PHP Hypertext Preprocessor

Sessions, Cookies and MySQL functions

Stefano Fontanelli

January 19, 2009

stefano.fontanelli@sssup.it

Session Handling

1. Session consists of a way to preserve certain data across subsequent accesses.
2. An unique id (called *session id*) is assigned to each visitor which accesses your web site.
3. It is possible to register arbitrary numbers of variables to be preserved across requests.
4. The constant `SID` is defined by the session extension.
5. `SID` contains the session name and session ID in the form of "name=ID" or empty string if session ID was set in an appropriate session cookie.

Session Handling: `session_start()`

1. `bool session_start ( void )`
2. It creates a session or resumes the current one based on the current *session id*.
3. It is being passed via a request (GET or POST) or a cookie.
4. It returns TRUE if session was started with success otherwise FALSE.
5. You must call `session_start()` (in cookie-based sessions) before anything is outputted to the browser.

Session Handling: `session_start()` (an example)

```
<?php
  if (session_start() == TRUE) {
    echo SID; // print the session id
  } else {
    echo 'Unable to start the session.';
  }
?>
```

Session Handling: register the session variables (1)

An example: *page1.php*

```
<?php

session_start();

$_SESSION['favcolor'] = 'green';
$_SESSION['animal'] = 'cat';
$_SESSION['time'] = time();

// Works if session cookie was accepted
echo '<br /><a href="page2.php">page 2</a>';

// Or maybe pass along the session id, if needed
echo '<br /><a href="page2.php?'  .  SID . '">page 2</a>';

?>
```

Session Handling: register the session variables (2)

An example: *page2.php*

```
<?php

session_start();

echo $_SESSION['favcolor']; // print 'green';
echo $_SESSION['animal']; // print 'cat';
echo $_SESSION['time']; // print the timestamp;

?>
```

Session Handling: free the session variables

`void session_unset ( void )`

This function frees all session variables currently registered.

```
<?php

  session_start();

  echo $_SESSION['favcolor']; // print 'green';
  echo $_SESSION['animal']; // print 'cat';

  session_unset();

  echo $_SESSION['favcolor']; // print nothing;
  echo $_SESSION['animal']; // print nothing;

  unset($_SESSION['animal']); // use this;
  unset($_SESSION); // don't use this;

?>
```

Session Handling: `session_destroy()` (2)

```
bool session_destroy ( void )
```

This function destroys all of the data associated with the current session.

It does not unset any of the global variables associated with the session, or unset the session cookie.

Session Handling: session_destroy() (2)

The next example shows how to kill the session:

```
<?php

  // Initialize the session.
  session_start();
  // Unset all of the session variables.
  $_SESSION = array();

  // If it's desired to kill the session,
  // also delete the session cookie.
  // Note:  This will destroy the session,
  // and not just the session data!

  if (isset($_COOKIE[session_name()])) {
   setcookie(session_name(), '', time()-42000, '/');
  }

  // Finally, destroy the session.
  session_destroy();

?>
```

Cookies (1)

Cookies are a mechanism for storing data in the remote browser and thus tracking or identifying return users.

In PHP it can be possible to access them via $_COOKIE associative array.

It can be possible to set cookies via `setcookie()` function.

Cookies (2)

```
bool setcookie ( string $name [, string $value
                 [, int $expire [, string $path
                 [, string $domain [, bool $secure
                 [, bool $httponly ]]]]]] )
```

It defines a cookie to be sent along with the rest of the HTTP headers.

Like other headers, cookies must be sent before any output.

1. *$name* is the name of the cookie.
2. *$value* is the value of the cookie.
3. *$expire* is the time the cookie expires.
4. *$path* is the path on the server in which cookie will be available on.
5. *$domain* is the domain that the cookie is available.
6. *$secure* indicates that the cookie should only be transmitted over a secure HTTPS connection from the client.
7. *$httponly* when TRUE the cookie will be made accessible only through the HTTP protocol.

Cookies (3)

```
<?php
  // set the cookies.
  setcookie("cookie[three]", "cookiethree");
  setcookie("cookie[two]", "cookietwo");
  setcookie("cookie[one]", "cookieone");

  if (isset($_COOKIE['cookie'])) {
   foreach ($_COOKIE['cookie'] as $name => $value) {
    echo "$name :  $value <br />";
   }
  }
?>
```

MySQL functions

1. These functions allow you to access MySQL database servers.
2. Information about MySQL can be found at http://www.mysql.com

MySQL functions: mysql_connect() (1)

```
resource mysql_connect ([ string $server
                          [, string $username
                          [, string $password
                          [, bool $new_link
                          [, int $client_flags ]]]]] )
```

This function open a connection to a MySQL Server.

Returns a MySQL link identifier on success, or FALSE on failure.

$server the MySQL server.
It can also include a port number. e.g. "hostname:port"
or a path to a local socket e.g. ":/path/to/socket" for the localhost.

$username the username.

$password the password.

MySQL functions: mysql_connect() (2)

```
<?php
  $link = mysql_connect('localhost',
                        'mysql_user',
                        'mysql_password');
  if (!$link) {
   die('Could not connect:  '  .  mysql_error());
  }
  echo 'Connected successfully';
  mysql_close($link);
?>
```

MySQL functions: `mysql_select_db()` (1)

```
bool mysql_select_db ( string $database_name
                       [, resource $link_identifier ] )
```

This function sets the current active database on the server
that's associated with the specified link identifier

$database_name the name of the database that is to be selected.

$link_identifier the MySQL connection.
If the link identifier is not specified,
the last link opened by `mysql_connect()` is assumed.

Returns `TRUE` on success or `FALSE` on failure.

MySQL functions: mysql_select_db() (2)

```
<?php
  $link = mysql_connect('localhost',
                        'mysql_user',
                        'mysql_password');
  if (!$link) {
   die('Could not connect: ' . mysql_error());
  }

  $db_selected = mysql_select_db('foo', $link);

  if (!$db_selected) {
   die ('Can't use foo : ' . mysql_error());
  }

?>
```

MySQL functions: `mysql_query()` (1)

```
resource mysql_query ( string $query
                     [, resource $link_identifier ] )
```

This function sends a unique query to the currently active database on the server that's associated with the specified *$link_identifier*.

Multiple queries are not supported.

$query a SQL query string. It should not end with a semicolon.

$link_identifier the MySQL connection.
If the link identifier is not specified,
the last link opened by `mysql_connect()` is assumed.

Return values:

1. a resource on success for the statements returning resultset (SELECT, SHOW, DESCRIBE, EXPLAIN),
2. `TRUE` on success (for INSERT, UPDATE, DELETE, DROP, ...),
3. `FALSE` on error.

MySQL functions: mysql_query() (2)

```
<?php
  $query = "SELECT * FROM customers WHERE lastname=";
  $query .= mysql_real_escape_string($lastname);
  $result = mysql_query($query);

  if (!$result) {
   $message = 'Invalid query:  '  .  mysql_error();
   $message .= 'Whole query:  '  .  $query;
   die($message);
  }
?>
```

MySQL functions: `mysql_real_escape_string()` (1)

```
string mysql_real_escape_string ( string $unescaped_string
                                  [, resource $link_identifier ] )
```

This function escapes special characters in the *unescaped_string*, taking into account the current character set of the connection so that it is safe to place it in a `mysql_query()`.
If binary data is to be inserted, this function must be used.

$unescaped_string the string that is to be escaped.

$link_identifier the MySQL connection.
If the link identifier is not specified,
the last link opened by `mysql_connect()` is assumed.

Returns the escaped string, or `FALSE` on error.

MySQL functions: mysql_real_escape_string() (2)

An example SQL Injection Attack:

```
<?php

  // $_POST['username'] = 'aidan';
  // $_POST['password'] = "' OR ''='";

  $query = "SELECT *
            FROM users
            WHERE user='$_POST['username']' AND
                  password='$_POST['password']'";

  // SELECT * FROM users
  // WHERE user='aidan' AND password='' OR ''=''

?>
```

MySQL functions: mysql_real_escape_string() (3)

The best practice example:

```
<?php

  // $_POST['username'] = 'aidan';
  // $_POST['password'] = "' OR ''='";

  $username = mysql_real_escape_string($_POST['username']);
  $password = mysql_real_escape_string($_POST['password']);

  $query = "SELECT *
            FROM users
            WHERE user='$username' AND
                  password='$password'";

  // SQL injection doesn't work.
  // SELECT * FROM users
  // WHERE user='aidan' AND password='\' OR \'\'=\''

?>
```

MySQL functions: mysql_fetch_array() (1)

```
array mysql_fetch_array ( resource $result
                          [, int $result_type ] )
```

This function returns an array that corresponds to the fetched row and moves the internal data pointer ahead.

$result is the result resource that is being evaluated.
This result comes from a call to `mysql_query()`.

$result_type is the type of array that is to be fetched.
It's a constant and can take the following values:
`MYSQL_ASSOC`, `MYSQL_NUM`, `MYSQL_BOTH`.

It returns an array of strings that corresponds to the fetched row, or `FALSE` if there are no more rows.
The type of returned array depends on how *$result_type* is defined.

MySQL functions: mysql_fetch_array() (2)

```
<?php
  mysql_connect("localhost", "user", "password") or
      die("Could not connect:  " .  mysql_error());
  mysql_select_db("mydb") or
      die("Could not use $mydb:  " .  mysql_error());
  $result = mysql_query("SELECT id, name FROM mytable");
  while ($row = mysql_fetch_array($result, MYSQL_BOTH)) {
   printf ("ID: %s Name:  %s", $row[0], $row["name"]);
  }
  mysql_free_result($result);
?>
```

Exercise

1. Create a database inside MySQL.
2. Create a table called *users* to store name, surname, email and password of each user.
3. Create a table called *products* to store code, name, description of each product.
4. Insert two/three users in the table *users* and do the same for the *products* table.
5. Create a script *index.php* which prints a login form (email, password).
6. It must send the request to *login.php*.
7. The script *login.php* performs following actions:
   1. it checks the user's credentials,
   2. it sends the browser to *products.php* if the user's credentials are correct (use `header('Location: products.php');`),
   3. otherwise it sends the browser back to *index.php* and prints an error.
8. The script *products.php* shows the products saved in the database.

References

PHP Manual (English HTML version): http://www.php.net