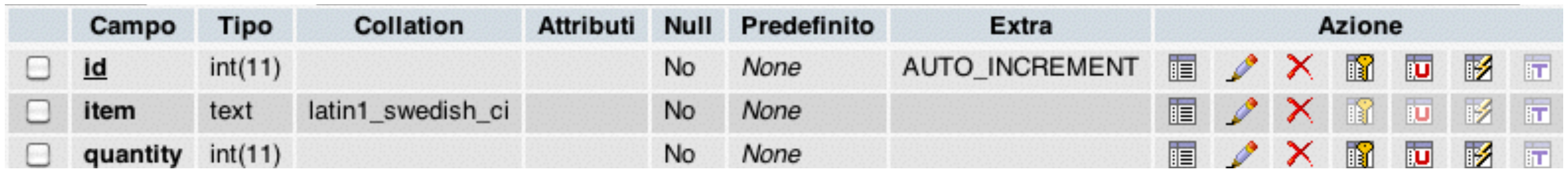
```
 <div id="credits">
    <p>  &copy; <?= date('Y') ?> - MICC </p>
 </div>

</div>

</body>

</html>
```

Define the data model

We create a new database and a new table, called **shopping_list.**

The table has three field: an **id**, an **item** name and a **quantity**.

```
CREATE TABLE IF NOT EXISTS `shopping_list` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `item` text NOT NULL,
  `quantity` int(11) NOT NULL,
  PRIMARY KEY (`id`)
);
```



	Campo	Tipo	Collation	Attributi	Null	Predefinito	Extra	Azione
☐	id	int(11)			No	*None*	AUTO_INCREMENT	
☐	item	text	latin1_swedish_ci		No	*None*		
☐	quantity	int(11)			No	*None*		

Configure database connection

Create a configuration file (config.php) where the database connection will be established.
We are going to include this file in every page of our application

```
<?php
$db_host = 'localhost';
$db_user = 'root';
$db_pass = 'root';
$db_name = 'crud_ppm';

$db_connect = mysql_connect($db_host,$db_user , $db_pass);
if (!$db_connect) {
    die('Not connected : ' . mysql_error());
}
if (! mysql_select_db($db_name) ) {
    die ('Can\'t use foo : ' . mysql_error());
}
?>
```

Listing items

In our index.php we will select from the DB all the values and display them in a table, using **`mysql_query`** and **`mysql_fetch_array`**.

For each item, we add links for the the **edit** and **delete** page passing the id of the item.

```
<?php
include('config.php');
?>
<table>
 <tr>
    <td><b>Id</b></td>
    <td><b>Item</b></td>
    <td><b>Quantity</b></td>
 </tr>

<?php

 $result = mysql_query("SELECT * FROM `shopping_list`") or trigger_error(mysql_error());
 while($row = mysql_fetch_array($result)){
 ?>
   <tr>
    <td ><?= strip_tags($row['id']) ?></td>
    <td ><?= strip_tags($row['item']) ?></td>
    <td ><?= strip_tags($row['quantity']) ?></td>
    <td ><a href='edit.php?id=<?= $row['id'] ?> '>Edit</a></td>
    <td><a href='delete.php?id=<?= $row['id'] ?> '>Delete</a></td>
 </tr>
<?php
} //end of while
?>
</table>
```

Adding items

We create a page for adding new items (new.php). The HTML of the page will display an input form.
At the beginning of the page, we check if form data have been send. If so, we prepare them for be insert in a SQL statement.

```
<?php
include('config.php');

if (isset($_POST['submitted'])) {
 foreach($_POST AS $key => $value) {
    $_POST[$key] = mysql_real_escape_string($value);
 }
 if (filter_var( $_POST['quantity'], FILTER_VALIDATE_INT) ){
    $sql = "INSERT INTO shopping_list(item,quantity)
            VALUES ('".$_POST[item']."','".$_POST['quantity']."') ";

   mysql_query($sql) or die(mysql_error());
 ?>
 Added row.<br />
 <a href='index.php'>Back To Listing</a>

<?php
 }
 else{
    echo "Quantity must be an integer number";
 }
}
?>
<form action='' method='POST'>
   Item: <textarea name='item'></textarea>
   Quantity: <input type='text' name='quantity'/>
   <input type='submit' value='Add Row' />
   <input type='hidden' value='1' name='submitted' />
</form>
```

Edit items

We create a page for editing existing items (edit.php). The HTML of the page will display an input form already filled with current values.
The **id** of the item is retrieved form the url of the page ($_GET value)

```
if (isset($_GET['id']) ) {
   $id = (int) $_GET['id'];
   if (isset($_POST['submitted'])) {
    foreach($_POST AS $key => $value) {
            $_POST[$key] = mysql_real_escape_string($value);
    }

    if ( filter_var( $_POST['quantity'], FILTER_VALIDATE_INT) ){
            $sql = "UPDATE shopping_list
                    SET item ='".$_POST['item']."', quantity = '".$_POST['quantity']."'
                    WHERE id = ".$id."'";

             mysql_query($sql) or die(mysql_error());
            echo (mysql_affected_rows()) ? "Edited row" : "Nothing changed";
            echo "<a href='index.php'>Back To Listing</a>";
    }
    else{
        echo "Quantity must be an integer number";
    }
   }
   $row = mysql_fetch_array ( mysql_query("SELECT * FROM `shopping_list`
                                            WHERE `id` = '".$id."' "));
?>

<form action='' method='POST'>
    Item:<textarea name='item'><?= strip_tags($row['item']) ?></textarea>
    Quantity:<input type='text' name='quantity' value='<?= strip_tags($row['quantity']) ?>'/>
    <input type='submit' value='Edit Row' />
    <input type='hidden' value='1' name='submitted' />
</form>
```

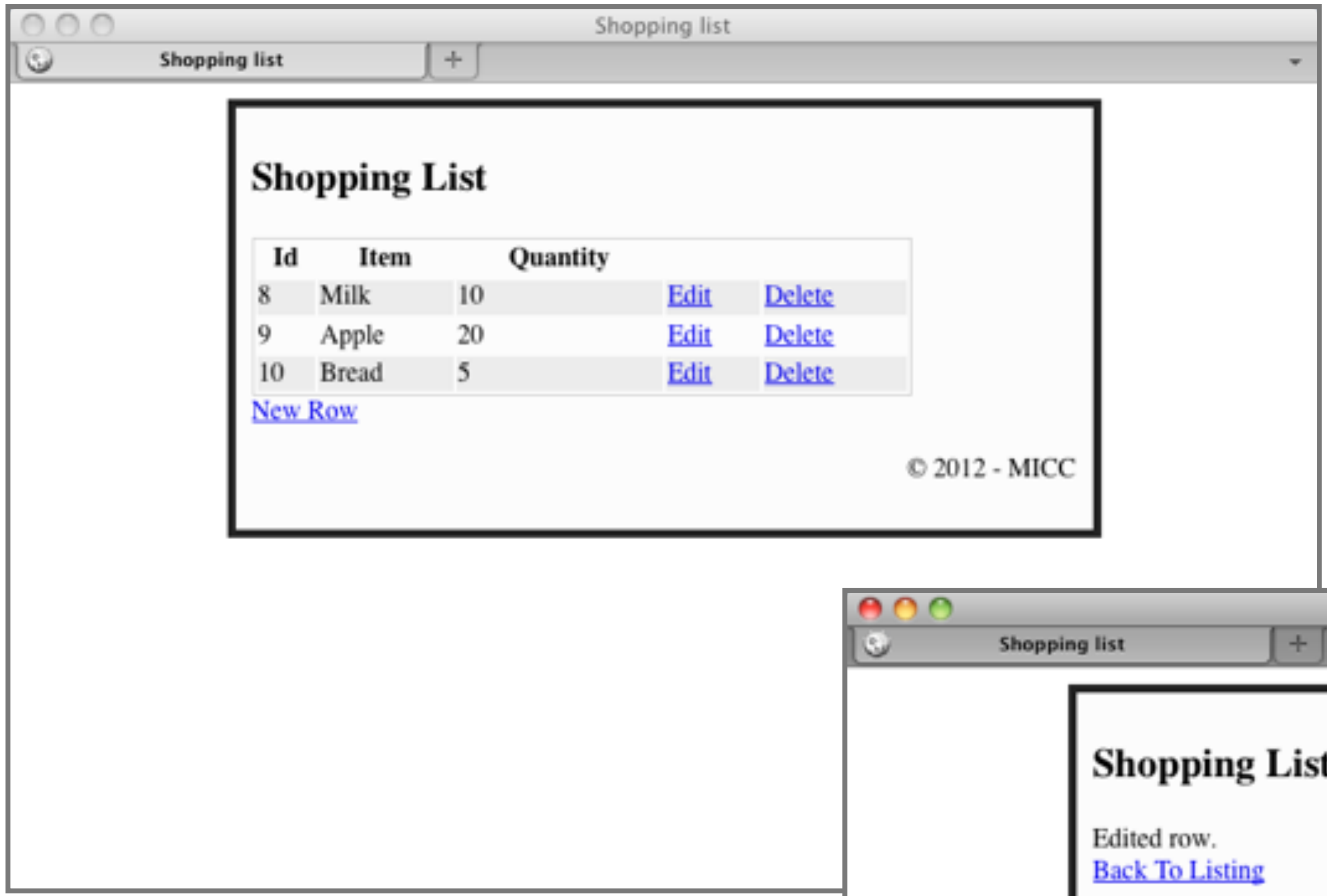
Delete items

We create a page for deleting existing items (delete.php).

The **id** of the item is retrieved form the url of the page ($_GET value)

```
$id = (int) $_GET['id'];

mysql_query("DELETE FROM shopping_list WHERE id = '$id' ") ;

echo (mysql_affected_rows()) ? "Row deleted." : "Nothing deleted.";

?>

<a href='index.php'>Back To Listing</a>
```

Results



Shopping list

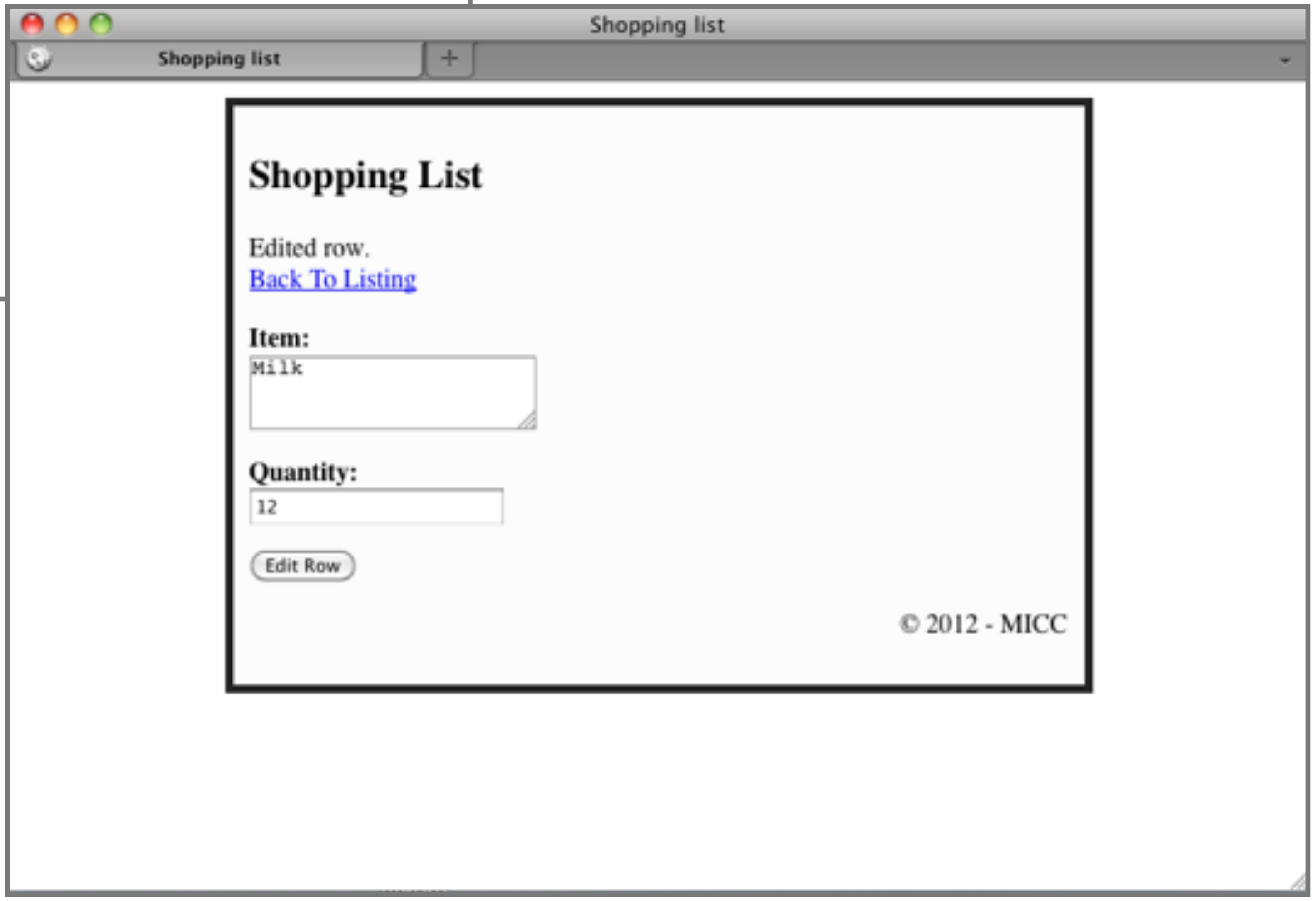
Shopping list

Shopping List

Id	Item	Quantity		
8	Milk	10	Edit	Delete
9	Apple	20	Edit	Delete
10	Bread	5	Edit	Delete

New Row

© 2012 - MICC



Shopping list

Shopping list

Shopping List

Edited row.
Back To Listing

Item:
Milk

Quantity:
12

© 2012 - MICC