Ahmadu Bello University, Zaria

Department of Mathematics

**2009/2010 First Semester Examination**

**COSC 405: Web Application Engineering II**

**Date**: May, 2010 **Time Allowed**: 120 Minutes

**Instructions**:

1. Attempt ANY FOUR questions.
2. Write all your answers in the spaces provided on this Question Paper.

**Student’s Registration Number**:………………………………………………………………… **Signature**:……………………………………..

**Date of Examination**:…………………………………………………………………………………**Time**:………………………………………………

**Scores**:

|  |  |  |
| --- | --- | --- |
| **Question** | **Maximum Scores** | **Scores Obtained** |
| 1 | 20 |  |
| 2 | 20 |  |
| 3 | 20 |  |
| 4 | 20 |  |
| 5 | 20 |  |
| 6 | 20 |  |
| **Total** | **80** |  |

1. Answer both the following questions:
   1. (**10 marks**). Compare and contrast the use of PHP and the use of Java applets for accessing a database server.

|  |
| --- |
| A difference between them is that applet code is executed on the client side while PHP code is executed on the server side.   * + - * + 2 marks   Applets programs can be more interactive and they are less likely to overwhelm the server than PHP code. However, applets take some time to download. Applets work well only on newer browsers.  Using applets, clients start executing the applet as a program; requests and updates are sent directly to the database.   * + - * + 4 marks   In non-applet programs, the client obtains and completes an HTML page containing a form from the server and sends it back to the server. The Web server executes a program that processes data from the client, for instance updating a database. He Web server sends a HTML page back to the client.   * + - * + 4 marks |

* 1. (**10 marks**). Identify three different ways of accessing form variables from other PHP files. Explain relative merit of each over the others.

|  |
| --- |
| Three different ways for accessing form variables from PHP files are:   1. Along with or through the URL. In this case the variables and their values will be visible to viewers of the page. Therefore, this is to be avoided when the variables passed around are security sensitive.    * + - * 3 marks 2. Using PHP configuration setting register\_globals. If this is set, then variables can be accessed directly (using their literal names) from other pages. This used to be the default in older versions of PHP. Newer versions set this to OFF by default. register\_globals should always be unset for security considerations.    * + - * 4 marks 3. Using super global arrays ($\_POST, $\_GET, etc). This is the safest way of accessing variables. Your code will be most secure using this method compared to the two earlier methods discussed. Your program will also be more portable among new and old browsers alike.    * + - * 4 marks |

1. Answer both the following questions.
   1. (**14 marks**). Show the HTML-code resulting from a call to the PHP-function my\_universities, presented below, with the array $univs given as argument

|  |
| --- |
| $univs = array (array ("KASU", 2), array ("ABU", 7),array("UMYU",3));  function my\_universities ($list) {  $res = "<table width=50%>";  for ($i = 0; $i < count($list); $i++) {  $name = $list [$i] [0];  $count = $list [$i] [1];  $n = $count;  $bar = "";  while ($n > 0) {  $bar = $bar . "\*";  $n = $n - 1;  }  $res = $res . "<tr><td width=25%>$name</td>  <td width=25%>$count: $bar</td>  </tr>";  }  return "$res.</table>";  } |

**Answer:**

|  |
| --- |
| **<table width=50%>** 🡪 2 marks  **<tr><td width=25%>KASU</td>**  **<td width=25%>2: \*\*</td>** 🡪 4 marks  **</tr>**  **<tr><td width=25%>ABU</td>**  **<td width=25%>7: \*\*\*\*\*\*\*</td>** 🡪 4 marks  **</tr>**  **<tr><td width=25%>UMYU</td>**  **<td width=25%>3: \*\*\*</td>** 🡪 4 marks  **</tr>.**  **</table>** |

* 1. (**6 marks**). Write a PHP function max\_rating that takes as argument an array of university rankings and returns the maximum ranking number.

Hint: calling the function with the following array should return the value 7:

array (array ("KASU", 4), array ("ABU", 7),array("UMYU",5))

It can be advantageous (but not required) to use the foreach command to traverse the array.

**Answer:**

|  |
| --- |
| **function max\_rating($list){ 🡪 1 mark**  **// assume $list is non-empty and that**  **// the first university is currently highest ranked.**  **$max = $list[0][1]; 🡪 1 mark**  **for($j=1; $j<count($list); $j++){ 🡪 2 marks**  **$sub\_array = $list[$j];**  **if ($sub\_array[1] > $max)**  **$max = $sub\_array[1]; 🡪 2 marks**  **}**  **return $max;**  **}** |

1. (**20 marks**). Study the following PHP code carefully and write down its output.

|  |
| --- |
| <?php  $responses = array(1,2,6,4,8,5,9,7,8,10,  1,6,3,8,6,10,3,8,2,7,  6,5,7,6,8,6,7,5, 6,6,  5,6,7,5,6,4,8,6,8,10);  for ( $i= 0; $i <= count($responses); $i=$i+1 ) {  $result[$responses[$i]] += 1;  }  $answer = "<table border='border'><tr>";  for ( $t=1 ; $t <= 10 ; $t=$t+1 ) {  $datum = $result[$t];  $answer = $answer."<td>$datum</td>";  }  echo "$answer.'</tr></table>'";  ?> |

**Answer:**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| * 2 marks for each correct cell  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **2** | **2** | **2** | **2** | **5** | **11** | **5** | **7** | **1** | **3** | |

1. Answer both the following questions.
   1. (**10 marks**). A *well-formed time* consists of two parts separated by a colon (:). The first part should be a number between 0 and 23 (both numbers inclusive). The second part should be a number between 0 and 59 (both numbers inclusive). For the first part, a prefixed zero (0) is optional if the number is between 0 and 9 (both numbers inclusive). For the second part, a prefixed zero (0) is mandatory if the number is between 0 and 9 (both numbers inclusive).

Examples of well-formed times are 23:59, 5:09, and 03:00. Examples of strings that are not well-formed times are 23:2 and 24:00.

Construct a regular expression that is matchable only by well-formed times.

**Answer:**

|  |
| --- |
| ([0-1]?[0-9]|2[0-3]):[0-5][0-9]  3 marks, 3 marks, 1 mark, 3 marks |

* 1. (**10 marks**). Construct a PHP function validate\_time for checking whether a time is well-formed, as described in (a) above. If the argument passed to the function is a well-formed time, the function should return immediately. Otherwise, the function should display an error message in the user’s browser and stop the program by calling the built-in PHP function exit.

Use the PHP function ereg to check that the password is well-formed.

**Answer:**

|  |
| --- |
| function validate\_time ($my\_time) { 🡪 2 marks  if (ereg (’^([0-1]?[0-9]|2[0-3]):[0-5][0-9]$’, $my\_time)) 🡪 5 marks  return; 🡪 1 mark  echo "The $my\_time is not a well-formed time. Go back and try again!”;  exit(); 🡪 2 marks  } |

1. Answer both the following questions
   1. (**10 marks**). Identify and write short notes on two forms of vulnerabilities that can occur in the database layer of an application.

|  |
| --- |
| Vulnerabilities that occur at the database layer are sometimes referred to as SQL injection attacks. SQL injection is a [code injection](http://en.wikipedia.org/wiki/Code_injection) technique that exploits a [security vulnerability](http://en.wikipedia.org/wiki/Security_vulnerability) occurring in the [database](http://en.wikipedia.org/wiki/Database) layer of an [application](http://en.wikipedia.org/wiki/Application_software). The vulnerability is present when user input is either incorrectly filtered for [string literal](http://en.wikipedia.org/wiki/String_literal) [escape characters](http://en.wikipedia.org/wiki/Escape_sequences) embedded in [SQL](http://en.wikipedia.org/wiki/SQL) statements or user input is not [strongly typed](http://en.wikipedia.org/wiki/Strongly-typed_programming_language) and thereby unexpectedly executed. It is an instance of a more general class of vulnerabilities that can occur whenever one programming or scripting language is embedded inside another.   * + - * + 2 mark   This vulnerability can take the following forms:   1. Incorrectly filtered escape characters   This form of SQL injection occurs when user input is not filtered for [escape characters](http://en.wikipedia.org/wiki/Escape_sequences) and is then passed into an [SQL](http://en.wikipedia.org/wiki/SQL) statement. This results in the potential manipulation of the statements performed on the database by the end user of the application.  The following line of code illustrates this vulnerability:  statement = "SELECT \* FROM users WHERE name = '" + userName + "';"  This SQL code is designed to pull up the records of the specified username from its table of users. However, if the "userName" variable is crafted in a specific way by a malicious user, the SQL statement may do more than the code author intended. For example, setting the "userName" variable as  a' or 't'='t  renders this SQL statement by the parent language:  SELECT \* FROM users WHERE name = 'a' OR 't'='t';   * + - * + 4 marks  1. Incorrect type handling   This form of SQL injection occurs when a user supplied field is not [strongly typed](http://en.wikipedia.org/wiki/Strongly_typed) or is not checked for [type](http://en.wikipedia.org/wiki/Data_type) constraints. This could take place when a numeric field is to be used in a SQL statement, but the programmer makes no checks to validate that the user supplied input is numeric. For example:  statement := "SELECT \* FROM userinfo WHERE id = " + a\_variable + ";"  It is clear from this statement that the author intended a\_variable to be a number correlating to the "id" field. However, if it is in fact a [string](http://en.wikipedia.org/wiki/String_%28computer_science%29) then the [end user](http://en.wikipedia.org/wiki/End_user) may manipulate the statement as they choose, thereby bypassing the need for escape characters. For example, setting a\_variable to  1;DROP TABLE users  will drop (delete) the "users" table from the database, since the SQL would be rendered as follows:  SELECT \* FROM userinfo WHERE id=1;DROP TABLE users;   * + - * + 4 marks  1. Vulnerabilities inside the database server  Vulnerabilities inside the database server Sometimes vulnerabilities can exist within the database server software itself, as was the case with the [MySQL](http://en.wikipedia.org/wiki/MySQL) server's mysql\_real\_escape\_string() function. This would allow an attacker to perform a successful SQL injection attack based on bad Unicode characters even if the user's input is being escaped. This bug was patched with the release of version 5.0.22 (released on 24th May 06).   * + - * + 4 marks |

* 1. Consider the following MySQL database table called Expense:

|  |  |  |  |
| --- | --- | --- | --- |
| **kind** | **dept** | **year** | **value** |
| Salary | Acquisition | 2001 | 490,000 |
| Salary | Sales | 2002 | 1,500,000 |
| Salary | Acquisition | 2002 | 500,000 |
| Coffee | Acquisition | 2003 | 800 |
| Coffee | Sales | 2003 | 300 |
| Salary | Sales | 2003 | 1,600,000 |
| Salary | Acquisition | 2003 | 510,00 |

Write MySQL queries to find

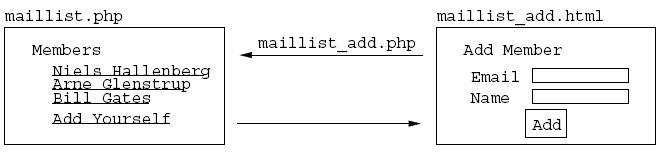
1. (**5 marks**). for each year, the total expenses of each kind.

|  |
| --- |
| SELECT year, dept, SUM(value) 🡪 5 marks  FROM Expense  ORDER BY year, dept; |

1. (**5 marks**). the total expenses grouped by department, kind, and year for the years 2002-2003 in which the sum of the expenses is less than 1,000,000.

|  |
| --- |
| SELECT kind, dept, year, SUM(value) 🡪 5 marks  FROM Expense  WHERE 2002 <= year AND year <= 2003  GROUP BY kind, dept, year  HAVING SUM(value) >= 1000000; |

1. Consider the following site map for an online registration Web application.



Unlabeled arrows represent hyperlinks to a new HTML page, possibly generated by a PHP script. Labeled arrows represent transactions that update the database by running a PHP script.

Answer the following questions:

1. (**10 marks**). The following is the code for the file maillist\_add.php which adds a user to a maillist table. Complete the missing code by writing in the empty rectangles following the comments.

|  |  |  |  |
| --- | --- | --- | --- |
| <?php  // Establish database connection  $db = mysql\_connect("localhost", "root");  mysql\_select\_db("mail", $db) or die (mysql\_error($db));  // read new member’s Name and Email from the maillist\_add.html form   |  | | --- | | $email = $\_POST['email']; 🡪 1.5 marks  $name = $\_POST['name']; 🡪 1.5 marks |   //Add the new member’s Name and Email into the maillist table of the database   |  | | --- | | mysql\_query("INSERT INTO Mailist (email, name) 🡪 4 marks  VALUES ('$email', '$name')"); |   // Jump to maillist.php   |  | | --- | | header("Location: mailist.php"); 🡪 3 marks |   ?> |

1. (**10 marks**). The following is the code for the file maillist.php that displays the updated list of members in the maillist.

|  |  |  |
| --- | --- | --- |
| <html><title>Members</title>  <body><h2>Members</h2>  <ul>  <?php  // Establish database connection  $db = mysql\_connect("localhost","root");  mysql\_select\_db('mail', $db) or die (mysql\_error($db));  // Extract rows from the table   |  | | --- | | $rows = mysql\_query("SELECT email, name FROM Mailist"); 🡪 3 marks |     // Iterate through the rows and display each row as a list item as shown in the figure above.   |  | | --- | | while ( $row = mysql\_fetch\_array($rows) ) { 🡪 3 marks  // Display a single row  echo '<li><a href="mailto:'.$row['email'].'">'.$row[1].'</a> </li>’; 🡪 4 marks  } |     ?>  </ul>  <p><a href="maillist\_add.html">Add Yourself</a>  </body></html> |