

## Contents

Topics	Page No.
Unit-8: PHP Advance Features	1
8.0 Structure of PHP Advance Features.....	1
8.1 Learning Outcomes .....	1
8.2 Form Handling in PHP.....	1
8.3 Cookies and Session Management in PHP.....	7
8.3.1 Cookies in PHP.....	7
8.3.2 Sessions in PHP.....	11
8.4 Database handling in PHP.....	13
8.5 Creating a simple database and database operations...	15
8.6 Self-Assessment Questions.....	20
8.7 Self-Assessment Activities.....	20
8.8 Multiple-Choice Questions.....	21
8.9 Key Answers to Multiple-Choice Questions.....	22
8.10 Summary .....	23
8.11 Keywords.....	24
8.12 Recommended Resources for Further Reading.....	24

## UNIT 8 – PHP Advanced Features

---

### 8.0 Structure of PHP Advanced Features

- 8.1 Learning Outcomes
  - 8.2 Form Handling in PHP
  - 8.3 Cookie and Session Management in PHP
  - 8.4 Database Handling in PHP
  - 8.5 Creating a simple database and database operations
  - 8.6 Self-Assessment Questions
  - 8.7 Self-Assessment Activities
  - 8.8 Multiple-Choice Questions
  - 8.9 Key answers to multiple-choice questions
  - 8.10 Summary
  - 8.11 Keywords
  - 8.12 Recommended resources for further reading
- 

### 8.1 Learning Outcomes:

After the successful completion of this unit, the student will be able to:

- Describe handling of form data in PHP using GET and POST methods[L2]
  - Explain cookies and session management in PHP.[L2]
  - Develop a simple database application in PHP using MySQL database.[L5]
- 

### 8.2 Form Handling in PHP

In PHP, form handling refers to the process of collecting form data submitted through XHTML forms and processing that data on the server side. This is a crucial aspect of web development, as it allows users to interact with a website by providing input through forms. Form data from XHTML is sent to the server either using the GET or POST method. The method attribute in the XHTML form element is used to specify whether data is being sent using the GET or POST method. By default, the GET method is used to send data. The action attribute of the XHTML form element specifies the URL where the form data should be sent.

## GET and POST methods in PHP

In PHP, the GET and POST methods are two ways to transfer data from the client (browser) to the server. Both methods are used with XHTML forms to submit user input, but they differ in the way they are sent and how it is handled on the server side.

### GET Method:

**Data in URL:** When a form is submitted using the GET method, the form data is appended to the URL as query parameters.

**Visibility:** The data is visible in the URL, making it less secure. It's suitable for non-sensitive data and when it is necessary to share the URL with others.

**Limitations:** There are restrictions on the amount of data that can be sent, and certain characters may need to be encoded.

In PHP, the implicit `$_GET` array is used to retrieve form data sent using the GET method.

Example: XHTML Form code with GET Method

```
<form action="process_form.php" method="GET">
    Name: <input type="text" id="name" name="name">
    <input type="submit" value="Submit">
</form>
```

PHP Script code to Process GET Data

```
<?php
    // process_form.php
    $name = $_GET['name'];
    echo "Hello, $name!";
?>
```

Generated URL: when data is sent using GET method

`http://localhost/process_form.php?name=Mary`

### POST Method:

**Data in HTTP Request Body:** When a form is submitted using the POST method, the form data is sent in the HTTP request body and is not visible in the URL.

Visibility: The data is not visible in the URL, which provides a higher level of security, making it suitable for sensitive information like passwords.

Data Size: The POST method can handle larger amounts of data compared to GET, and it supports binary data.

In PHP, the implicit `$_POST` array is used to retrieve form data sent using the POST method.

Example: XHTML Form code with POST Method

```
<form action="process_form.php" method="POST">
    Password:<input type="password" id="password" name="password">
    <input type="submit" value="Submit">
</form>
```

PHP Script code to Process POST Data

```
<?php
    // process_form.php
    $password = $_POST['password'];
    // Process and validate the password
?>
```

Generated URL: when data is sent using the POST method

`http://localhost/process_form.php`

In PHP, an implicit `$_REQUEST` array can be used to access both GET and POST data, but generally `$_GET` or `$_POST` arrays are used explicitly based on the form's method used for clarity and security.

When sensitive information like passwords is to be sent using GET is avoided, as the data is visible in the URL.

Choosing between GET and POST is based on the nature of the data being transmitted and the security requirements of the application.

## Handling Forms in PHP

- When PHP is used for form handling, PHP script handles form data in an XHTML document that defines the form and even PHP script can be embedded in an XHTML document.

The value of the action attribute in the form tag specifies the PHP script that handles it. `<form action="process_form.php" method="POST">`

- The implicit arrays for form values, `$_POST` and `$_GET` are used depending on the method used to send data.
- These arrays have keys that match the form element names and values that were input by the client during execution.

Example: To create a login form using an XHTML document. Retrieve the form data in a PHP script, and validate it – Check whether the user has entered the username and password or if the fields are empty. If any of the fields are empty display an error message else a success message.

XHTML document: Login Form to collect input from the user

```
<html>
</head>
<body>
    <h4> Login Form for entering Student Credentials </h4>
    <form name="f1" method="POST" action="display.php">
        Username: <input type="text" name="txtname"/> <br/><br/>
        Password: <input type="password" name="txtpassword"/>
                <br/><br/>
        <input type="submit" value="submit">
        <input type="reset" value="reset"/>
    </form>
</body>
</html>
```

Figure 8.1: XHTML document with login form

An XHTML document contains textboxes for entering username and password

```
Username: <input type="text" name="txtname"/> <br/><br/>
Password: <input type="password" name="txtpassword"/>
```

The method used to send data is "POST" and the PHP script to be executed after submitting the form is specified in the action attribute of the opening form tag as follows -

```
<form name="f1" method="POST" action="display.php">
```

Here "display.php" is the PHP script that gets executed at the click of the submit button.

"display.php"

```
<?php
//retrieve form data and store in PHP variables
$name=$_POST['txtname'];
$password=$_POST['txtpassword'];
$n1 = strlen($name);
$n2 = strlen($password);
if ($n1==0)
{
    print "Username not entered correctly";
}
if ($n2==0)
{
    print "Password not entered correctly";
}
if ($n1 !=0 && $n2 !=0)
{
    print "Hello ".$name.", login data entered successfully";
}
?>
```

Figure 8.2: PHP script for handling form data

The form data [Username and password] is retrieved using the implicit \$\_POST array.

```
$name=$_POST['txtname'];
$password=$_POST['txtpassword'];
```

The strlen() method is used to find the length of the username and password entered by the user.

```
$n1 = strlen($name);
$n2 = strlen($password);
```

If the length is zero it means data is not entered.

If \$n1 is equal to zero, it means the username is not entered

If \$n2 is equal to zero, it means the password is not entered

During execution, if the user has not entered the username, then the message "Username not entered correctly" is displayed as follows –

Output 1:

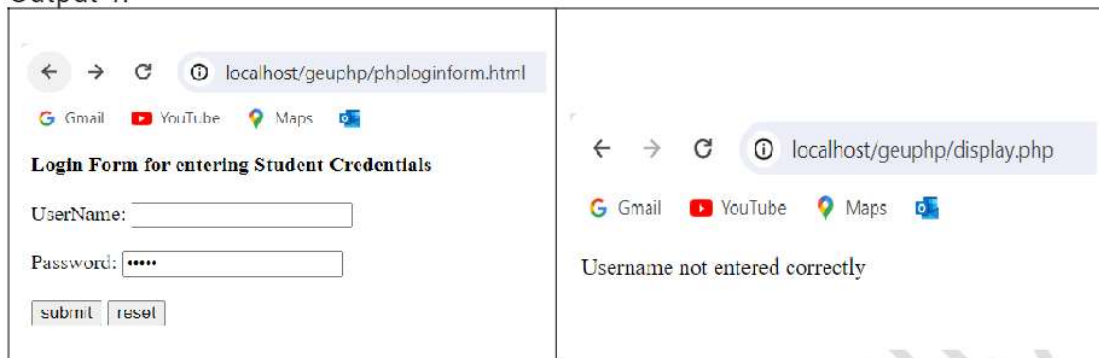


Figure 8.3: Output – Displaying error message username not entered

During execution, if the user has not entered the password then the message "Password not entered correctly" is displayed as follows –

Output 2:

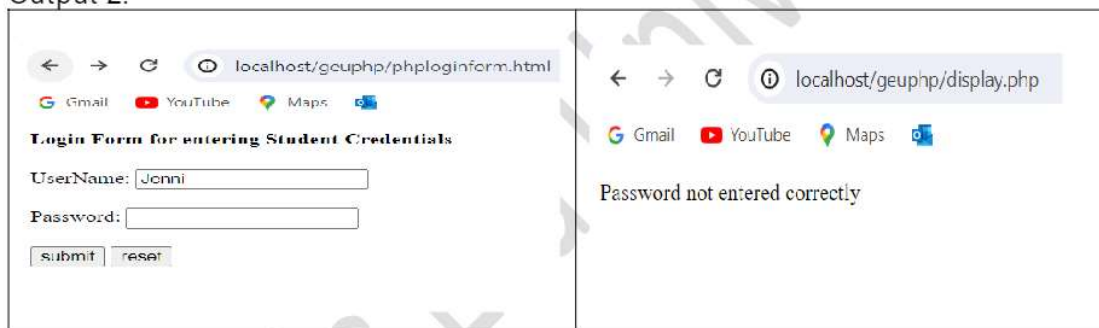


Figure 8.4: Output – Displaying error message password not entered

If the username and password are entered correctly, then the success message "login data entered successfully" is displayed as follows –

Output 3:

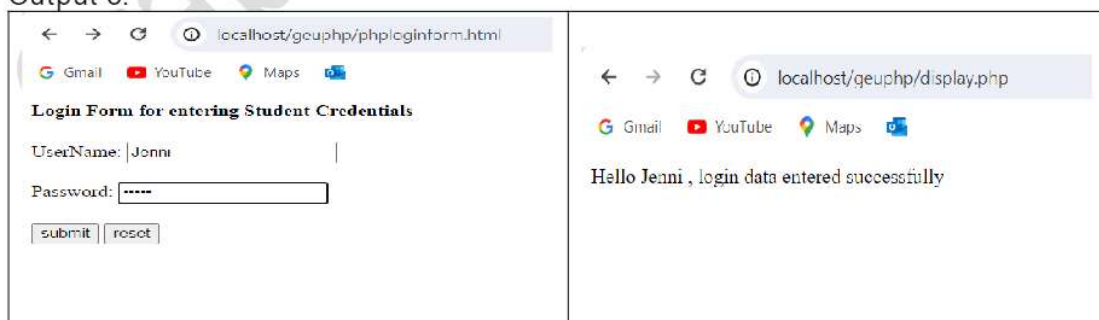


Figure 8.5 Output – Displaying success message when username and password are correctly entered



## 8.3 Cookie and Session Management in PHP

In PHP, cookies, and session management are essential features for handling user data and maintaining the state between different requests. Tracking the users can be done using cookies and sessions in PHP.

### 8.3.1 Cookies in PHP

In PHP, a cookie is a small text file that is stored on the user's (client) computer. The information in the file consists of the name of the cookie and a textual value. A cookie is created by some software on the server and is sent as part of the HTTP response header to the client where it is stored. The communication between a browser and the server is done through the HTTP protocol. An HTTP request is a message from a browser to a server and an HTTP response is a message from a server to a browser. The header part of the HTTP request and response consists of a header that includes the information of one or more cookies.

Only the server that created a cookie can receive the cookie from the browser, so a particular cookie is information that is exchanged exclusively between one specific browser and one specific server. Every browser request includes all of the cookies its host machine has stored that are associated with the web server to which the request is directed.

The client can change the browser setting to refuse to accept the cookies as sometimes it may be a privacy concern. The browser user can view, alter, or delete a cookie at any time as cookies are stored as text files on the client's computers. Most browsers have a limit on the number of cookies that can be accepted from a particular server.

A cookie can be created, altered, or deleted by using the `setcookie()` function. The `setcookie()` function should be used in the PHP code before the `<html>` tag as cookie information is sent as part of the header. The maximum file size of the cookie is 4KB.

Syntax:

```
setcookie(name, value, expire, path, domain, secure);
```

The `setcookie()` function can have one or more parameters. The description of the parameters is given in Table 8.1.



Table 8.1: The setcookie() parameters and their description

Parameter Name	Description
name	It is a first and mandatory argument. It is used to specify the name of the cookie and it should be a string.
value	It is used to store the value of the cookie.
expire	It is used to specify the expiry time of a cookie in seconds given as an integer. The default value is zero, which indicates the cookie is destroyed at the end of the current session. It is an optional argument.
path	It is used to specify the path through which the cookie will be accessible from the server. It is an optional argument.
domain	It is used to specify the domain for which the cookie is available. It is an optional argument.
secure	It is set to 1, the cookie is available and sent only over HTTPS connection.

#### Creating/Setting a cookie:

A cookie is created/set by using a setcookie() function.

Example: This example sets a cookie named "user" with the value "GEU" that expires after one hour (time() + 3600).

```
<?php
    // Set a cookie with a name, value, and expiry time
    $cookie_name = "user";
    $cookie_value = "GEU";
    $expiry_time = time() + 3600; // 1 hour from now
    // Set the cookie
    setcookie($cookie_name, $cookie_value, $expiry_time);
?>
```

#### Retrieving a cookie:

All cookies that arrive with the HTTP request are stored in the implicit array \$\_COOKIE[], which contains cookie names as keys and cookie values as values.

The cookie can be retrieved from the implicit array as -

```
$val = $_COOKIE[$cookie_name];
```

Example :

If the name of the cookie is "user" it can be retrieved as –

```
$val = $_COOKIE["user"];
```

The `isset()` function can be used to check whether the cookie is present in the implicit array. If the cookie is present it returns true otherwise false.

Example: To retrieve the cookie with the name "user", and if it exists display the name of the cookie and its value otherwise display the cookie does not exist.

The PHP code is written as -

```
<?php
    $cookie_name = "user";
    if(!isset($_COOKIE[$cookie_name]))
    {
        echo "Cookie named <b>'". $cookie_name . "' </b> is not set! <br/>";
    }
    else
    {
        echo "Cookie by the name <b> '" . $cookie_name . "' </b> is set
            <br/><br/>";
        echo "Value is : <b>'". $_COOKIE[$cookie_name]. "' </b><br/>";
    }
?>
```

Deleting a cookie:

For deleting a cookie the expiry time can be set a negative value.

```
<?php
    // Set the expiry time of the cookie to a negative value
    $cookie_name = "user";
    $cookie_value = "GEU";
    $expiry_time = time() - 3600;    // negative value deletes the cookie
    // Deleting the cookie
    setcookie($cookie_name, $cookie_value, $expiry_time);
?>
```

A complete example of setting, retrieving, and displaying the cookie value along with its name, if it exists is given in Figure 8.6

```
<?php
    // Set a cookie with a name, value, and expiry time
    $cookie_name = "user";
    $cookie_value = "GEU";
    $expiry_time = time() + 3600; // 1 hour from now
    // Set the cookie
    setcookie($cookie_name, $cookie_value, $expiry_time);
?>
<html>
<body>
<?php
    if(!isset($_COOKIE[$cookie_name]))
    {
        echo "Cookie named <b>'" . $cookie_name . "'</b> is not set!
            <br/>";
    }
    else
    {
        echo "Cookie by the name <b> '" . $cookie_name . "' </b>
            is set <br/><br/>";
        echo "Value is : <b>'" . $_COOKIE[$cookie_name]. "' </b><br/>";
    }
?>
</body>
</html>
```

Figure 8.6: Creating/Setting a cookie, retrieving the value, and displaying it in PHP

When the PHP code is executed for the first time, the cookie value is set using the `setcookie()` function. It is then sent to the client as part of the HTTP header so the `setcookie()` function should appear before the `<html>` tag. The cookie information gets stored on the client's computer during the first execution/run and can be retrieved/alterd during further executions.

During the first execution – it sets the value of the cookie but is presently not available. So `isset()` function returns false and the message "cookie is not set" is displayed.

Output: During the first execution



Figure 8.7: Output – Displaying the value is not set as it is presently not available

Output: During the second execution, retrieve the value and display it.

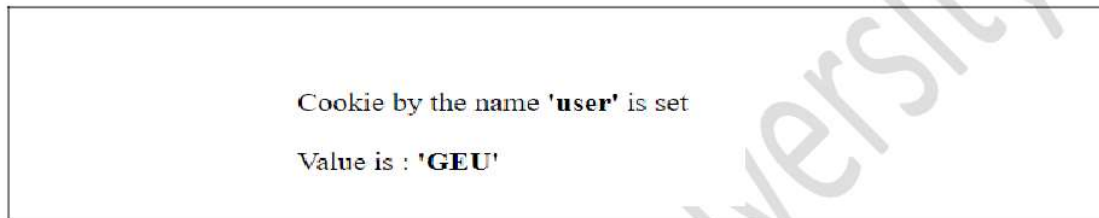


Figure 8.8: Output - Displaying the retrieved cookie value

### 8.3.2 Sessions in PHP

Sessions in PHP provide a way to preserve data across multiple pages during a user's visit to a website. Unlike cookies, which are stored on the user's computer, sessions store data on the server. Sessions are commonly used to maintain user authentication, store user preferences, and manage other temporary data related to a user's interaction with a website. Rather than using one or more cookies to store information on a client computer a single session array can be used to store information about a particular session on the server. A session array stores a unique session ID for a particular session. A significant difference between cookies and sessions is, that cookies are stored on the client's computer whereas session information is stored on the server.

Starting a session:

To use sessions in PHP, a session has to be started using the `session_start()` function. A first call to the `session_start()` function creates a session ID. A session ID is an internal value that identifies a particular session. Typically, this function is called at the beginning of each script that needs to access or store session data. Subsequent calls to the `session_start()` function in the same session retrieve the information stored in the `$_SESSION[]` array.

```
<?php
    session_start();
    // rest of the PHP code
?>
```

It's a good practice to call `session_start()` at the beginning of every PHP script where session variables are used.

#### Storing Data in Sessions:

Data can be stored in the session by using the `$_SESSION` global array. This array is accessible across different pages as long as the session is active.

Example:

```
<?php
    // Storing data in a session variable
    $_SESSION['user_id'] = 999;
    $_SESSION['username'] = 'GEU';
?>
```

In this example, two values (`user_id` and `username`) are stored in the session.

#### Accessing Session Data:

To retrieve session data on subsequent pages, the `$_SESSION[]` array is used.

Example:

```
<?php
    // Accessing session data
    echo "User ID: ". $_SESSION['user_id'];
    echo "Username: ". $_SESSION['username'];
?>
```

The session data is available as long as the session is active. The session usually persists until the user closes their browser or the session expires.

#### Destroying a Session:

One can end a session and clear all session data using the `session_destroy()` function.

Example:

```
<?php
    session_destroy();    // Ending a session
?>
```

However, `session_destroy()` alone does not unset the session variables. To clear session data and unset the session variables, `session_unset()` is called before calling `session_destroy()`.

Example:

```
<?php
    // Clearing session data and unsetting variables
    session_unset();
    session_destroy();
?>
```

Calling `session_destroy()` does not immediately delete the session data on the server; it simply marks the session as closed. PHP eventually cleans up expired sessions based on the setting.

## 8.4 Database Handling in PHP

There are various methods/functions used in PHP to handle/access MySQL database.

Functions in PHP for Database Access :

- `mysqli_connect();`
- `die();`
- `mysqli_close();`
- `mysqli_select_db();`
- `mysqli_num_row();`
- `mysqli_num_fields();`
- `mysqli_fetch_array();`
- `mysqli_fetch_row();`

### 1. `mysqli_connect()`

The method `mysqli_connect()` is used to connect to a MySQL server.

It takes three parameters, all of which are optional.

The first parameter is the hostname that is running MySQL. The default is localhost.

The second parameter is the username for MySQL.

The third parameter is the password for the database. The default is empty.

Example-

```
$db=mysqli_connect();
```

Or `$db=mysqli_connect($hostname, $username, $password);`



## 2. die()

The call to die() is used in conjunction with mysqli\_connect(). If the connect operation fails, it returns a false, in which case the die() method gets invoked.

Example-

```
$db = mysqli_connect($hostname, $username, $password) or  
die("Unable to connect to MySQL");
```

## 3. mysqli\_close()

The connection to a database is terminated with the mysqli\_close() function. This function is not necessary when using MySQL through a PHP script because the connection will be closed implicitly when the script terminates.

Example-

```
mysqli_close();  
or  
mysqli_close($db);
```

## 4. mysqli\_select\_db()

This method is used to select the MySQL database.

Example-

```
mysqli_select_db("test");
```

## 5. mysqli\_query()

MySQL query operations are executed through the mysqli\_query() function. Typically, the query operation, in the form of a string literal, is assigned to a variable. Then mysqli\_query() is called with the variable as its parameter.

Example-

```
$query="SELECT * from Student";  
$result=mysqli_query($query);
```

## 6. mysqli\_num\_row()

This is to determine the number of rows returned by the query

Example-

```
$query="SELECT * from Student";  
$result=mysqli_query($query);  
$no = mysqli_num_row($result);
```

#### 7. mysqli\_num\_fields()

This is to determine the number of fields in the result row.

Example-

```
$query="SELECT * from Student";  
$result=mysqli_query($query);  
$no = mysqli_num_row($result);
```

#### 8. mysqli\_fetch\_array()

The rows of the result can be retrieved into an array by using this function.

Example-

```
$query="SELECT * from Student";  
$result=mysqli_query($query);  
while($ary = mysqli_fetch_array($result))  
{  
    ... // PHP code to print the values  
}
```

#### 9. mysqli\_fetch\_row()

The rows of the result can be retrieved into an array by using this function.

Example-

```
$query="SELECT * from Student";  
$result=mysqli_query($query);  
while($ary = mysqli_fetch_row($result))  
{  
    ... // PHP code to print the values  
}
```

### 8.5 Creating a simple database and database operations

Creating a simple database application using PHP and MySQL involves several steps, setting up a database, creating an XHTML form for collecting user information, creating a PHP script that collects form data and establishes a connection to a MySQL database, and then performing CRUD (Create, Read, Update, Delete) operations, and displaying the retrieved data on the browser.

Example: To create an XHTML form that collects student information [student name, age] and write a PHP script that collects form data and stores the information in MySQL database. Also, retrieves all the data and displays the same.

Steps for creating a simple database application for storing data in the MySQL database, retrieving the same, and displaying it using PHP script.

### 1. Create a database in phpMyAdmin

The database created here is "geustudb", and the table name is "student" with two fields "studname" and "studage". The structure of the table is shown in Figure 8.9.

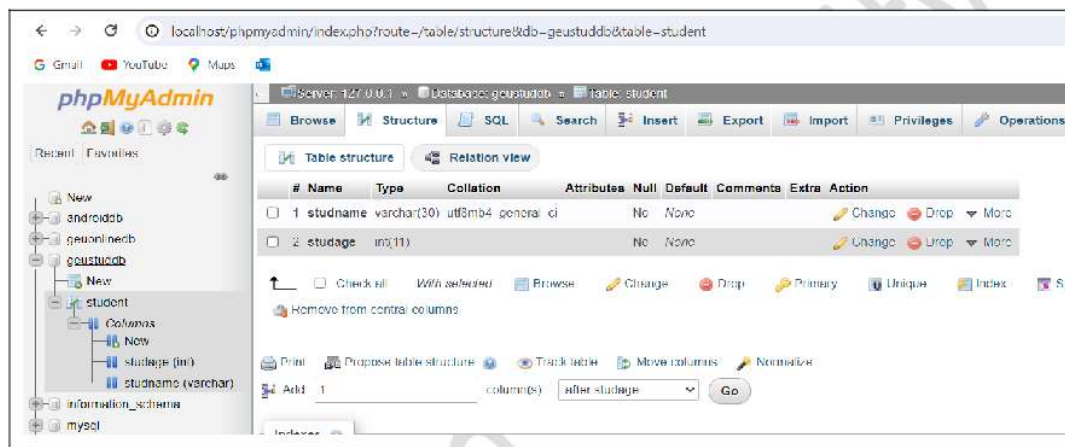


Figure 8.9: Database and structure of table "student"

### 2. The XHTML document with the form for accepting input from the user is created as shown in Figure 8.10.

XHTML document – "phpform1.html"

```
<html>
</head>
<body>
    <h4> Form for entering Student Information </h4>
    <form name="f1" method="POST" action="storedisplay.php">
        Name:      <input type="text" name="txtname"/> <br/><br/>
        Age:       <input type="text" name="txtage"/> <br/><br/>
                 <input type="submit" value="submit">
                 <input type="reset" value="reset"/>
    </form>
</body>
</html>
```

Figure 8.10: XHTML document with a form for collecting input from the user

3. Create a PHP Script for collecting form data, storing the data in MySQL database, retrieving the same, and displaying it.

PHP script – "storedisplay.php"

```
<?php

// retrieve form data and store in PHP variables
$name=$_POST['txtname'];
$age=$_POST['txtage'];

// connect to MySQL database
$dbh=mysqli_connect("localhost","root","","geustudb") or
    die("Unable to connect");

// query
$qry="insert into student values('$name','$age')";

/* insert data into the student table */
$result=mysqli_query($dbh,$qry);

/* Retrieve all records from the student table */
$qry1="select * from student";
$result1=mysqli_query($dbh,$qry1);
$n=mysqli_num_rows($result1);
/* If data is present in the table display it */
if($n>0)
{
    print "<table border=2 width=25%>";
    print "<caption>Data from the Student table </caption>";
    print "<tr><th>Name</th><th>Age</th></tr>";
    /* fetch data from the recordset and display it one record at a time*/
    while($ary=mysqli_fetch_row($result1))
    {
        print "<tr>";
        foreach($ary as $i)
        {
            print "<td>$i</td>";
        }
    }
}
```

```

    }
    print"</tr>";
    print "</table>";

}
else
{
    print "Error in storing data into table";
}
mysqli_close($dbh);
?>

```

Figure 8.11: PHP script - collects form data, stores in MySQL database, retrieves and displays it on the browser

On form submission, the data is retrieved using an implicit `$_POST` array. The textbox names given in the form are used as indexes to retrieve the data.

```

$name=$_POST['txtname'];
$age=$_POST['txtage'];

```

To connect to the MySQL database "geustddb" - the method used is as follows –  
`$dbh=mysqli_connect("localhost","root","","geustddb")` or  
`die("Unable to connect");`

The form data collected is stored in the "student" table by executing the query –  
`$qry="insert into student values('$name','$age')";`  
`$result=mysqli_query($dbh,$qry);`

The data is retrieved in a recordset, from the "student" table by using the query –  
`$qry1="select * from student";`  
`$result1=mysqli_query($dbh,$qry1);`

The data is fetched from the recordset "\$result1" one row at a time and displayed using a while loop -

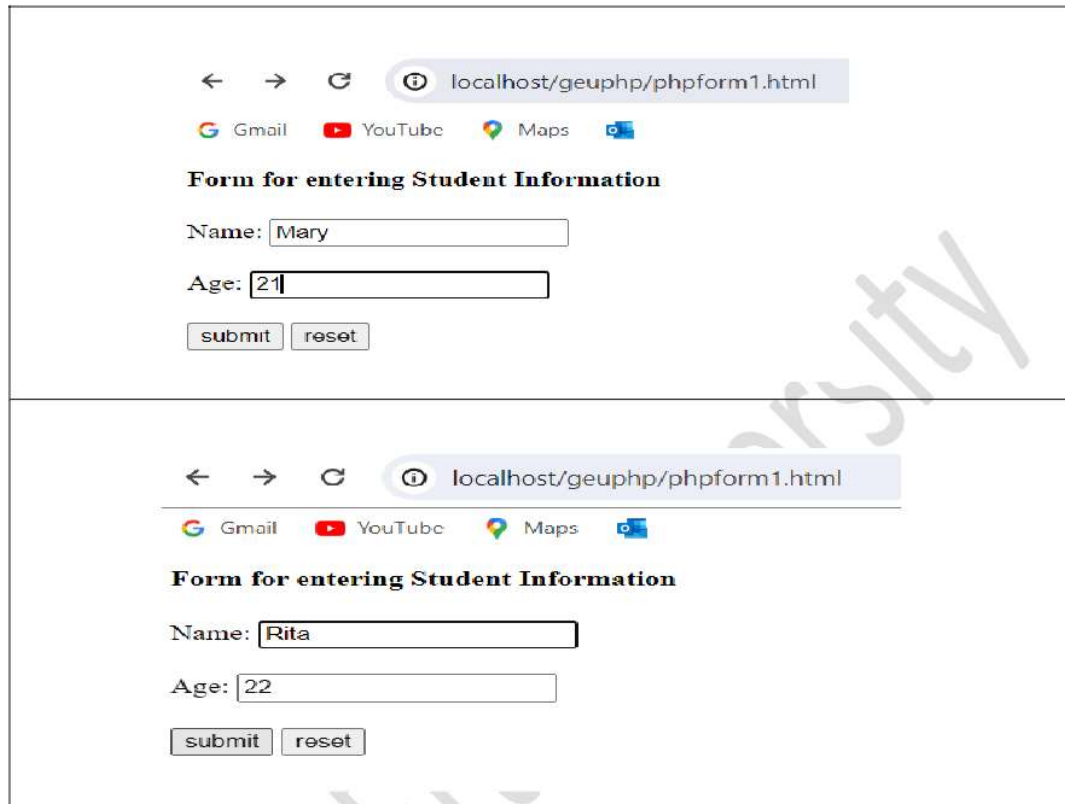
```

while($ary=mysqli_fetch_row($result1))

```

4. On executing the XHTML document for collecting the user input, the output is as shown in Figure 8.12.

Output:

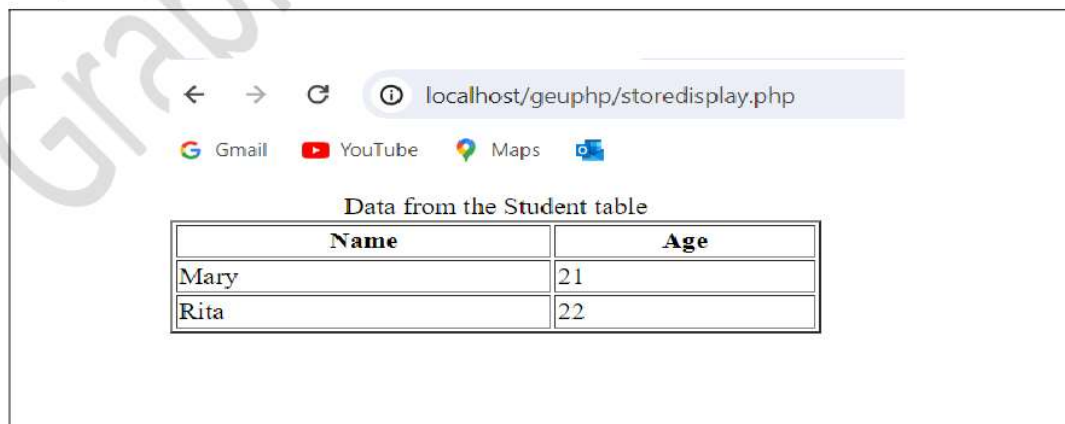


The figure consists of two screenshots of a web browser. The browser's address bar shows the URL 'localhost/geuphp/phpform1.html'. The page title is 'Form for entering Student Information'. The form contains two input fields: 'Name:' and 'Age:'. In the first screenshot, the 'Name' field contains 'Mary' and the 'Age' field contains '21'. In the second screenshot, the 'Name' field contains 'Rita' and the 'Age' field contains '22'. Both screenshots include a 'submit' button and a 'reset' button.

Figure 8.12: Output - XHTML document with a form for collecting input from the user

5. On processing the PHP script, the data is stored in the "student" table, later retrieved and displayed on the browser as shown in Figure 8.13.

Output:



The figure shows a screenshot of a web browser. The browser's address bar shows the URL 'localhost/geuphp/storedisplay.php'. The page title is 'Data from the Student table'. Below the title is a table with two columns: 'Name' and 'Age'. The table contains two rows of data: 'Mary' with '21' and 'Rita' with '22'.

Name	Age
Mary	21
Rita	22

Figure 8.13: Output – Retrieving data from the table and displaying the same



## 8.6 Self-Assessment Questions

- Q1. Explain the concept of form handling in PHP with an example. [8 marks, L2]
- Q2. What are the different ways of submitting form data to the server in PHP? Explain with an example. [6 marks, L2]
- Q3. Explain the differences between the GET and POST methods used for submitting form data to the server in PHP. [4 marks, L2]
- Q4. Write an XHTML document to create a login form with the following – [8 marks, L2]

- A text box to collect the user's name
- A text box to collect the password
- A submit and reset button

Give appropriate labels for the same

Write a PHP program that collects the form data sent using the POST method and displays the username along with a greeting message.

- Q5. Explain the use of the setcookie() function and describe the parameters used with it in PHP. [4 marks, L2]
- Q6. What is a cookie? Where are cookies stored? How are cookies created, retrieved, and deleted in PHP explain with suitable examples. [Write the appropriate code for the same] [ 8 marks, L2]
- Q7. What is a session? Where is session information stored? Describe the use of sessions in PHP. With suitable examples explain how session variables are used in web pages in PHP. [8 marks, L2]
- Q8. Write a PHP program to collect student information [enrollment\_no, student\_name, gender, address, age) and store it in MySQL database. Also, retrieve all the data and display the same. [8 marks, L2]
- Q9. Explain the methods used in PHP to handle the MySQL database. [8 marks, L2]
- Q10. Explain the use of the following in PHP. [3 marks, L2]
- \$\_POST[]
  - \$\_COOKIE[]
  - \$\_SESSION[]

## 8.7 Self-Assessment Activities

- A1. Write a program in PHP, to create a cookie to store information of your choice. Later alter its value and delete it. Check the output during each execution.
- A2. Write a program in PHP, to create session variables to store information of your choice. Later use the values across other web pages.

## 8.8 Multiple-Choice Questions

1. The purpose of the action attribute in an XHTML form tag is \_\_\_\_\_. [1 mark, L1]
  - a) It specifies the method to be used for form submission.
  - b) It defines the name of the form.
  - c) It specifies the URL where the form data should be sent.
  - d) It sets the encoding type for form data.
2. To access form data sent using the POST method, \_\_\_\_\_ implicit array is used in PHP. [1 mark, L1]
  - a) \$\_GET[]
  - b) \$\_POST[]
  - c) \$\_REQUEST[]
  - d) \$\_SERVER[]
3. In PHP, a cookie is set using \_\_\_\_\_. [1 mark, L1]
  - a) setcookie()
  - b) create\_cookie()
  - c) cookie\_set()
  - d) \$\_COOKIE['name'] = 'value';
4. \_\_\_\_\_ is used to retrieve the value of a cookie named "user" in PHP. [1 mark, L1]
  - a) get\_cookie("user")
  - b) retrieve\_cookie("user")
  - c) cookie\_value("user")
  - d) \$\_COOKIE["user"]
5. \_\_\_\_\_ are mechanisms in PHP used to manage and persist user data across multiple pages or visits. [1 mark, L1]
  - a) Events and Event handlers
  - b) GET and POST methods
  - c) Cookies and Session
  - d) None of the above
6. \_\_\_\_\_ is used to start a session in PHP. [1 mark, L1]
  - a) start\_session()

- b) session\_start()
  - c) init\_session()
  - d) begin\_session()
7. \_\_\_\_ is an example, of setting a session variable in PHP.[1 mark, L1]
- a) session\_add("username", "Robert");
  - b) set\_session("username", "Robert");
  - c) \$\_SESSION["username"] = "Robert"
  - d) create\_session("username", "Robert");
8. The \_\_\_\_\_ statement is true, regarding cookies and sessions in PHP.[1 mark, L1]
- a) Cookies are stored on the client's computer, whereas the session on the server
  - b) Cookies are stored on the server, whereas the session on the client's computer
  - c) Cookies and Sessions both are stored on the server
  - d) Cookies and Sessions both are stored on the client's computer
9. The purpose of the mysqli\_connect() function in PHP is to \_\_\_\_\_. [1 mark, L1]
- a) create a new database
  - b) establish a connection to a MySQL database
  - c) insert data into a database
  - d) execute a SELECT query
10. \_\_\_\_ is used to execute an SQL query in PHP using the MySQL.[1 mark, L1]
- a) execute\_query()
  - b) mysqli\_execute()
  - c) mysqli\_query()
  - d) run\_query()

## 8.9 Key Answers to Multiple-Choice Questions

1. The purpose of the action attribute in an HTML form tag is, it specifies the URL where the form data should be sent.[c]
2. To access form data sent using the POST method, \$\_POST[] implicit array is used in PHP.[b]
3. In PHP, a cookie is set using setcookie().[a]
4. \$\_COOKIE["user"] is used to retrieve the value of a cookie named "user" in PHP.[d]



5. Cookies and Sessions are mechanisms in PHP used to manage and persist user data across multiple pages or visits.[c]
6. `session_start()` is used to start a session in PHP.[b]
7. `$_SESSION["username"] = "Robert"` is an example, to set a session variable in PHP.[c]
8. The Cookies are stored on **the client's** computer, whereas the session on the server statement is true, regarding cookies and sessions in PHP.[a]
9. The purpose of the `mysqli_connect()` function in PHP is to establish a connection to a MySQL database.[b]
10. `mysqli_query()` is used to execute an SQL query in PHP using MySQL.[c]

## 8.10 Summary

In PHP, form handling refers to the process of collecting form data submitted through XHTML forms and processing that data on the server side. This is a crucial aspect of web development, as it allows users to interact with a website by providing input through forms. Form data from XHTML is sent to the server either using the GET or POST method. The method attribute in the XHTML form element is used to specify whether data is being sent using the GET or POST method. By default, the GET method is used to send data. The action attribute of the XHTML form element specifies the URL where the form data should be sent.

In PHP, the GET and POST methods are two ways to transfer data from the client (browser) to the server. Both methods are used with XHTML forms to submit user input, but they differ in the way they are sent and how it is handled on the server side.

In PHP, cookies, and session management are essential features for handling user data and maintaining the state between different requests. Tracking the users can be done using cookies and sessions in PHP.

In PHP, a cookie is a small text file that is stored on the user's (client) computer. The information in the file consists of the name of the cookie and a textual value. A cookie is created by some software on the server and is sent as part of the HTTP response header to the client where it is stored. Only the server that created a cookie can receive the cookie from the browser, so a particular cookie is information that is exchanged exclusively between one specific browser and one specific server. Every browser request includes all of the cookies its host machine has stored that are associated with the web server to which the request is directed.

Sessions in PHP provide a way to preserve data across multiple pages during a user's visit to a website. Unlike cookies, which are stored on the user's computer, sessions store data on the server. Sessions are commonly used to maintain user authentication, store user preferences, and manage other temporary data related to a user's interaction with a website. Rather than using one or more cookies to store information on a client computer a single session array can be used to store information about a particular session on the server. A session array stores a unique session ID for a particular session. A significant difference between cookies and sessions is, that cookies are stored on the client's computer whereas session information is stored on the server.

Creating a simple database application using PHP and MySQL involves several steps, setting up a database, creating an XHTML form for collecting user information, creating a PHP script that collects form data and establishes a connection to a MySQL database, and then performing CRUD (Create, Read, Update, Delete) operations, and displaying the retrieved data on the browser. There are various methods/functions used in PHP to handle/access MySQL database. They are - `mysqli_connect()`, `die()`, `mysqli_close()`, `mysqli_select_db()`, `mysqli_num_row()`, `mysqli_num_fields()`, `mysqli_fetch_array()`, `mysqli_fetch_row()`, etc.

## 8.11 Keywords

- Form Handling
- GET and POST
- Cookies
- Sessions
- Session ID
- `$_COOKIE[]`
- `$_SESSION[]`
- MySQL
- CRUD (Create, Read, Update, and Delete)

## 8.12 Recommended resources for further reading

### a. Essential Reading

1. Sebesta, R. W. (2010). Programming the World Wide Web (6th ed.), Pearson education.
2. Subramanian, V. (2019). Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node (2nd Ed.), Apress.

#### b. Recommended Reading

1. DT Editorial Services. (2016). HTML 5: Covers CSS3, JavaScript, XML, XHTML, AJAX, PHP & jQuery: Black Book, Dreamtech Press.
2. Koroliova, E. W. I., (2018). MERN Quick Start Guide: Build Web applications with MongoDB, Express.js, React and Node, Packt.

--\*--