

Version 2.1: Translated 2013-04-08  
Version 2.1: Uppdaterad 2013-03-25  
Version 2: Uppdaterad 2013-03-22  
Version 1.1: Uppdaterad 2013-03-07  
Version 1: Uppdaterad 2013-03-05

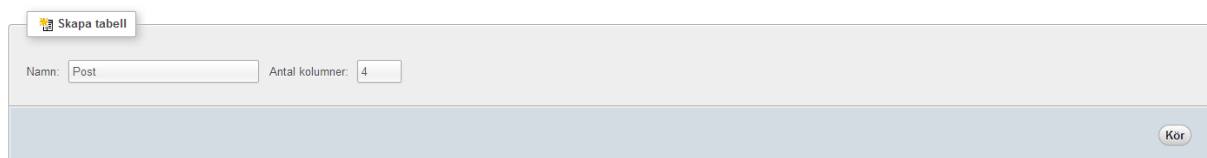
## Lab 4

We are going to continue with the webpage we have begun to make. This time we're going to create a **guestbook** and a **log in** function. In other words, we will **connect PHP to MySQL** in order to both get and send information to the database.

### 1

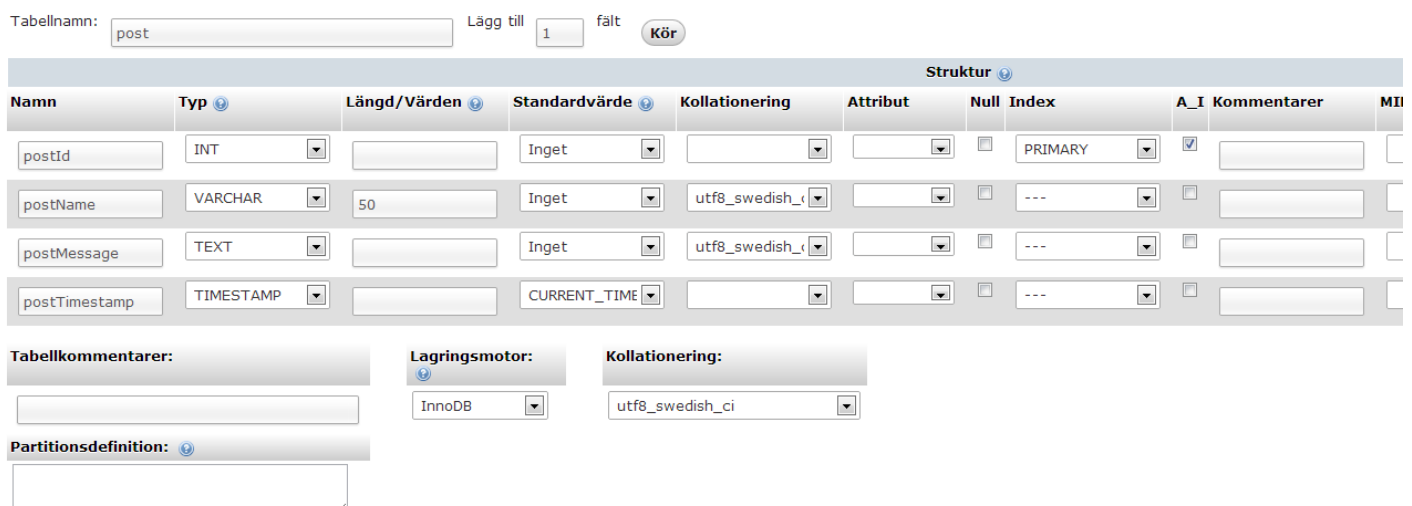
If you haven't already done so, go to **phpMyAdmin** (if you are working through **XAMPP** you can click the **Admin-button** on the **row for MySQL** in the control panel – if you are working on the school's server there is info on how you can access the course information on Blackboard). If you are using **localhost** you can **create a new database for this exercise**, but if you are using the **school's server** you **only have one database (username\_db)** for creating tables.

**Create a new table** in your database. Name it **post**. **Add 4 columns**.



Add the following attributes (columns):

post		
PK	<u>postId</u>	int
	postName	varchar(50)
	postMessage	text
	postTimestamp	timestamp



Don't forget to **select** the square below **A\_I** (stands for auto increment) for the **primary key** for it to be added automatically, as well as **choose PRIMARY** in the column "Index". **Standard value** for **postTimestamp** should be **CURRENT\_TIMESTAMP** so that it **automatically adds date and time** that the post was created.

**Click save** when you're finished. The table that will contain the posts in your **guestbook** is now done.

## 2

Create **gb.php**. Include **HTMLTemplate** and add what is needed to create the HTML-code for the page. Create a form with **text input** for **name** (**name="name"**), and a **text area** (**name="msg"**) for the message. Also add **a hidden text input field for preventing spam** in the same way that we did with the e-mail form. The form's **method should be post** and **action should be gb.php**.

```
<form action="gb.php" method="post">
  <label for="name">Name:</label>
  <input type="text" id="name" name="name" value="" />
  <input type="text" id="address" name="address" />
  <label for="msg">Message:</label>
  <textarea id="msg" name="msg"></textarea>
  <input type="submit" value="Submit" />
</form>
```

We will **reuse some things from the e-mail form**. Start by **adding an if-clause** that **checks that \$\_POST is not empty**.

**Get the values** from the **text input** fields and the **text area** to the variables **\$name**, **\$msg**, and **\$spamText**. **Check the spamtest** (the hidden input field), and **add appropriate feedback** if it isn't empty.

**Check that the user has written both name and message**, using an **if/else-clause**. **In the if clause**, make sure that **appropriate feedback** is printed. **Leave the else-clause empty** for now.

It should now be possible to open the page and a form should be shown. The user should be able to input information and send it. If the user hasn't filled out all the fields feedback should be shown. If all the information is correctly filled out, nothing happens right now, but we're going to fix that.

Don't forget to **prevent a XSS attack using htmlspecialchars()**!

```

1  <?php
2  /*-----
3  gb.php
4  Displays guestbook posts
5  and handles adding of new posts
6  -----*/
7
8  include_once("inc/HTMLTemplate.php");
9
10 if(!empty($_POST)) {
11
12     $name =      isset($_POST['name']) ? $_POST['name'] : '';
13     $msg =      isset($_POST['msg']) ? $_POST['msg'] : '';
14     $spamTest = isset($_POST['address']) ? $_POST['address'] : '';
15
16     if($spamTest != '') {
17         die("I think you're a robot. If you're not, go back and try again.");
18     }
19
20     if($name == '' || $msg == '') {
21         $feedback = "<p class='feedback-yellow'>Please fill out all fields.</p>";
22     } else {
23         //Add to DB
24     }
25 }
26
27 $name = htmlspecialchars($name);
28 $msg = htmlspecialchars($msg);
29
30 $content = <<<END
31
32     <div id="container">
33         {$feedback}
34         <form action="gb.php" method="post">
35             <label for="name">Name:</label>
36             <input type="text" id="name" name="name" value="{ $name}" />
37             <input type="text" id="address" name="address" />
38             <label for="msg">Message:</label>
39             <textarea id="msg" name="msg">{$msg}</textarea>
40             <input type="submit" value="Submit" />
41         </form>
42     </div><!-- container -->
43
44 END;
45
46 echo $header;
47 echo $content;
48 echo $footer;
49
50 ?>

```

### 3

Create a file that you name **Connstring.php** and save it in the inc-folder. You should **add all the information about the login to the MySQL database here**, so we can reuse it every time we need access to the database.

**NB!** We are going to use the **mysqli** object in PHP to handle the database connection. This is the object oriented way of doing this. There is, however, **a non-oop way**, which works slightly differently. If you google this you will notice that the variable handling the database is called **\$mysqli** when it's **non-oop** and **\$mysqli** when it's **oop**.

We will **begin by contacting the database**. Add the following in Connstring.php:

```
$mysqli = new mysqli('', '', '', ''); // (HOST, USER, PASSWORD, DATABASE)
```

The `mysqli` constructor that we call here requires 4 parameters: host, username, password and database. If you are using XAMPP it's the following information:

Host:	localhost
Username:	root
Password:	(nothing, leave the string empty)
Database:	(the name you have given your database)

If you are using the school's database it's the following information:

Host:	localhost
Username:	(your username)
Password:	(the password you have been given for your database – not your usual password)
Database:	username_db (change to your username)

Add your information as strings in the `mysqli`-constructor. It can e.g. look like this:

```
$mysqli = new mysqli('localhost', 'root', '', 'myDB');
```

Then add the following:

```
if (mysqli_connect_error()) {  
    echo "Connect failed: " . mysqli_connect_error() . "<br>";  
    exit();  
}  
$mysqli->set_charset("utf8");
```

This if clause prints feedback if something goes wrong when we try to contact the database. Finally we set charset to utf8.

```
1  <?php  
2  /*-----  
3  Connstring.php  
4  Include file for connecting to database  
5  -----*/  
6  
7  $mysqli = new mysqli('localhost', 'root', '', 'WEBMULTI03'); //(HOST, USER, PASSWORD, DATABASE)  
8  
9  if (mysqli_connect_error()) {  
10     echo "Connect failed: " . mysqli_connect_error() . "<br>";  
11     exit();  
12 }  
13 $mysqli->set_charset("utf8");  
14  
15  
16  ?>
```

## 4

Return to `gb.php`. Add an include for `Connstring.php` that we just created. It's appropriate to add it in the beginning of the file, just after you included `HTMLTemplate`. However, note that as soon as we include the file we will contact the database. There might thus already be error messages if something is not right.

Create a variables called `$tablePost`, with the value "post", i.e. the name of the table we created earlier.

In the empty else clause, after the if clause that checks if the user has added all their information, we are going to add the code needed for saving the new post to the database.

Start by adding the following two lines:

```
$name      = $mysqli->real_escape_string($name);  
$msg       = $mysqli->real_escape_string($msg);
```



This method (`real_escape_string()`) is used for preventing SQL-injektions. I.e. when someone enters SQL code in the text input and tries to change your database. Remember to use this every time you send any user defined information to the database!

Since there is a chance that the user has a name including e.g. `ääö`, or writes a message with `ääö`, we should use `utf8_encode()` on the strings that we are going to send to the database. Therefore, add the following:

```
$name      = utf8_encode($mysqli->real_escape_string($name));  
$msg       = utf8_encode($mysqli->real_escape_string($msg));
```

Note that we should use `utf8_decode()` later when we retrieve the value from the database in order for it to be presented correctly.

Now we are going to write our MySQL query. Create the variable `$query` and, using heredoc, assign it the following query (i.e. as a string value):

```
--  
-- Inserts new message into DB  
--  
INSERT INTO {$tablePost}(postName, postMessage)  
VALUES ('{$name}', '{$msg}');
```

The first three rows are just comments. Then it's a regular INSERT clause, but we use the `$tablePost` variable that we created, along with the `$name` and `$msg` that we retrieved with `$_POST` from the user input. As you remember the primary key `postId` is created automatically, so we don't have to

add that. `postTimestamp` is also created automatically since we have a default value. This is thus all we have to do.

We then use the `$query` variable to send the query to the database. This is done in the following way:

```
$res = $mysqli->query($query) or die("Could not query database" . $mysqli->errno .  
" : " . $mysqli->error); //Performs query
```

This is a kind of if clause. We are saying that `$res` equals `$mysqli->query($query)` if it is true, otherwise a `die()` is run, which prints out an error message. If everything goes well, the result of the query is saved in the `$res` variable.

You can now try to add a post to the database. Go to phpMyAdmin and make sure that a row had been added to the table.

The screenshot shows the phpMyAdmin interface. At the top, there are navigation tabs: Bläddra, Struktur, SQL, Sök, Lägg till, Exportera, and Importera. Below these, a green status bar indicates "Visar rader 0 - 4 (~5 totalt)" and "Frågan tog 0.0005 sek". The SQL query editor shows the query: `SELECT * FROM `post` LIMIT 0, 30`. Below the query editor, there are input fields for "Visa : Starta rad: 0", "Antal rader: 30", and "Rubriker var 100 rad". A dropdown menu for "Sortera efter nyckel:" is set to "Inget". Below this, a table of results is displayed with columns: `postId`, `postName`, `postMessage`, and `postTimestamp`. The first row contains the values: 1, Susanne, Test01, and 2013-02-26 10:32:41. A blue arrow points to the `postTimestamp` column header. At the bottom of the table, there are icons for "Redigera", "Kopiera", and "Radera".

	<code>postId</code>	<code>postName</code>	<code>postMessage</code>	<code>postTimestamp</code>
	1	Susanne	Test01	2013-02-26 10:32:41

```

29     } else {
30         //-----
31         //Prevents SQL injections
32         $name = utf8_encode($mysqli->real_escape_string($name));
33         $msg = utf8_encode($mysqli->real_escape_string($msg));
34
35         //-----
36         //SQL query
37         $query = <<<END
38         --
39         -- Inserts new message into DB
40         --
41         INSERT INTO {$tablePost} (postName, postMessage)
42         VALUES ('{$name}', '{$msg}');
43
44     END;
45
46     $mysqli->query($query) or die("Could not query database" . $mysqli->errno . " : " . $mysqli->error); //Performs query
47     $feedback = "<p class='feedback-green'>Your post has been added. Thanks!</p>";
48
49 }

```

## 5

We are going to retrieve the posts from the database and display them underneath the form on the guestbook page. It is suitable to add the query for this after the HTML-code for the form (i.e. outside of the if clause for the \$\_POST, since we want the posts to be shown even before the user writes a post of their own).

We don't have to retrieve any variables from the user this time, so we can go straight to the query. Create a variable called \$query and using heredoc assign the following query:

```

--
-- Gets all posts from DB
--
SELECT postName, postMessage, postTimestamp
FROM {$tablePost}
ORDER BY postTimestamp DESC;

```

This time we are using a regular SELECT clause for retrieving the columns postName, postMessage and postTimestamp from \$tablePost. I have entered that I want to organise the posts with the latest post first, but do as you please.

Retrieve the \$res variable in the same way as before.

We are now going to write a loop which goes through all the rows in the result and prints them. \$res obviously can contain everything from 0 to a very large amount of rows depending on how many posts there are, so the code has to be able to handle this. We are going to use a while loop for this.

```
while($row = $res->fetch_object()) {  
  
}
```

For each row that is being retrieved from the database a \$row variable is going to be created inside the whole loop. We can refer to the columns from the table like this:

```
$row->postName
```

Inside the while loop, add the HTML code for presenting a post. Add this code to \$content by writing .= (i.e. concatenation). Don't forget to use utf8\_decode() and htmlspecialchars() on the variables.

Finally, we have to, outside of the while loop, close \$res and \$mysqli since they still are open objects. Therefore, add the following:

```
$res->close();  
$mysqli->close();
```

If you open gb.php in the browser now all posts should be displayed below the form, and it should be possible to add posts which are displayed immediately after the page has been updated. In other words: we now have a functioning guestbook.



```

69 //-----
70 //SQL query
71 $query = <<<END
72 --
73 -- Gets all posts from DB
74 --
75 SELECT postName, postMessage, postTimestamp
76 FROM {$tablePost}
77 ORDER BY postTimestamp DESC;
78
79 END;
80
81 $res = $mysqli->query($query) or die("Could not query database" . $mysqli->errno . " : " . $mysqli->error); //Performs query
82
83 //Loops through results
84 while($row = $res->fetch_object()) {
85     $postName      = utf8_decode(htmlspecialchars($row->postName));
86     $postMessage    = utf8_decode(htmlspecialchars($row->postMessage));
87
88     $content .= <<<END
89     <div class="gb-post">
90     <p class="gb-name">Written by: {$postName}</p>
91     <p class="gb-msg">{$postMessage}</p>
92     <p class="gb-date">{$row->postTimestamp}</p>
93     </div>
94
95     END;
96 }
97
98 $content .= "</div><!-- container -->";
99
100 $res->close(); //Closes results
101 $mysqli->close(); //Closes DB connection
102
103 echo $header;
104 echo $content;
105 echo $footer;
106
107 ?>

```

## 6

There is one last thing that we should do before we finish the guestbook. As you can see, the **timestamp** information we retrieve from the database is showing the **entire date and time**, including seconds. This feels a little **superfluous**, so we are going to **reformat the date**.

In order to be able to **use the PHP function date()** we have to **reformat the timestamp that we get from MySQL to a so called PHP-timestamp**. We do this by **adding the following in the while loop (before creating the HTML code)**:

```
$date = strtotime($row->postTimestamp);
```

The **\$date** variable **now contains a PHP-adapted timestamp** that **we can use with date()**.

```
$date = date("d M Y H:i", $date);
```

I have chosen to format the date like this:

```
26 Feb 2013 11:01
```

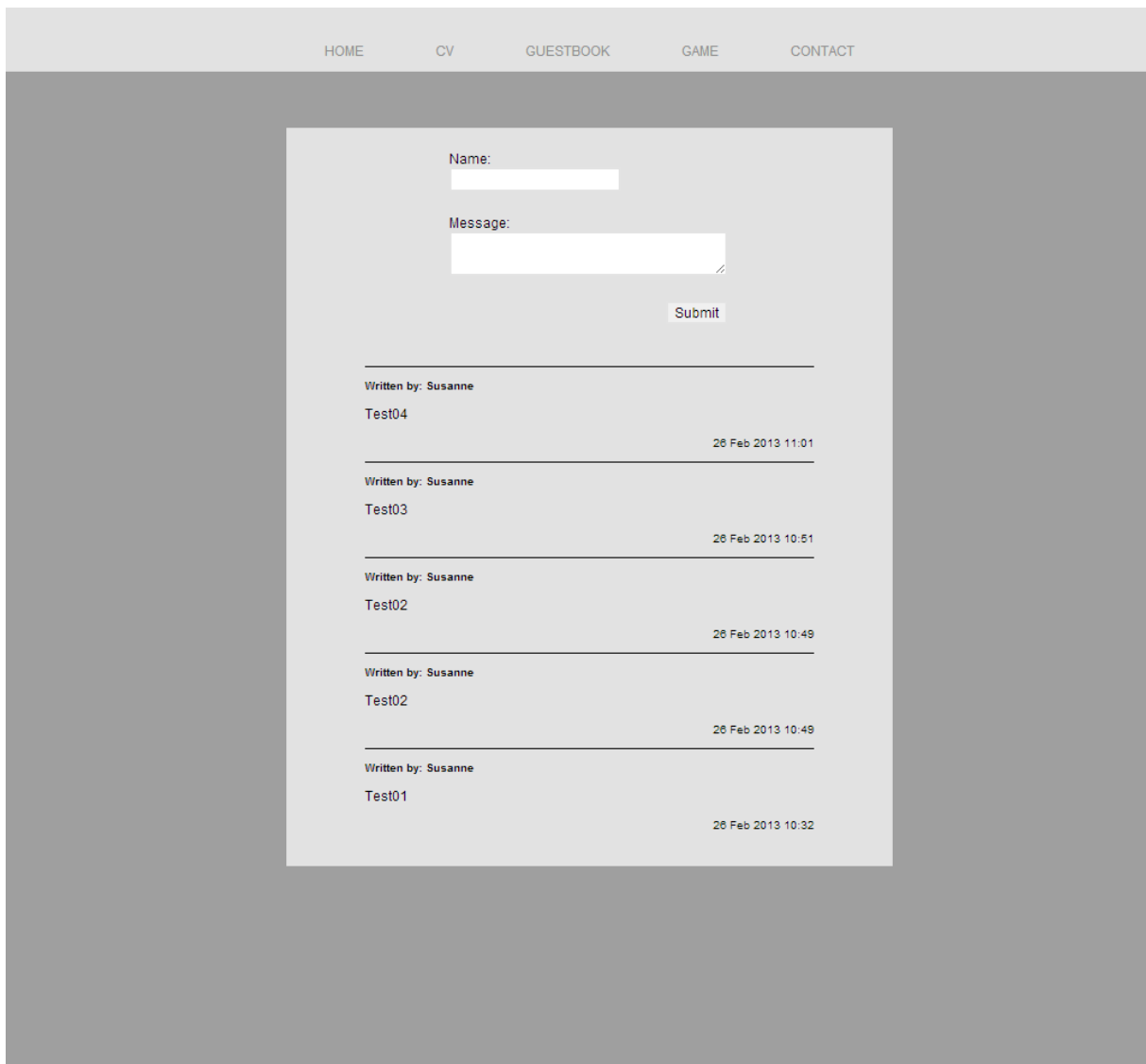
You can choose yourself how you want the date to be displayed. Read more about **date formatting** [here](#):

<http://php.net/manual/en/function.date.php>

Now `print $date` in the HTML code instead of `$row->postTimestamp`, and we will have a nicely formatted date. Now we're finished with the guestbook.

```
83 //Loops through results
84 while($row = $res->fetch_object()) {
85     $date = strtotime($row->postTimestamp);
86     $date = date("d M Y H:i", $date); //http://php.net/manual/en/function.date.php
87
88     $postName      = utf8_decode(htmlspecialchars($row->postName));
89     $postMessage    = utf8_decode(htmlspecialchars($row->postMessage));
90
91     $content .= <<<END
92     <div class="gb-post">
93         <p class="gb-name">Written by: {$postName}</p>
94         <p class="gb-msg">{$postMessage}</p>
95         <p class="gb-date">{$date}</p>
96     </div>
97
98 END;
99 }
```

My guestbook looks like this in the browser:



HOME CV GUESTBOOK GAME CONTACT

Name:

Message:

Submit

---

Written by: Susanne  
Test04  
26 Feb 2013 11:01

---

Written by: Susanne  
Test03  
26 Feb 2013 10:51

---

Written by: Susanne  
Test02  
26 Feb 2013 10:49

---

Written by: Susanne  
Test02  
26 Feb 2013 10:49

---

Written by: Susanne  
Test01  
26 Feb 2013 10:32

## 7

We're now going to create a login page that checks with the database if the user exists and if the password is correct. Start by creating `login.php`. Add a link to the page somewhere discreet. I added a link in the upper right corner that has the same color as the page header, what the changes color on hover. It was easy to add in HTMLTemplate, and will thus be available on all pages.



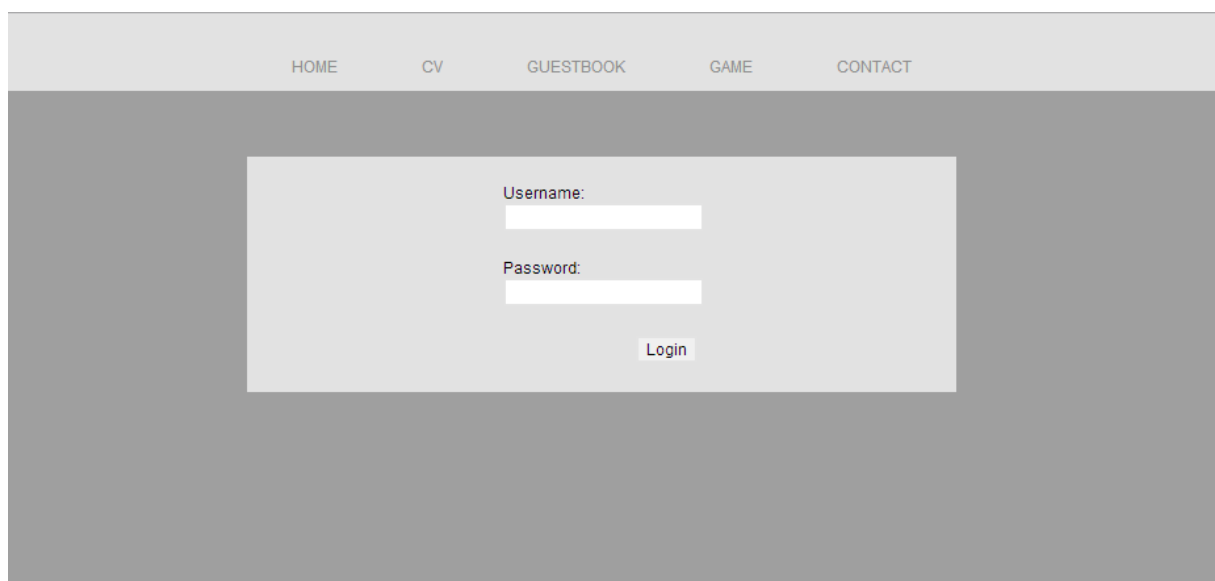
In `login.php`, include `HTMLTemplate` and in `$content` write the HTML code necessary for showing a form with two input fields and a submit button. Use `type="password"` for the input field that is meant to take the password so it isn't written out. The form's action should be `login.php`, and method should be `post`. Don't add the anti-spam field this time.

Don't forget to prevent a XSS-attack using `htmlspecialchars()`!

The HTML for my form:

```
<form action="login.php" method="post" id="login-form">
    <label for="username">Username:</label>
    <input type="text" id="username" name="username" value="" />
    <label for="password">Password:</label>
    <input type="password" id="password" name="password" value="" />
    <input type="submit" value="Login" />
</form>
```

My login page looks like this:



HOME CV GUESTBOOK GAME CONTACT

Username:

Password:

Login

## 8

We are going to create a table in phpMyAdmin that contains all admin users and their passwords. Create the following table:

admin		
PK	<u>adminId</u>	int
	adminName adminPassword	varchar(50) varchar(100)

Also add a post to the table so that we have an admin to test against when we're going to try to log in. You can add posts under "Add" in the table menu.

Kolumn	Typ	Funktion	Null	Värde
adminId	int(11)	<input type="text"/>		
adminName	varchar(50)	<input type="text"/>		testaren
adminPassword	varchar(100)	<input type="text"/>		choklad

Kör

Click Run to add the post.

## 9

We're going to reuse a lot of the guestbook's code. Start by using an if-clause to check that `$_POST` isn't empty, and in that case retrieve the variables `$username` and `$password` from the fields the user filled in.

Make sure that the user filled in something in both fields. If they didn't, give feedback. If they did, use `real_escape_string()` to prevent SQL-injections (on both variables).

Create the variable `$query` with the following content:

```
--
-- Gets username and password based on user input
--
SELECT adminName, adminPassword
FROM {$tablePost}
WHERE adminName = "{$username}";
```

Here we are **using a SELECT clause** to retrieve adminName and adminPassword for the user who has the username the user entered into the input field.

Run the query (copy the long \$res = \$mysqli->query... row from the guestbook). Don't forget to **include Connstring.php** at a suitable place.

```
13 if(!empty($_POST)) {
14     include_once("inc/Connstring.php");
15     $table = "admin";
16
17     $username = isset($_POST['username']) ? $_POST['username'] : '';
18     $password = isset($_POST['password']) ? $_POST['password'] : '';
19
20     if($username == '' || $password == '') {
21         $feedback = "<p class='feedback-yellow'>Please fill out all fields.</p>";
22     } else {
23         //-----
24         //Prevents SQL injections
25         $username = $mysqli->real_escape_string($username);
26         $password = $mysqli->real_escape_string($password);
27
28         //-----
29         //SQL query
30         $query = <<<END
31         --
32         -- Gets username and password based on user input
33         --
34         SELECT adminName, adminPassword
35         FROM {$table}
36         WHERE adminName = "{$username}";
37
38     END;
39
40
41     $res = $mysqli->query($query) or die("Could not query database" . $mysqli->errno . " : " . $mysqli->error); //Performs query
```

Now we will do some things we haven't done before. We are going to use `num_rows` to find out how many rows were returned from our query. If the query doesn't return any rows we know that the username doesn't exist, but if we return one (or more) rows, we'll know that it exists. Therefore, write an `if/else-clause` that uses

```
$res->num_rows
```

to check if one or more rows are returned. If there is no row returned, send feedback to the user in the `else clause`.

In the `if clause` (i.e. if one or more rows are returned), start by retrieving the row that was returned by writing:

```
$row = $res->fetch_object();
```

Then use another `if/else clause` to check if the password that the user entered is the same as `$row->adminPassword`. In the `else clause` (if it's not the same), print feedback to the user. If the password is the same the user should be logged in. We are, however, not going to write that code now, but you can just add the following:

```
die("Login success.");
```

This way we'll know that everything works, and we can add the code to handle the login later instead.

**Try to login.** It should be possible to login with the correct username and password, and if it's the wrong username or password appropriate feedback should be given.

**NB!** As you may have noticed, the code isn't adapted to work with the causality that there is more than one person with the same username. It's therefore important to add a control point that the username isn't already used when a new user is added. We are not going to create this functionality (it's easy to create based on what we're doing in these exercises however), but it's something to remember.

```

23     } else {
24         //-----
25         //Prevents SQL injections
26         $username = $mysqli->real_escape_string($username);
27         $password = $mysqli->real_escape_string($password);
28
29         //-----
30         //SQL query
31         $query = <<<END
32         --
33         -- Gets username and password based on user input
34         --
35         SELECT adminName, adminPassword
36         FROM {table}
37         WHERE adminName = "{$username}";
38
39     END;
40
41     $res = $mysqli->query($query) or die("Could not query database" . $mysqli->errno . " : " . $mysqli->error); //Performs query
42
43     if($res->num_rows == 1) {
44         $row = $res->fetch_object();
45         if($row->adminPassword == $password) {
46             die("login success");
47         } else {
48             $feedback = "<p class='feedback-red'>Password is incorrect.</p>";
49         }
50         $res->close();
51     } else {
52         $feedback = "<p class='feedback-red'>Username is incorrect.</p>";
53     }
54
55     $mysqli->close();
56
57 }

```

## 10

There is one last thing to do. As you have noticed, the password is saved as text in the database at the moment. If anyone accesses this information they will have access to all the passwords. We are therefore going to **encrypt the passwords**. We are going to use a built in function in PHP that's called **md5()**. It's also available in phpMyAdmin when you add posts to the table.

Go to **phpMyAdmin** and **open the admin table**. Click **Edit** by the post that already exists in the database. In the Function column there is **a drop down list that contains the function that should be used when you add posts to the database**. Find **MD5**.

adminPassword varchar(100)	MD5	choklad
----------------------------	-----	---------

**Click Run**. When you now check the content of the table there are going to be **a bunch of letters and numbers instead of the old password**.

There will, however, be a problem with the login now, because we can't enter the old password in the login form and get a true result from our comparing if clause. We are going to have to **use md5() on the password the user has entered** and **then compare** it to the encrypted password from the database.

**Add the following row** before you to the comparison **between \$row->adminPassword and \$password**:



```
$pswmd5 = md5($password);
```

Then compare, in the `if` clause, `$row->adminPassword` and `$pswmd5` instead. Now it should be possible to login as before.

```
43 if($res->num_rows == 1) {  
44     $pswmd5 = md5($password);  
45     $row = $res->fetch_object();  
46     if($row->adminPassword == $pswmd5) {  
47         die("login success");  
48     } else {  
49         $feedback = "<p class=\"feedback-red\">Password is incorrect.</p>";  
50     }  
51     $res->close();  
}
```

Now we have finished both the guestbook and the login function. We are going to keep working on there now though, to add more functionality.

## 11

We are going to keep building on the guestbook and login function that we started. If you want, you can save a backup of the files you have up to this point, because we are going to keep working on and changing them. We are going to add an admin function that allows whoever logged in to change and remove guestbook posts. We are also going to make it possible to comment posts.

Let's start with the comment function. The target is that the guestbook should look like this in the end:

[HOME](#)
[CV](#)
[GUESTBOOK](#)
[GAME](#)
[CONTACT](#)

Name:   
Message:

---

Written by: testaren  
test ååö igen  
Write a comment 27 Feb 2013 14:21

Written by: testaren  
ååååååååöööö

27 Feb 2013 14:26

---

Written by: testaren  
Test04 och test om edit fungerar  
Write a comment 26 Feb 2013 11:01

Written by: Susanne  
I'm commenting myself...

27 Feb 2013 11:13

Written by: suli  
Admin test.

27 Feb 2013 14:58

---

Written by: Susanne  
Test03  
Write a comment 26 Feb 2013 10:51

To start off, we have to **add** a new table in the database called **comment**. Therefore, create the following table:

comment		
PK	<u>commId</u>	int
	commName	varchar(50)
	commMessage	text
	commTimestamp	timestamp
	postId	int
	adminId	int

Note that **adminId** had to be able to be NULL.

Also **add an attribute** in the **post table** for **adminId** (we are going to **save adminId** if the user is logged in when they write a post, and then **add a marker in the guestbook** when the post has been written by an admin user).

post		
PK	<u>postId</u>	int
	postName postMessage postTimestamp <b>adminId</b>	varchar(50) text timestamp <b>int</b>

As you can tell, the attribute **postId in the comment table** is a **foreign key**, as is **adminId** in both these **tables**. Also here **adminId** has the be able to be **NULL**.

## 12

---

Start by creating **gb-comm.php**. Include **HTMLTemplate** and **Connstring**.

Write the code necessary for printing a **form** (the same as in the **guestbook**) for adding a **comment**. (**Don't forget to prevent XSS-attacks using htmlspecialchars()**)

We are going to **user GET to send information about which post should be commented**, and **then retrieve it from the database**. We are also **going to get any existing comments that belong to that post**. When the user **clicks the button** to save a comment, we're going to **add it to the comment table**, and **use the foreign key postId in order to know which post the comment belongs to**.

We are going to be reusing a lot of the code from **gb.php**, so you should recognize a lot of what we're going. Therefore, you should try yourself first before you check my solutions. **Start by declaring the following variables:**

```
$tableComment = "comment";  
$tablePost = "post";
```

**Get \$postId** from GET as we have done before.

**Check that \$postId isn't empty** – if it is, **send the user back to the guestbook**, **or give some sort of feedback**.

Write the code that is needed to **get the chosen post from the database** (don't forget to prevent SQL-injections). Reuse code you already have for this.

**Use num\_rows** (as we did with the login) to check if the post exists in the database. If it **doesn't**, give the user **feedback**. If it **does**, print the HTML code necessary to show the post.

Create a new query to get all the comments that belong to the chosen comment. You'll get the query here:

```
--  
-- Gets all comments for chosen post from DB  
--  
SELECT commName, commMessage, commTimestamp  
FROM {$tableComment}  
WHERE postId = {$postId}  
ORDER BY commTimestamp ASC;
```

It's a regular SELECT clause, but with a WHERE clause that specifies which postId that the comment should have.

Print the HTML code for the comments. Remember that the code should be able to handle both if there are no comments, and if there are many.

Don't forget to close \$res and \$mysqli.

If you now test the page and everything is correct, a form should be shown, followed by the post and the comments that belong to it. So far there are no comments, but if you want to add some using phpMyAdmin in order to test the page, that's fine.

```

1  <?php
2  /*-----
3  gb-comm.php
4  Shows chosen GB-post with comment
5  and allows adding of comments
6  -----*/
7  include_once("inc/HTMLTemplate.php");
8  include_once("inc/Connstring.php");
9  $tableComment = "comment";
10 $tablePost = "post";
11
12 $feedback = "";
13 $name = "";
14 $msg = "";
15
16 $postId = isset($_GET['id']) ? $_GET['id'] : '';
17
18 //Checks that post id has been entered
19 if($postId == '') {
20     header("Location: gb.php");
21     exit();
22 }
23
24 //-----
25 //Prevents SQL injections
26 $postId = $mysqli->real_escape_string($postId);
27
28 //-----
29 //SQL query
30 $query = <<<END
31 --
32 -- Gets chosen post from DB
33 --
34 SELECT postName, postMessage, postTimestamp
35 FROM {$tablePost}
36 WHERE postId = {$postId};
37
38 END;
39
40 $res = $mysqli->query($query) or die("Could not query database" . $mysqli->errno . " : " . $mysqli->error); //Perfor
41
42 //Checks that post exists
43 if($res->num_rows < 1) {
44     $content = <<<END
45         <div id="container">
46             <p>The post you chose cannot be found. Please try again.</p>
47             <p><a href="gb.php">Guestbook</a></p>
48         </div><!-- container -->
49     END;
50
51 } else {
52
53     if(!empty($_POST)) {
54
55         //Add new comment to DB
56
57     }
58
59     $row = $res->fetch_object(); //Gets chosen post from DB
60
61     //Formats date
62     $date = strtotime($row->postTimestamp);
63     $date = date("d M Y H:i", $date); //http://php.net/manual/en/function.date.php
64
65     $postName = utf8_decode(htmlspecialchars($row->postName));
66     $postMessage = utf8_decode(htmlspecialchars($row->postMessage));
67
68     $postHTML = <<<END
69
70         <h3>Write a comment to:</h3>
71         <div class="gb-post">
72             <p class="gb-name">Written by: {$postName}</p>
73             <p class="gb-msg">{$postMessage}</p>
74             <p class="gb-date">{$date}</p>
75         </div>
76
77     END;

```

```

78
79 $name = htmlspecialchars($name);
80 $msg = htmlspecialchars($msg);
81
82 $content = <<<END
83     <div id="breadcrumbs">
84         <p><a href="gb.php">Guestbook</a> &gt; Comment</p>
85     </div><!-- breadcrumbs -->
86
87     <div id="container">
88         { $feedback }
89         <form action="gb-comm.php?id={ $postId }" method="post">
90             <label for="name">Name:</label>
91             <input type="text" id="name" name="name" value="{ $name }" />
92             <input type="text" id="address" name="address" />
93             <label for="msg">Message:</label>
94             <textarea id="msg" name="msg">{ $msg }</textarea>
95             <input type="submit" value="Submit" />
96         </form>
97
98         { $postHTML }
99     END;

```

```

100
101 //-----
102 //SQL query
103 $query = <<<END
104 --
105 -- Gets all comments for chosen post from DB
106 --
107 SELECT commName, commMessage, commTimestamp
108 FROM { $tableComment }
109 WHERE postId = { $postId }
110 ORDER BY commTimestamp ASC;
111
112 END;
113
114 $res = $mysqli->query($query) or die("Could not query database" . $mysqli->errno . " : " . $mysqli->error); //Performs qu
115
116 //Loops through results
117 while($row = $res->fetch_object()) {
118     $date = strtotime($row->commTimestamp);
119     $date = date("d M Y H:i", $date); //http://php.net/manual/en/function.date.php
120
121     $commName = utf8_decode(htmlspecialchars($row->commName));
122     $commMessage = utf8_decode(htmlspecialchars($row->commMessage));
123
124     $content .= <<<END
125         <div class="gb-comm">
126             <p class="gb-name">Written by: { $commName }</p>
127             <p class="gb-msg">{ $commMessage }</p>
128             <p class="gb-date">{ $date }</p>
129         </div>
130     END;
131 }
132
133 $content .= "</div><!-- container -->";
134
135 $res->close(); //Closes results
136 $mysqli->close(); //Closes DB connection
137
138 }
139
140 echo $header;
141 echo $content;
142 echo $footer;
143
144 ?>

```

In order to be able to add a new comment we have to use more or less the same code as for creating a post. We only have to make sure to add the postId in the comment table so that we save which post the comment belongs to.

In the same else clause where you get the content from the chosen post you can add an if() that checks if \$\_POST is empty, and if it isn't, add the new comment. It's just the same as we've done before. Don't forget to check if \$name or \$msg are empty, as well as prevent SQL injections. The query you can use is:

```
--
-- Inserts new comment into DB
--
INSERT INTO {$tableComment}(commName, commMessage, postId)
VALUES ('{$name}', '{$msg}', {$postId});
```

Note that the placement of this if() is important. If you are reusing variable names such as \$query e.g., it can't be in a place where it overwrites the old query before we've had a chance to use it. Make sure to test the functionality properly.

Also add feedback when the comment has been added.

If everything is correct it should now be possible to add comments, and they should immediately show up in the list underneath the form. Add a breadcrumb to the page so it's obvious where the user is.

```
53 if(!empty($_POST)) {
54
55     $name = isset($_POST['name']) ? $_POST['name'] : '';
56     $msg = isset($_POST['msg']) ? $_POST['msg'] : '';
57     $spamTest = isset($_POST['address']) ? $_POST['address'] : '';
58
59     if($spamTest != '') {
60         die("I think you're a robot. If you're not, go back and try again.");
61     }
62
63     if($name == '' || $msg == '') {
64         $feedback = "<p class='feedback-yellow'>Please fill out all fields.</p>";
65     } else {
66         //-----
67         //Prevents SQL injections
68         $name = utf8_encode($mysqli->real_escape_string($name));
69         $msg = utf8_encode($mysqli->real_escape_string($msg));
70
71         //-----
72         //SQL query
73         $query = <<<END
74         --
75         -- Inserts new comment into DB
76         --
77         INSERT INTO {$tableComment}(commName, commMessage, postId)
78         VALUES ('{$name}', '{$msg}', {$postId});
79
80     END;
81
82     $mysqli->query($query) or die("Could not query database" . $mysqli->errno . " : " . $mysqli->error); //Performs q
83     $feedback = "<p class='feedback-green'>Your comment has been added. Thanks!</p>";
84
85     $msg = "";
86     $name = "";
87
88 }
89
90 }
```

## 14

---

Open `gb.php`. Add the code necessary to retrieve the comments that belong to each post. **Tip:** run a query to get comments in the while loop you already have for retrieving posts from the database. Try to solve this yourself. My solution is on the next page if you need it.



```

81 //Loops through results
82 while($row = $res->fetch_object()) {
83     $date = strtotime($row->postTimestamp);
84     $date = date("d M Y H:i", $date); //http://php.net/manual/en/function.date.php
85
86     $postName      = utf8_decode(htmlspecialchars($row->postName));
87     $postMessage    = utf8_decode(htmlspecialchars($row->postMessage));
88
89     $content .= <<<END
90     <div class="gb-post">
91         <p class="gb-name">Written by: {$postName}</p>
92         <p class="gb-msg">{$postMessage}</p>
93         <p><span class="gb-comment"><a href="gb-comm.php?id={$row->postId}">Write a comment</a></span><span class="gb-date">{$date}</span></p>
94     </div>
95
96 END;
97
98 //Query for comments
99 $query = <<<END
100 --
101 -- Gets all comments for current post from DB
102 --
103 SELECT commName, commMessage, commTimestamp
104 FROM {$tableComment}
105 WHERE postId = {$row->postId}
106 ORDER BY commTimestamp ASC;
107
108 END;
109
110 $res2 = $mysqli->query($query) or die("Could not query database" . $mysqli->errno . " : " . $mysqli->error); //Performs query
111
112 //Loops through results for comments
113 while($row2 = $res2->fetch_object()) {
114     $date = strtotime($row2->commTimestamp);
115     $date = date("d M Y H:i", $date); //http://php.net/manual/en/function.date.php
116
117     $commName      = utf8_decode(htmlspecialchars($row2->commName));
118     $commMessage    = utf8_decode(htmlspecialchars($row2->commMessage));
119
120     $content .= <<<END
121     <div class="gb-comm">
122         <p class="gb-name">Written by: {$commName}</p>
123         <p class="gb-msg">{$commMessage}</p>
124         <p class="gb-date">{$date}</p>
125     </div>
126
127 END;
128 }
129 }

```

## 15

Now we are going to add the actions that occur when the user logs in. Open `login.php`. As you recall, we only added a `die()` when the user managed to login. What we're going to do now is to start a session which remembers that the user is logged in. A **session** is a special variable in PHP that remembers things for as long as the browser is open. If the user closes the browser they will be logged out automatically, but as long as the browser is open the session will remember the information even when we reload pages.

The session variable is written like this:

```
$_SESSION
```

It works as an associative array, which means that we can add variables by creating our own names for the keys:

```
$_SESSION["whateverYouWant"]
```

If we e.g. want to save the username from our variable `$username` we can do the following:

```
$_SESSION["username"] = $username;
```

In order to be able to use a session we first have to start it:

```
session_start();
```

This **always has to be done**, even when there is a session with saved variables. If we don't start it we can't access `$_SESSION`.

In the **if clause** in `login.php` that compares the entered password with that which is saved in the database, **remove the line with `die()`** that we added earlier.

**Instead add the following:**

```
session_start();
session_regenerate_id();

$_SESSION["username"] =      $username;
$_SESSION["userId"] =      $row->adminId;

header("Location: index.php");
```

What we're doing here is to **start the session**, and then we **generate a new ID**. This is necessary if there already exists an ID, i.e. if someone is already logged in but is trying to log in again.

Then we **save the value of `$username`** and also the **`adminId`** the user has. You may possibly have to **add `adminId` in the query that is performed just before this if clause**, if you don't already have it there.

Finally, we **use a `header()` to redirect the user to the start page**.

```

40     if($res->num_rows == 1) {
41         $pswmd5 = md5($password);
42         $row = $res->fetch_object();
43         if($row->adminPassword == $pswmd5) {
44             session_start();
45             session_regenerate_id();
46
47             $_SESSION["username"] = $username;
48             $_SESSION["userId"] = $row->adminId;
49
50             header("Location: {$redirect}.php");
51
52         } else {
53             $feedback = "<p class=\"feedback-red\">Password is incorrect.</p>";
54         }
55         $res->close();
56     } else {
57         $feedback = "<p class=\"feedback-red\">Username is incorrect.</p>";
58     }

```

## 16

Now it's possible to log in, but so far there is no difference to the user who is logged in. Open `HTMLTemplate.php` from the inc folder. We are going to adapt the menu a little depending on whether the user is logged in or not. Start by, at the top, starting the session.

**NB!** Since we are starting the session in `HTMLTemplate`, and always include `HTMLTemplate` in all pages that have HTML code, we won't have to start the session on any other page. Since we're adding it here we thus don't have to add it anywhere else.

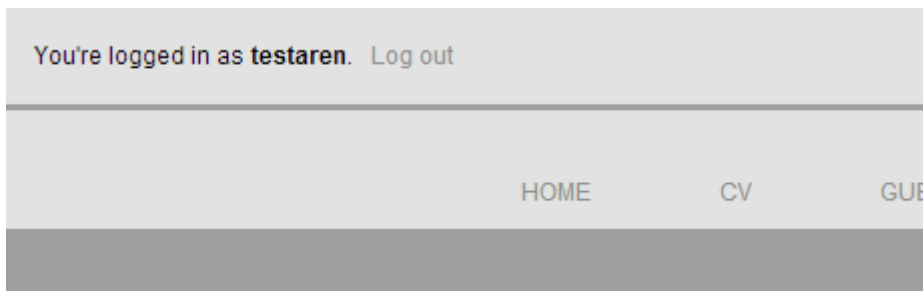
We are going to create a variable called `$adminHTML`, that will contain the code that will only be shown when the user is logged in. I'm going to make it simple for myself and add a row over the menu where I print the username of the person who is logged in, along with a log out link. We are soon going to create that page, so you can already link to `logout.php`.

This row I only want to show when the user is logged in, so I will write an if clause that uses `isset()` and checks if `$_SESSION["username"]` exists. If it exists I will add the HTML code that is needed to show the row in `$adminHTML`, otherwise I will leave `$adminHTML` empty.

Finally, I will add `$adminHTML` in `$header`, right above the code for the menu.

If I reload the page I will now see an extra row above the menu that shows that I am logged in. This row will exist on all pages, since it's placed in `HTMLTemplate`.

My solution looks like this in the browser:



The code:

```
1  <?php
2  /*-----
3  HTMLTemplate.php
4  Contains HTML code that is the same
5  over several pages
6  -----*/
7
8  session_start();
9
10 $adminHTML = "";
11 if(isset($_SESSION["username"])) {
12     $adminHTML = <<<END
13     <div id="admin-head">
14         <p>You're logged in as <strong>{$_SESSION["username"]}</strong>. &nbsp; <a href="logout.php">Log out</a><p>
15     </div><!-- admin-head -->
16 END;
17 }
18
19 $header = <<<END
20 <!DOCTYPE HTML>
21 <html>
22     <head>
23         <title></title>
24         <link rel="stylesheet" type="text/css" href="css/style.css" />
25     </head>
26     <body>
27         <div id="header">
28             { $adminHTML }
29             <div id="admin-login">
30                 <a href="login.php">Admin</a>
31             </div><!-- admin-login -->
32             <ul id="menu">
33                 <li><a href="index.php">Home</a></li>
34                 <li><a href="cv.php">CV</a></li>
35                 <li><a href="gb.php">Guestbook</a></li>
36                 <li><a href="#">Game</a></li>
37                 <li><a href="contact.php">Contact</a></li>
38             </ul>
39         </div><!-- header -->
40
41         <div id="content">
42
43 END;
```

## 17

Create **logout.php**. We are going to add the code for logging out. We **don't have to include HTMLTemplate here**, because we're not going to show any HTML. However, that means that we will have to **add session\_start()**. Begin by doing that.

**Then add the following:**

```
$_SESSION = array();

session_unset();
session_destroy();

header("Location: index.php");
```

First we **reset** the `$_SESSION` variable by saying that it is a new, empty array. Then we technically **reset** it again by doing `session_unset()` (it's best to do things properly, right?). Then we **"destroy"**, or **"close"** the session with `session_destroy()`. Finally, we **use `header()` to send the user back to the start page.**

```
1  <?php
2  /*-----
3   logout.php
4   Handles logging out
5   -----*/
6   session_start();
7
8   $_SESSION = array();
9
10  session_unset();
11  session_destroy();
12
13  header("Location: index.php");
14  ?>
```

## 18

As you have noticed, we send the user back to the start page, both on login and log out. Best would have been if the user were **sent back to the page they originated from**, i.e. if I'm on the résumé page and click Log out, I want to return to the résumé when I've been logged out. Try to solve this by using a **GET variable** and the following:



```
basename($_SERVER["PHP_SELF"], ".php");
```

This code will get the URL to the current page, and cuts everything except the file name from the string (the file type is cut as well). A tip is to use this e.g. in HTMLTemplate... I won't be giving you my solution to this, so you have to solve it on your own.

## 19

In the guestbook we are going to give all the posts created by an admin an adminId that we'll save in the database. Thus, we have to change some of the code we have. Open gb.php and scroll down to the query that adds a new post to the database.

Create a variable, \$adminId, that you give the value from \$\_SESSION["userId"] if it exists, and otherwise gives it the value "NULL" (simply a string with the word NULL in capital letters).

Add adminId in the INSERT clause in the query. Now when you add a post to the database it will check if you're logged in, and if you are an adminId will be added. If you're not logged in the value NULL will be added.

Test this, and add a few posts while you're logged in.

In order to make it easier for the user, add the following row just before the \$content that prints the form:

```
$name = isset($_SESSION["username"]) ? $_SESSION["username"] : $name;
```

Make sure that value="{ \$name}" in the input field that is meant to contain the name.

Now the username will already be added to the field when the user goes to the guestbook. This way, they don't have to enter it each time.

```

25 //-----
26 //Prevents SQL injections
27 $name = utf8_encode($mysqli->real_escape_string($name));
28 $msg = utf8_encode($mysqli->real_escape_string($msg));
29
30 $adminId = isset($_SESSION["userId"]) ? $_SESSION["userId"] : "NULL" ;
31
32 //-----
33 //SQL query
34 $query = <<<END
35 --
36 -- Inserts new message into DB
37 --
38 INSERT INTO {$tablePost}(postName, postMessage, adminId)
39 VALUES ('{$name}', '{$msg}', {$adminId});
40
41 END;
42
43 $res = $mysqli->query($query) or die("Could not query database" . $mysqli->errno . " : " . $mysqli->error); //Performs query

```

```

52 $name = isset($_SESSION["username"]) ? $_SESSION["username"] : $name;
53
54 $name = htmlspecialchars($name);
55 $msg = htmlspecialchars($msg);
56
57 $content = <<<END
58 <div id="container">
59     {$feedback}
60     <form action="gb.php" method="post">
61         <label for="name">Name:</label>
62         <input type="text" id="name" name="name" value="{ $name}" />
63         <input type="text" id="address" name="address" />
64         <label for="msg">Message:</label>
65         <textarea id="msg" name="msg">{$msg}</textarea>
66         <input type="submit" value="Submit" />
67     </form>
68
69 END;

```

## 20

Make the same adaptations for `gb-comm.php` regarding adding adminId in the comment table, and printing the username in the input field. Test to make sure it's working as it should.

## 21

We are going to give all posts written by an admin a little marker. Open `gb.php`. I have an image with a label that says admin, but you can choose whatever you like (e.g. another background color). I will att the image as a background using CSS, för all posts with the class admin.



Using `is_null()`, check if `$rox->adminId` is null. If it isn't, add admin as a class in the div that contains the post.

Do the same with the comments, and in `gb-comm.php`.

This is what my solutions looks like in the browser:



The code might look like this:

```
90     $adminClass = (!is_null($row->adminId)) ? " admin" : "";
91
92     $content .= <<<END
93     <div class="gb-post{$adminClass}">
94         <p class="gb-name">Written by: {$postName}</p>
95         <p class="gb-msg">{$postMessage}</p>
96         <p><span class="gb-comment"><a href="gb-comm.php?id={$row->postId}">Write a comment</a></span><spa
97     </div>
98
99 END;
```

## 22

We are going to make it possible to remove comments. Create `gb-delete.php`. Include `HTMLTemplate`.

Before we add functionality to remove a post we are going to make sure that the person opening the page really is logged in – we don't want that anyone just adds the URL and removes the posts when they don't have the right to do to. Therefore, start by checking that `$_SESSION["username"]` exists, and if it doesn't, redirect the user to index using `header()`. Also add an `exit()`.



Include Connstring() and create the variables \$tablePost and \$tableComment with the table names.

We are going to use the same file for removing both posts and comments. We are going to use GET to send the information about which post or comment is to be removed. We are going to use pid for postId and cid for commentId. Thus, get the values from these to the variables \$postId and \$commentId.

Check to make sure both variables aren't empty (i.e. there is no ID). If they are, print feedback, and a link back to the guestbook.

In the else clause (i.e. if the variables aren't empty) you can add a short text asking the user to confirm that they really want to remove the chosen post or comment. Add a link (with the text "yes") to gb-delete.php, including the pid or cid that is given in the GET, as well as another GET variables you name s and give the value y. As you may have guessed, we are going to use these variables to see if the user has answered yes.

It's also good to add a breadcrumb so it's easy to go back to the guestbook, alternatively a link with the text "no" leading back to the guestbook.

```

1  <?php
2  /*-----
3  gb-delete.php
4  Deletes chosen post or comment from DB
5  -----*/
6
7  include_once("inc/HTMLTemplate.php");
8
9  if(!isset($_SESSION["username"])) {
10     header("Location: index.php");
11     exit();
12 }
13
14 include_once("inc/Connstring.php");
15
16 $tablePost = "post";
17 $tableComment = "comment";
18
19 $postId = isset($_GET['pid']) ? $_GET['pid'] : '';
20 $commId = isset($_GET['cid']) ? $_GET['cid'] : '';
21
22 if($postId == '' && $commId == '') {
23     $content = <<<END
24         <div id="breadcrumbs">
25             <p><a href="gb.php">Guestbook</a> &gt; Delete</p>
26         </div><!-- breadcrumbs -->
27         <div id="container">
28             <p>No post or comment has been chosen. Please try again.</p>
29         </div><!-- container -->
30     END;
31
32 } else {
33     $type = ($postId != '') ? "post" : "comment" ;
34
35     $content = <<<END
36         <div id="breadcrumbs">
37             <p><a href="gb.php">Guestbook</a> &gt; Delete</p>
38         </div><!-- breadcrumbs -->
39         <div id="container">
40             <p>Are you sure you want to remove the chosen {$type}?</p>
41             <p><a href="gb-delete.php?pid={$postId}&cid={$commId}&s=y">Yes</a></p>
42         </div><!-- container -->
43     END;
44 }
45
46 echo $header;
47 echo $content;
48 echo $footer;
49
50
51 ?>

```

## 23

We are now going to add the code that removes the post or comment if the user clicks "yes".

Before the else clause, but after the if clause that checks if the variables cid and pid are empty, add an else if clause that checks if s has a value. Don't forget to get the value of s to the variable \$s earlier in the code.

In the else if clause, check if it's cid or pid that has been entered (e.g. by checking if they are != ""). Depending on the result we are going to create different queries. For removing posts we are going to use:

```
--  
-- Deletes chosen post  
--  
DELETE FROM {$tablePost}  
WHERE postId = {$postId};
```



And for removing comments we are going to use:

```
--  
-- Deletes chosen comment  
--  
DELETE FROM {$tableComment}  
WHERE commId = {$commId};
```

Give the value to \$query using if clauses. The rest of the code is the same no matter if we remove a post or a comment, so we can leave the if clauses we had for creating our queries. We will need to (1) perform the query, (2) using \$mysqli->affected\_rows check if a row has been affected, and in that case give feedback (if a row has been affected the delete was successful, if not it wasn't), and (3) close \$mysqli. Do this.

```

37 } else if ($s != '') {
38     $query = "";
39
40     if($postId == "y") {
41         $postId = $mysqli->real_escape_string($postId);
42         $type = "post";
43
44         //-----
45         //SQL query
46         $query = <<<END
47         --
48         -- Deletes chosen post
49         --
50         DELETE FROM {$tablePost}
51         WHERE postId = {$postId};
52
53     END;
54
55     } else if ($commId != '') {
56         $commId = $mysqli->real_escape_string($commId);
57         $type = "comment";
58
59         //-----
60         //SQL query
61         $query = <<<END
62         --
63         -- Deletes chosen comment
64         --
65         DELETE FROM {$tableComment}
66         WHERE commId = {$commId};
67
68     END;
69 }

```

```

70
71 $mysqli->query($query) or die("Could not query database" . $mysqli->errno . " : " . $mysqli->error); //Performs
72
73 if($mysqli->affected_rows >= 1) {
74     $feedback = "The {$type} has been removed.";
75 } else {
76     $feedback = "Something went wrong and the {$type} was not removed.";
77 }
78
79 $mysqli->close();
80
81 $content = <<<END
82     <div id="breadcrumbs">
83         <p><a href="gb.php">Guestbook</a> &gt; Delete</p>
84     </div><!-- breadcrumbs -->
85     <div id="container">
86         <p>{$feedback}</p>
87         <p><a href="gb.php">Back to guestbook</a></p>
88     </div><!-- container -->
89 END;
90 } else {

```

## 24

You can test gb-delete.php by manually adding a pid or cid in the URL field, but we are going to add a link to the page from gb.php. The link should of course only be shown if the user is logged in.

Find the code that prints the HTML code for the posts. Using an if clause, check that the user is logged in (as we've done before), and in that case print an extra row with links to the delete page and to gb-edit.php (which we will soon create). Add the posts id to the get variable pid in the link URL.

Do the same with the comments, but of course with the comment's cid value this time.

Now you can **try to remove** comments and posts **without having to enter the address manually**. Also **make sure** to test that the links are **only visible when you're logged in**.

My solution looks like this in the browser:



The code can e.g. look like this:

```
89     $adminRow = "";
90     $adminClass = (!is_null($row->adminId)) ? " admin" : "";
91
92     if(isset($_SESSION["username"])) {
93         $adminRow = <<<END
94         <p class="gb-admin-row"><a href="gb-edit.php?pid={$row->postId}">Edit</a> &middot; <a href="gb-delete.php?pid={$row->postId}">Delete</a></p>
95     END;
96     }
97
98     $content .= <<<END
99     <div class="gb-post{$adminClass}">
100         <p class="gb-name">Written by: {$postName}</p>
101         <p class="gb-msg">{$postMessage}</p>
102         <p><span class="gb-comment"><a href="gb-comm.php?id={$row->postId}">Write a comment</a></span><span class="gb-date">{$date}</span></p>
103         {$adminRow}
104     </div>
105
106 END;
```

## 25

Finally we're going to create **gb-edit.php** that enables us to change the posts.

**Initially** (after including HTMLTemplate) **check** that the user is **logged in**, and **if not redirect them to the start page or give some other kind of feedback**.

**Include connstring** and **create** the variables **\$tablePost** and **\$tableComment**. **Get \$postId** från the GET variables **pid** and **\$commId** from the GET variable **cid**. As you can see there is a lot of code we're reusing, as always.

**Start** by **retrieving the chosen post or comment from the database**. You will need to create different queries depending on whether a pid or cid is chosen (as in gb-delete). Don't forget to give **feedback if no post of comment can be found**.

**Print a form** (you can copy the form from the guestbook) that **includes the name and message** that is in the database for the chosen post/comment.

If you **test** the page **now** it should **contain a form with the post or comment you choose from the guestbook** (we **added the link to the page earlier**, remember?).

```

1  <?php
2  /*-----
3  gb-edit.php
4  Allows editing of chosen post or comment
5  -----*/
6
7  include_once("inc/HTMLTemplate.php");
8
9  if(!isset($_SESSION["username"])) {
10     header("Location: index.php");
11     exit();
12 }
13
14 include_once("inc/Connstring.php");
15 $tablePost = "post";
16 $tableComment = "comment";
17
18 $feedback = "";
19 $name = "";
20 $msg = "";
21
22 $postId = isset($_GET['pid']) ? $_GET['pid'] : '';
23 $commId = isset($_GET['cid']) ? $_GET['cid'] : '';
24
25 $type = ($postId != '') ? "post" : "comment" ;
26
27 if(!empty($_POST)) {
28     //Edit post or comment in DB
29
30 } else {
31
32
33     if($type == "post") {
34         $query = <<<END
35         --
36         -- Gets chosen post from DB
37         --
38         SELECT postId, postName, postMessage, postTimestamp, adminId
39         FROM {$tablePost}
40         WHERE postId = {$postId};
41     END;
42
43     } else {
44         $query = <<<END
45         --
46         -- Gets chosen comment from DB
47         --
48         SELECT commId, commName, commMessage, commTimestamp, adminId
49         FROM {$tableComment}
50         WHERE commId = {$commId};
51     END;
52
53 }
54
55 $res = $mysqli->query($query) or die("Could not query database" . $mysqli->errno . " : " . $mysqli->error); //Perf
56
57 if($res->num_rows < 1) {
58     $content = <<<END
59     <div id="container">
60         <p>The {$type} you chose cannot be found. Please try again.</p>
61         <p><a href="gb.php">Guestbook</a></p>
62     </div><!-- container -->
63 END;
64
65 } else {
66     $row = $res->fetch_object(); //Gets result from DB
67
68     $name = ($type == "post") ? $row->postName : $row->commName;
69     $msg = ($type == "post") ? $row->postMessage : $row->commMessage;
70
71     $name = utf8_decode($name);
72     $msg = utf8_decode($msg);
73
74     $content = getFormHTML($type, $postId, $commId, $name, $msg, $feedback);
75 }
76

```

```

77
78 echo $header;
79 echo $content;
80 echo $footer;
81
82 function getFormHTML($type, $postId, $commId, $name, $msg, $feedback) {
83     $name = htmlspecialchars($name);
84     $msg = htmlspecialchars($msg);
85
86     return <<<END
87         <div id="breadcrumbs">
88             <p><a href="gb.php">Guestbook</a> &gt; Edit</p>
89         </div><!-- breadcrumbs -->
90
91         <div id="container">
92             <h2>Edit the chosen {$type}</h2>
93             {$feedback}
94             <form action="gb-edit.php?pid={$postId}&cid={$commId}" method="post">
95                 <label for="name">Name:</label>
96                 <input type="text" id="name" name="name" value="{ $name}" />
97                 <label for="msg">Message:</label>
98                 <textarea id="msg" name="msg">{$msg}</textarea>
99                 <input type="submit" value="Save changes" />
100             </form>
101         </div><!-- container -->
102
103     END;
104 }
105
106 ?>

```

## 26

We are going to finish by creating the code that changes the post or comment when the user submits the form. Start by checking if `$_POST` is empty or not. If it isn't, get `$name` and `$msg` from the form, check that they've been filled in (if not, give appropriate feedback), and prevent SQL injections.

Depending on whether it's a post or comment that is being changes we are going to need different queries again. To change a post we can use:


```

--
-- Changes chosen post in DB
--
UPDATE {$stablePost}
SET postName = '{$name}', postMessage = '{$msg}'
WHERE postId = {$postId};

```

And to change a comment we can use:--

```
-- Changes chosen comment in DB
--
UPDATE {$tableComment}
SET commName = '{$name}', commMessage = '{$msg}'
WHERE commId = {$commId};
```

Resten av koden är densamma oavsett om det är ett inlägg eller en kommentar. 

Perform the query. Use `$mysqli->affected_rows` to check that we have changed something in the database, and print appropriate feedback. Close `$mysqli`.

Test the page. If everything works you will have a complete functionality for writing in and commenting the guestbook, logging in as admin and changing or removing posts. Good job!

## Handing in lab 4

---

Hand in a link to your **index.php**.

The following files should be handed in using a zip-file through Blackboard:

- gb.php
- gb-comm.php
- gb-delete.php
- gb-edit.php
- login.php
- logout.php