S@ifur's Solutions

69/B Monwara Plaza (Level-5)
East Panthapath
Dhaka-1205
Cell: 01197 11 82 77

Lecture Sheet on "+MySQL"
Day-12: (PHP)

Cookies and Sessions **Time: 2** hours

This course provides the learner with information about using cookies and sessions with PHP to overcome the statelessness of the Web.

Objectives	Topics
❖ Use cookies ❖ Use sessions ❖ Improve session security ❖ Manage the relationship between cookies and sessions	• Setting cookies • Accessing cookies • Setting cookie parameters • Deleting cookies • Setting session variables • Accessing session variables • Deleting session variables • Changing session behavior • Changing the session cookie settings

Cookies allow the webmaster to store information about the site visitor on their computer to be accessed again the next time they visit. One common use of cookies is to store your username and password on your computer so you don't need to login again each time you visit a website. Cookies can also store other things such as your name, last visit, shopping cart contents, etc.

The main difference between a cookie and a session is that a cookie is stored on your computer, and a session is not. Although cookies have been around for many years and most people do have them enabled, there are some who do not. Cookies can also be removed by the user at any time, so don't use them to store anything too important.

Setcookie: setcookie - Send a cookie

Syntax **setcookie(name,value,expire,path,domain,secure)**

Parameter	Description
name	Required. Specifies the name of the cookie
value	Required. Specifies the value of the cookie
expire	Optional. Specifies when the cookie expires. time()+3600*24*30 will set the cookie to expire in 30 days. If this parameter is not set, the cookie will expire at the end of the session (when the browser closes).
path	Optional. Specifies the server path of the cookie If set to "/", the cookie will be available within the entire domain. If set to "/test/", the cookie will only be available within the test directory and all sub-directories of test. The default value is the current directory that the cookie is being set in.
domain	Optional. Specifies the domain name of the cookie. To make the cookie available on all subdomains of example.com then you'd set it to ".example.com". Setting it to www.example.com will make the cookie only available in the www subdomain
secure	Optional. Specifies whether or not the cookie should only be transmitted over

	a secure HTTPS connection. TRUE indicates that the cookie will only be set if a secure connection exists. Default is FALSE.

Return Values

If output exists prior to calling this function, **setcookie()** will fail and return **FALSE**. If **setcookie()** successfully runs, it will return **TRUE**. This does not indicate whether the user accepted the cookie.
A cookie is set with the following code: **setcookie(name, value, expiration)**

```
<?php
$Month = 2592000 + time();
setcookie(AboutVisit, date("F jS - g:i a"), $Month); //this adds 30 days to the current time

?>
```

The above code sets a cookie in the visitor's browser called "AboutVisit". The cookie sets the value to the current date, and set's the expiration to be be in 30 days (2592000 = 60 seconds * 60 mins * 24 hours * 30 days.)

Now let's retrieve the cookie.

```
<?php
if(isset($_COOKIE['AboutVisit']))
        {
        $last = $_COOKIE['AboutVisit'];
        echo "Welcome back! <br> You last visited on ". $last;
        }
        else
        {
        echo "Welcome to our site!";
        }
?>
```

This code first checks if the cookie exists. If it does, it welcomes the user back and tells them when they last visited. If they are new, it skips this and prints a generic welcome message.

TIP: If you are calling a cooking on the same page you plan to set one - be sure you retrieve it first, before you overwrite it!

To destroy the cookie, simply use *setcookie* again, only set the expiration date to be in the past. This is often done when you 'logout' of a site. Here is an example:

```
<?php
$past = time() - 10;
setcookie(AboutVisit, date("F jS - g:i a"), $past); //this makes the time 10 seconds ago
?>
```

REMEMBER: Cookies need to be set in the header. This means they must be sent **before** any HTML is set to the page, or they will not work.

setcookie() delete example

When deleting a cookie you should assure that the expiration date is in the past, to trigger the removal mechanism in your browser. Examples follow how to delete cookies sent in previous example:

```
<?php
setcookie ("TestCookie", "", time() - 3600);            // set the expiration date to one hour ago
setcookie ("TestCookie", "", time() - 3600, "/~rasmus/", ".example.com", 1);
?>
```

Session

Session support in PHP consists of a way to preserve certain data across subsequent accesses. This

enables you to build more customized applications and increase the appeal of your web site.
A visitor accessing your web site is assigned a unique id, the so-called session id. This is either stored in a cookie on the user side or is propagated in the URL.

The session support allows you to register arbitrary numbers of variables to be preserved across requests. When a visitor accesses your site, PHP will check automatically (if session.auto_start is set to 1) or on your request (explicitly through **session_start()** or implicitly through session_register()) whether a specific session id has been sent with the request. If this is the case, the prior saved environment is recreated.

Starting a Session

A session is a way to store information (in the form of variables) to be used across multiple pages. Unlike a **cookie**, specific variable information is not stored on the users computer. It is also unlike other variables in the sense that we are not passing them individually to each new page, but instead retrieving them from the session we open at beginning of each page.

```
<?php
session_start();        // this starts the session

// this sets variables in the session
$_SESSION['color']='red';
$_SESSION['size']='small';
$_SESSION['shape']='round';
print "Done";
?>
```

The first thing we do with this code, is open the session using session_start(). We then set our first session variables (color, size and shape) to be red, small and round respectively.

Just like with our cookies, the *session_start()* code must be in the header and you can not send anything to the browser before it. It's best to just put it directly after the *<?php* to avoid potential problems.

So how will it know it's me? Most sessions set a cookie on your computer to uses as a key... it will look something like this: 350401be75bbb0fafd3d912a1a1d5e54. Then when a session is opened on another page, it scans your computer for a key. If there is a match, it accesses that session, if not it starts a new session for you.

Using Session Variables

Now we are going to make a second page. We again will start with session_start() (we need this on every page) - and we will access the session information we set on our first page. Notice we aren't passing any variables, they are all stored in the session.

```
<?php
session_start();        // this starts the session

// echo variable from the session, we set this on our other page
echo "Our color value is ".$_SESSION['color'];
echo "Our size value is ".$_SESSION['size'];
echo "Our shape value is ".$_SESSION['shape'];
?>
```

All of the values are stored in the $_SESSION array, which we access here. Another way to show

this is to simply run this code:

```
<?php
session_start();
Print_r ($_SESSION);
?>
```

69/B Monwara Plaza (Level-5)
East Panthapath
Dhaka-1205
Cell: 01197 11 82 77

You can also store an array within the session array. Let's go back to our mypage.php file and edit it slightly to do this:

```
<?php
session_start();

$colors=array('red', 'yellow', 'blue');          // makes an array
$_SESSION['color']=$colors;            // adds it to our session
$_SESSION['size']='small';
$_SESSION['shape']='round';
print "Done";
?>
```

Now let's run this on mypage2.php to show our new information:

```
<?php
session_start();
Print_r ($_SESSION);
echo "<p>";
echo $_SESSION['color'][2];  //echo a single entry from the array

?>
```

Modify or Remove a Session

```
<?php
        session_start();                // you have to open the session to be able to modify or remove it
        $_SESSION['size']='large';      // to change a variable, just overwrite it
        unset($_SESSION['shape']);      //you can remove a single variable in the session
        session_unset();                // or this would remove all the variables in the session, but not the session itself
        session_destroy();              // this would destroy the session variables
?>
```

The code above demonstrates how to edit or remove individual session variables, or the entire session. To change a session variable we just reset it to something else. We can use unset() to remove a single variable, or session_unset() to remove all variables for a session. We can also use session_destroy() to destroy the session completely.

By default a session lasts until the user closes their browser. This can be changed in the php.ini file by change the 0 in *session.cookie_lifetime = 0* to be the number of seconds you want the session to last, or by using session_set_cookie_params().

Session_start() PHP Function

Definition: *Session_start()* is used in PHP to initiate a session on each PHP page. It must be the first thing sent to the browser, or it won't work properly, so it's usually best to place it right after the *<?php* tags. This must be on every page you intend to use sessions on.

```
<?php
session_start();                // this starts the session
$_SESSION['test']='testing';    // this sets variables in the session
print "Done";
?>
```

Session_unset () PHP Function

Definition: *Session_unset ()* is used to remove all variables in a session. You must first open a session to use session_unset (). You should use this function instead of *unset($_session)* to remove the entire session, to avoid problems with global variables. It should usually be used with session_destroy() to more thoroughly remove the session, if that is your goal.

```
<?php
session_start();                    // you have to open the session to be able to modify or remove it
session_unset();                    // or this would remove all the variable in the session
session_destroy();                  // destroy the session

?>
```

Session_destroy () PHP Function

Definition: *Session_destroy()* destroys all of the data associated with the current session, however it does not unset any of the global variables associated with the session, nor does it unset the session cookie. It is often used with session_unset() to more thoroughly remove the session.

```
<?php
session_start();                    // you have to open the session first
session_unset();                    // remove all the variables in the session
session_destroy();                  // destroy the session
?>
```

Unset () PHP Function

Definition: *Unset ()* is used to destroy a variable in PHP. It can be used to remove a single variable, multiple variables, or an element from an array. It is phrased as **Unset ($remove)**.

```
<?php
unset($a);                          // remove a single variable
unset($my_array['element']);        // remove a single element in an array
unset($a, $b, $c);                  // remove multiple variables
?>
```

Session_set_cookie_params () PHP Function

Definition: *Session_set_cookie_params ()* to set information about the session cookie, including the duration. It is phrased as session_set_cookie_params (seconds, optional_path, optional_domain, optional_security). This is not as effective as editing the php.ini file, and only lasts for the duration of the script, so you would need to call it on every page before _start().

```
session_set_cookie_params(1200); //sets cookie to 1200 seconds or 20 mins
session_start();
```

Making a secure PHP login Script

Users with shell access to the web server can scan valid session id's if the default /tmp directory is used to store the session data.

The protection against this kind of attack is the IP check. Somebody who has a site (on a shared host with you) can generate valid session for your site.

This is why the checkSession method is used and the session id is recorded in the database.

Somebody may sniff network traffic and catch the cookie. The IP check should eliminate this problem too.

Preparation

You need first to decide what information to store about members, the examples provided will assume almost nothing to make it easier to read.

Database Schema

This is only an example bare structure suitable for online administration, if you want to have registered members you should add more columns.

```
CREATE TABLE member (
id int NOT NULL auto_increment,
username varchar(20) NOT NULL default '',
password char(32) binary NOT NULL default '',
cookie char(32) binary NOT NULL default '',
session char(32) binary NOT NULL default '',
ip varchar(15) binary NOT NULL default '',
PRIMARY KEY (id),
UNIQUE KEY username (username)
);
```

The password and cookie fields are md5 hashes which are always 32 octets long. Cookie is the cookie value that is sent to the user if he/she requests to be remembered, session and ip are respectively the session id and the current IP of the visitor.

Connecting to the Database

```
function &db_connect() {
require_once 'DB.php';
PEAR::setErrorHandling(PEAR_ERROR_DIE);
$db_host = 'localhost';
$db_user = 'shaggy';
$db_pass = 'password';
$db_name = 'shaggy';
$dsn = "mysql://$db_user:$db_pass@unix+$db_host/$db_name";
$db = DB::connect($dsn);
$db->setFetchMode(DB_FETCHMODE_OBJECT);
return $db;
}
```

This function connects to the database returning a pointer to a PEAR database object.

Session Variables

To ease access to the current user's information we register it as session variables but to prevent error messages and set some defaults we use the following function.

```
function session_defaults() {
$_SESSION['logged'] = false;
$_SESSION['uid'] = 0;
$_SESSION['username'] = '';
$_SESSION['cookie'] = 0;
$_SESSION['remember'] = false;
}
```

... with a check like:

```
if (!isset($_SESSION['uid']) ) { session_defaults();  }
```

to set the defaults. Of course session_start must be called before that.

To the Core of the Script

To allow easier integration with other scripts and make things more modular the core script is an object with very simple interface.

```
class User {
var $db = null; // PEAR::DB pointer
var $failed = false; // failed login attempt
var $date; // current date GMT
var $id = 0; // the current user's id
function User(&$db) {
$this->db = $db;
$this->date = $GLOBALS['date'];
i f ($_SESSION['logged']) { $this->_checkSession(); }
elseif (isset($_COOKIE['mtwebLogin'])){ $this->_checkRemembered($_COOKIE['mtwebLogin']);
}
}
```

This is the class definition and the constructor of the object. OK it's not perfectly modular but a date isn't much of a problem. It is invoked like:

```
$date = gmdate("'Y-m-d'");
$db = db_connect();
$user = new User($db);
```

Now to clear the code purpose, we check if the user is logged in. If he/she is then we check the session (remember it is a secure script), if not and a cookie named just for example mtwebLogin is checked - this is to let remembered visitors be recognized.

Logging in Users

To allow users to login you should build a web form, after validation of the form you can check if the user credentials are right with $user->_checkLogin('username', 'password', remember). Username and password should not be constants of course, remember is a boolean flag which if set will send a cookie to the visitor to allow later automatic logins.

```
function _checkLogin($username, $password, $remember) {
$username = $this->db->quote($username);
$password = $this->db->quote(md5($password));
$sql = "SELECT * FROM member WHERE " .
"username = $username AND " .
"password = $password";
```

```
$result = $this->db->getRow($sql);
if ( is_object($result) ) {
$this->_setSession($result, $remember);
return true;
} else {
$this->failed = true;
$this->_logout();
return false;
}
}
```

The function definition should be placed inside the User class definition as all code that follows. The function uses PEAR::DB's quote method to ensure that data that will be passed to the database is safely escaped. The WHERE statement is optimized (the order of checks) because username is defined as UNIQUE.

No checks for a DB_Error object are needed because of the default error mode set above. If there is a match in the database $result will be an object, so set our session variables and return true (successful login). Otherwise set the failed property to true (checked to decide whether to display a login failed page or not) and do a logout of the visitor.

The logout method just executes session_defaults().

Setting the Session

```
function _setSession(&$values, $remember, $init = true) {
$this->id = $values->id;
$_SESSION['uid'] = $this->id;
$_SESSION['username'] = htmlspecialchars($values->username);
$_SESSION['cookie'] = $values->cookie;
$_SESSION['logged'] = true;
if ($remember) {
$this->updateCookie($values->cookie, true);
}
if ($init) {
$session = $this->db->quote(session_id());
$ip = $this->db->quote($_SERVER['REMOTE_ADDR']);
$sql = "UPDATE member SET session = $session, ip = $ip WHERE " .
"id = $this->id";
$this->db->query($sql);
}
}
```

This method sets the session variables and if requested sends the cookie for a persistent login, there is also a parameter which determines if this is an initial login (via the login form/via cookies) or a subsequent session check.

Persistent Logins

If the visitor requested a cookie will be send to allow skipping the login procedure on each visit to the site. The following two methods are used to handle this situation.

```
function updateCookie($cookie, $save) {
$_SESSION['cookie'] = $cookie;
if ($save) {
$cookie = serialize(array($_SESSION['username'], $cookie) );
set_cookie('mtwebLogin', $cookie, time() + 31104000, '/directory/');
} }
```

Checking Persistent Login Credentials

If the user has chosen to let the script remember him/her then a cookie is saved, which is checked via the following method.

```
function _checkRemembered($cookie) {
list($username, $cookie) = @unserialize($cookie);
if (!$username or !$cookie) return;
$username = $this->db->quote($username);
$cookie = $this->db->quote($cookie);
$sql = "SELECT * FROM member WHERE " .
"(username = $username) AND (cookie = $cookie)";
$result = $this->db->getRow($sql);
if (is_object($result) ) {
$this->_setSession($result, true);
}
}
```

This function should not trigger any error messages at all. To make things more secure a cookie value is saved in the cookie not the user password. This way one can request a password for areas which require even higher security.

Ensuring Valid Session Data

```
function _checkSession() {
$username = $this->db->quote($_SESSION['username']);
$cookie = $this->db->quote($_SESSION['cookie']);
$session = $this->db->quote(session_id());
$ip = $this->db->quote($_SERVER['REMOTE_ADDR']);
$sql = "SELECT * FROM member WHERE " .
"(username = $username) AND (cookie = $cookie) AND " .
"(session = $session) AND (ip = $ip)";
$result = $this->db->getRow($sql);
if (is_object($result) ) {
$this->_setSession($result, false, false);
} else {
$this->_logout();
}
}
```

So this is the final part, we check if the cookie saved in the session is right, the session id and the IP address of the visitor. The call to setSession is with a parameter to let it know that this is not the first login to the system and thus not update the IP and session id which would be useless anyway.