Chapter 3: Creating Dynamic Web Sites

PHP Form Handling

The PHP superglobals $_GET and $_POST are used to collect form-data.

The example below displays a simple HTML form with two input fields and a submit button:

Ex:

```
<html>
   <body>
      <form action="welcome.php" method="post">
      Name: <input type="text" name="name"><br>
      E-mail: <input type="text" name="email"><br>
      <input type="submit">
      </form>
   </body>
</html>
```

- When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php". The form data is sent with the HTTP POST method.
- To display the submitted data you could simply echo all the variables. The "welcome.php" looks like this:

```
<html>
   <body>
      Welcome <?php echo $_POST["name"]; ?><br>
      Your email address is: <?php echo $_POST["email"]; ?>
   </body>
</html>
```

- The same result could also be achieved using the HTTP GET method:

```
<html>
   <body>
        <form action="welcome_get.php" method="get">
        Name: <input type="text" name="name"><br>
        E-mail: <input type="text" name="email"><br>
        <input type="submit">
        </form>
   </body>
</html>
```

and "welcome_get.php" looks like this:

```
<html>
   <body>
      Welcome <?php echo $_GET["name"]; ?><br>
      Your email address is: <?php echo $_GET["email"]; ?>
   </body>
</html>
```

GET vs. POST

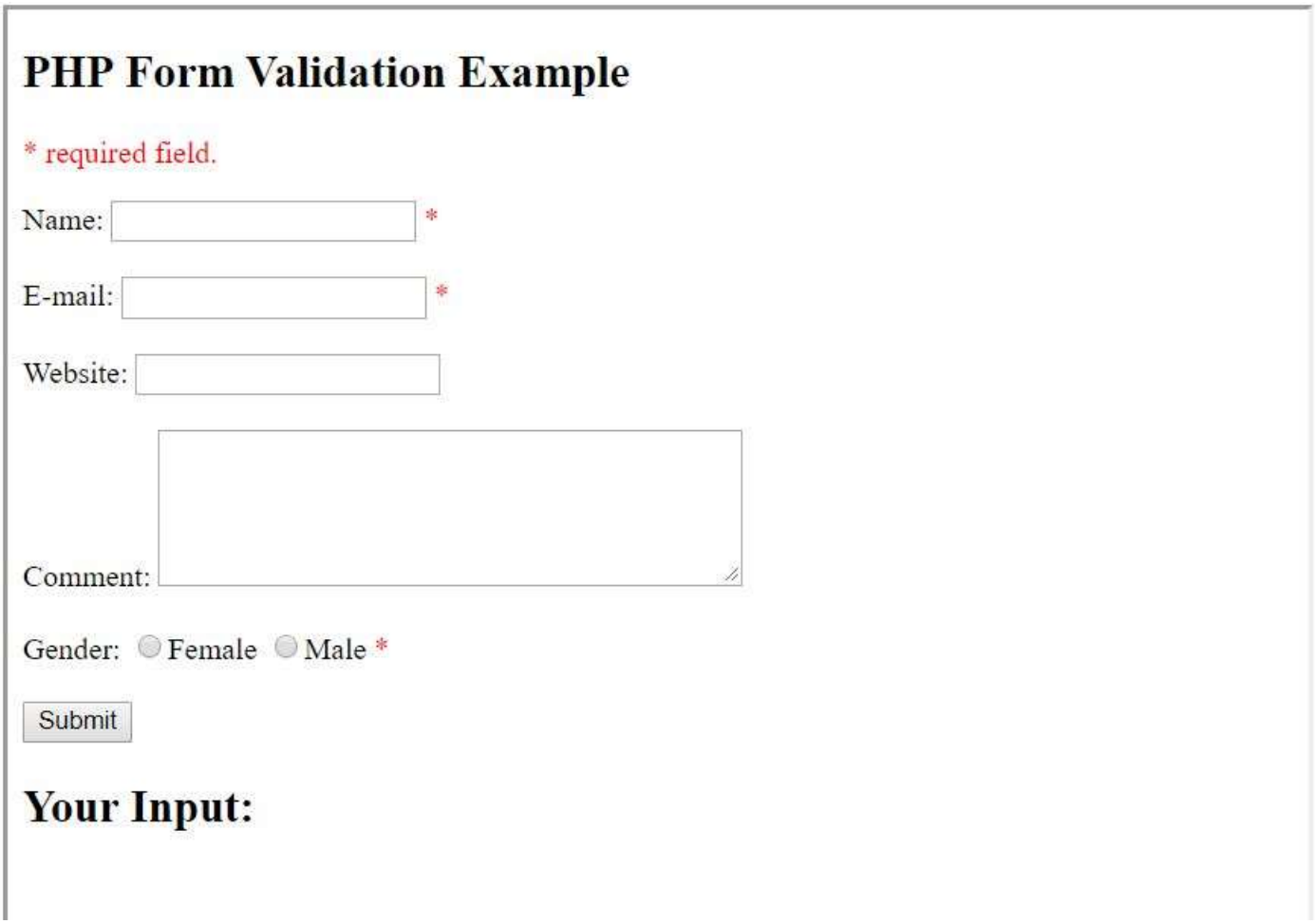
$_GET is an array of variables passed to the current script via the URL parameters.
$_POST is an array of variables passed to the current script via the HTTP POST method.

- **When to use GET?**
  - Information sent from a form with the GET method is **visible to everyone** (all variable names and values are displayed in the URL).
  - GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.
  - GET may be used for sending non-sensitive data.
- **When to use POST?**
  - Information sent from a form with the POST method is **invisible to others** (all names/values are embedded within the body of the HTTP request) and has **no limits** on the amount of information to send.
  - the variables are not displayed in the URL, it is not possible to bookmark the page.

Form Validation

The HTML form we will be working at in these chapters, contains various input fields: required and optional text fields, radio buttons, and a submit button:

Think SECURITY when processing PHP forms!



The validation rules for the form above are as follows:

Field	Validation Rules
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one

➢ Text Fields

The name, email, and website fields are text input elements, and the comment field is a textarea. The HTML code looks like this:

```
Name: <input type="text" name="name">
E-mail: <input type="text" name="email">
Website: <input type="text" name="website">
Comment: <textarea name="comment" rows="5" cols="40"></textarea>
```

➢ Radio Buttons

The gender fields are radio buttons and the HTML code looks like this:

```
Gender:
<input type="radio" name="gender" value="female">Female
<input type="radio" name="gender" value="male">Male
```

➢ The Form Element

The HTML code of the form looks like this:

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

- **What is the $_SERVER["PHP_SELF"] variable?**
  The $_SERVER["PHP_SELF"] is a super global variable that returns the filename of the currently executing script.
  So, the $_SERVER["PHP_SELF"] sends the submitted form data to the page itself, instead of jumping to a different page. This way, the user will get error messages on the same page as the form.
- **What is the htmlspecialchars() function?**
  The htmlspecialchars() function converts special characters to HTML entities. This means that it will replace HTML characters like < and > with < and >. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms

➢ Validate Form Data With PHP

- The first thing we will do is to pass all variables through PHP's htmlspecialchars() function.

  When we use the htmlspecialchars() function; then if a user tries to submit the following in a text field:

  ```
  <script>location.href('http://www.hacked.com')</script>
  ```

  This would not be executed, because it would be saved as HTML escaped code, like this:

  ```
  &lt;script&gt;location.href('http://www.hacked.com')&lt;/script&gt;
  ```

  The code is now safe to be displayed on a page or inside an e-mail.

- We will also do two more things when the user submits the form:
  1. Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP **trim()** function).
  2. Remove backslashes (\) from the user input data (with the PHP **stripslashes()** function).

  Ex:

```php
<?php
// define variables and set to empty values
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $name = test_input($_POST["name"]);
  $email = test_input($_POST["email"]);
  $website = test_input($_POST["website"]);
  $comment = test_input($_POST["comment"]);
  $gender = test_input($_POST["gender"]);
}

function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>
```

Required Fields

In the following code we have added some new variables: $nameErr, $emailErr, $genderErr, and $websiteErr. These error variables will hold error messages for the required fields. We have also added an if else statement for each $_POST variable. This checks if the $_POST variable is empty (with the PHP empty() function). If it is empty, an error message is stored in the different error variables, and if it is not empty, it sends the user input data through the test_input() function:

```
<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
  }
  if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
  }
  if (empty($_POST["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]);
  }
  if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }
  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
    $gender = test_input($_POST["gender"]);
  }
}
?>
```

Display The Error Messages

Then in the HTML form, we add a little script after each required field, which generates the correct error message if needed (that is if the user tries to submit the form without filling out the required fields):

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
Name: <input type="text" name="name">
<span class="error">* <?php echo $nameErr;?></span>
<br><br>
E-mail:
<input type="text" name="email">
<span class="error">* <?php echo $emailErr;?></span>
<br><br>
Website:
<input type="text" name="website">
<span class="error"><?php echo $websiteErr;?></span>
<br><br>
Comment: <textarea name="comment" rows="5" cols="40"></textarea>
<br><br>
Gender:
<input type="radio" name="gender" value="female">Female
<input type="radio" name="gender" value="male">Male
<span class="error">* <?php echo $genderErr;?></span>
<br><br>
<input type="submit" name="submit" value="Submit">
</form>
```

Forms - Validate Name , E-mail and URL

- **Validate Name**

  The code below shows a simple way to check if the name field only contains letters and whitespace. If the value of the name field is not valid, then store an error message:

```
$name = test_input($_POST["name"]);
if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
  $nameErr = "Only letters and white space allowed";
}
```

  Note: The **preg_match()** function searches a string for pattern, returning true if the pattern exists, and false otherwise.

- **Validate E-mail**

  The easiest and safest way to check whether an email address is well-formed is to use PHP's **filter_var()** function.

  In the code below, if the e-mail address is not well-formed, then store an error message:

```
$email = test_input($_POST["email"]);
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
  $emailErr = "Invalid email format";
}
```

- **Validate URL**

  The code below shows a way to check if a URL address syntax is valid (this regular expression also allows dashes in the URL). If the URL address syntax is not valid, then store an error message:

```
$website = test_input($_POST["website"]);
if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-
9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$website)) {
  $websiteErr = "Invalid URL";
}
```

Keep the Values in The Form

To show the values in the input fields after the user hits the submit button, we add a little PHP script inside the value attribute of the following input fields: name, email, and website. In the comment textarea field, we put the script between the <textarea> and </textarea> tags. The little script outputs the value of the $name, $email, $website, and $comment variables.

Then, we also need to show which radio button that was checked. For this, we must manipulate the checked attribute (not the value attribute for radio buttons):

```
Name: <input type="text" name="name" value="<?php echo $name;?>">
E-mail: <input type="text" name="email" value="<?php echo $email;?>">
Website: <input type="text" name="website" value="<?php echo $website;?>">
Comment: <textarea name="comment" rows="5" cols="40"><?php echo
$comment;?></textarea>
Gender:
<input type="radio" name="gender"
<?php if (isset($gender) && $gender=="female") echo "checked";?>
value="female">Female
<input type="radio" name="gender"
<?php if (isset($gender) && $gender=="male") echo "checked";?>
value="male">Male
```

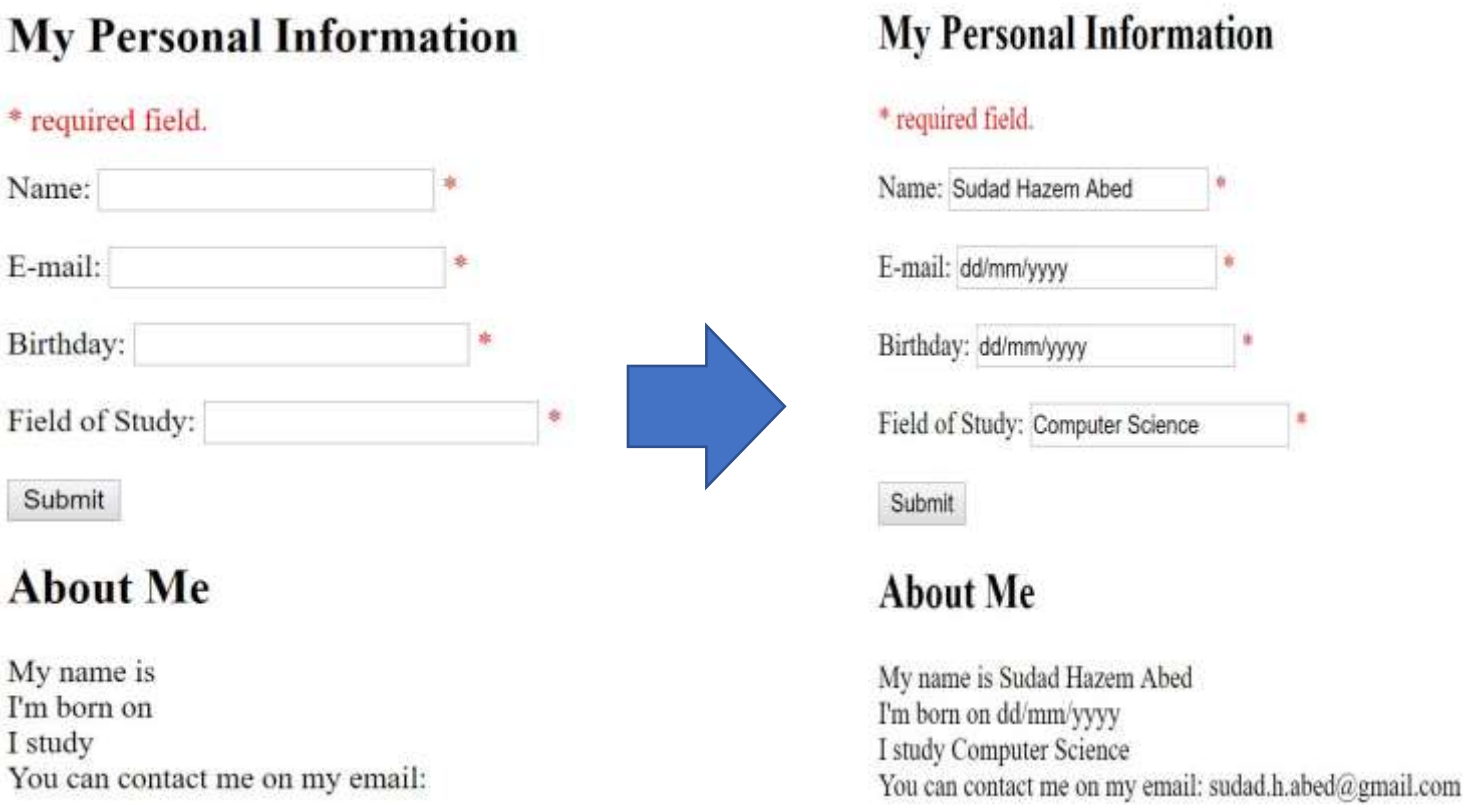
Now, the script looks like this:

```
<!DOCTYPE HTML>
<html>
<head>
<style>
.error {color: #FF0000;}
</style>
</head>
<body>

<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
    // check if name only contains letters and whitespace
    if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
      $nameErr = "Only letters and white space allowed";
    }
  }

  if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
    // check if e-mail address is well-formed
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
      $emailErr = "Invalid email format";
    }
  }
  if (empty($_POST["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]);
    // check if URL address syntax is valid (this regular expression also
allows dashes in the URL)
    if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-
9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$website)) {
      $websiteErr = "Invalid URL";
    }
  }
```

```
  if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }

  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
    $gender = test_input($_POST["gender"]);
  }
}

function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>

<h2>PHP Form Validation Example</h2>
<p><span class="error">* required field.</span></p>
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?
>">
  Name: <input type="text" name="name" value="<?php echo $name;?>">
  <span class="error">* <?php echo $nameErr;?></span>
  <br><br>
  E-mail: <input type="text" name="email" value="<?php echo $email;?>">
  <span class="error">* <?php echo $emailErr;?></span>
  <br><br>
  Website: <input type="text" name="website" value="<?php echo $website;?>">
  <span class="error">><?php echo $websiteErr;?></span>
  <br><br>
  Comment: <textarea name="comment" rows="5" cols="40"><?php echo $comment;?></
textarea>
  <br><br>
  Gender:
  <input type="radio" name="gender" <?php if (isset($gender) &&
$gender=="female") echo"checked";?> value="female">Female
  <input type="radio" name="gender" <?php if (isset($gender) &&
$gender=="male") echo"checked";?> value="male">Male
  <span class="error">* <?php echo $genderErr;?></span>
  <br><br>
  <input type="submit" name="submit" value="Submit">
</form>
```

```
<?php
echo "<h2>Your Input:</h2>";
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;
echo "<br>";
echo $gender;
?>
</body>
</html>
```

Assignment 2: Write the needed HTML code to make a form that has the fields presented below, then use PHP to collect the values and print them as shown below:



Chapter 4: Introduction to MySQL

What is MySQL?

- MySQL is the most popular database system used with PHP.
- MySQL is a database system used on the web.
- MySQL is a database system that runs on a server.
- MySQL is ideal for both small and large applications.
- MySQL is very fast, reliable, and easy to use.
- MySQL uses standard SQL.
- MySQL compiles on a number of platforms.
- MySQL is free to download and use.
- MySQL is the de-facto standard database system for web sites with HUGE volumes of both data and end-users (like Facebook, Twitter, and Wikipedia).

Databases are useful for storing information categorically. For example, A company may have a database with the following tables:

- Employees
- Products
- Customers
- Orders

Specific Text Types

- CHAR
- VARCHAR
- TINYTEXT
- TEXT
- MEDIUMTEXT
- LONGTEXT

Specific Numeric Types

- TINYINT
- SMALLINT
- MEDIUMINT
- INT
- BIGINT
- FLOAT
- DOUBLE
- DECIMAL

Specific Date and Time Types

- DATE
- DATETIME
- TIMESTAMP
- TIME

CHAR	VARCHAR
• Fixed length • Generally, requires more disk space • Generally faster • Best used for values that will always be a fixed length	• Variable length • Generally, requires less disk space • Generally slower • Best used for values that will be of any length

Important Column Properties

- **Length**
- **NULL/NOT NULL** → Best to use NOT NULL as much as possible.
- **DEFAULT** →The DEFAULT *value* clause in a data type specification indicates a default value for a column.
- **UNSIGNED** → Best to use UNSIGNED whenever appropriate for numbers.
- **ZEROFILL → declared as INT(4) ZEROFILL, a value of 5 is retrieved as 0005.**
- **AUTO_INCREMENT→** AUTO_INCREMENT is used with primary key columns.

Primary Key

- Must always have a value.
- The value must never change.
- The value must be unique for each record in the table.
- Almost always unsigned, not null integers.
- Use AUTO_INCREMENT to set the value.

To Create a Storage in Database

1- Determine the database's name.
2. Determine the table names.
3. Determine the column names for each table.
4. Determine types for columns.
5. Define Index, Restrictions etc.

PHP Connect to MySQL

PHP 5 and later can work with a MySQL database using:

- **MySQLi extension** (the "i" stands for improved)
- **PDO (PHP Data Objects)**

Earlier versions of PHP used the MySQL extension. However, this extension was deprecated in 2012.

➢ Open a Connection to MySQL

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
// Create connection
$conn = new mysqli($servername, $username, $password);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";
?>
```

PHP Create a MySQL Database

A database consists of one or more tables.

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
// Create connection
$conn = new mysqli($servername, $username, $password);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
// Create database
$sql = "CREATE DATABASE myDB";
if ($conn->query($sql) === TRUE) {
    echo "Database created successfully";
} else {
    echo "Error creating database: " . $conn->error;
}

$conn->close();
?>
```

PHP Create MySQL Tables

A database table has its own unique name and consists of columns and rows.

- We will create a table named "MyGuests", with five columns: "id", "firstname", "lastname", "email" and "reg_date":

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// sql to create table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP
)";

if ($conn->query($sql) === TRUE) {
    echo "Table MyGuests created successfully";
} else {
    echo "Error creating table: " . $conn->error;
}

$conn->close();
?>
```

PHP Insert Data Into MySQL

After a database and a table have been created, we can start adding data in them.
Here are some syntax rules to follow:

- The SQL query must be quoted in PHP
- String values inside the SQL query must be quoted
- Numeric values must not be quoted
- The word NULL must not be quoted

The INSERT INTO statement is used to add new records to a MySQL table:

```
INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)
```

Ex:

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if ($conn->query($sql) === TRUE) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}
$conn->close();
?>
```

PHP Get ID of Last Inserted Record

If we perform an INSERT or UPDATE on a table with an AUTO_INCREMENT field, we can get the ID of the last inserted/updated record immediately.

In the table "MyGuests", the "id" column is an AUTO_INCREMENT field:

Ex:

```php
$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if ($conn->query($sql) === TRUE) {
    $last_id = $conn->insert_id;
    echo "New record created successfully. Last inserted ID is: " .
$last_id;
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}
```

PHP Insert Multiple Records Into MySQL

Multiple SQL statements must be executed with the mysqli_multi_query() function.

The following examples add three new records to the "MyGuests" table:

```
$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Mary', 'Moe', 'mary@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Julie', 'Dooley', 'julie@example.com')";

if ($conn->multi_query($sql) === TRUE) {
    echo "New records created successfully";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}
```

PHP Prepared Statements

Prepared statements are very useful against SQL injections.

Prepared statements basically work like this:

1. Prepare: An SQL statement template is created and sent to the database. Certain values are left unspecified, called parameters (labeled "?"). Example: INSERT INTO MyGuests VALUES(?, ?, ?).
2. The database parses, compiles, and performs query optimization on the SQL statement template, and stores the result without executing it.
3. Execute: At a later time, the application binds the values to the parameters, and the database executes the statement. The application may execute the statement as many times as it wants with different values.

Ex:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
```

```
}

// prepare and bind
$stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email)
VALUES (?, ?, ?)");
$stmt->bind_param("sss", $firstname, $lastname, $email);

// set parameters and execute
$firstname = "John";
$lastname = "Doe";
$email = "john@example.com";
$stmt->execute();

$firstname = "Mary";
$lastname = "Moe";
$email = "mary@example.com";
$stmt->execute();

$firstname = "Julie";
$lastname = "Dooley";
$email = "julie@example.com";
$stmt->execute();

echo "New records created successfully";

$stmt->close();
$conn->close();
?>
```

- The argument may be one of four types:

- i - integer
- d - double
- s - string
- b - BLOB

- We must have one of these for each parameter.
  By telling mysql what type of data to expect, we minimize the risk of SQL injections.

Chapter 5: MySQL

❖ PHP Select Data From MySQL

➢ Select Data From a MySQL Database

The SELECT statement is used to select data from one or more tables:

```
SELECT column_name(s) FROM table_name
```

or we can use the * character to select ALL columns from a table:

```
SELECT * FROM table_name
```

The following example selects the id, firstname and lastname columns from the MyGuests table and displays it on the page:

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
    // output data of each row
    while($row = $result->fetch_assoc()) {
        echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. "
" . $row["lastname"]. "<br>";
    }
} else {
    echo "0 results";
}
$conn->close();
?>
```

In the Example above,

- First, we set up an SQL query that selects the id, firstname and lastname columns from the MyGuests table. The next line of code runs the query and puts the resulting data into a variable called $result.
- Then, the function num_rows() checks if there are more than zero rows returned.
- If there are more than zero rows returned, the function fetch_assoc() puts all the results into an associative array that we can loop through. The while() loop loops through the result set and outputs the data from the id, firstname and lastname columns.

❖ PHP Select Data From MySQL

The DELETE statement is used to delete records from a table:

```
DELETE FROM table_name WHERE some_column = some_value
```

Note: Notice the WHERE clause in the DELETE syntax: The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

If we have the "MyGuests" table as follow:

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Moe	mary@example.com	2014-10-23 10:22:30
3	Julie	Dooley	julie@example.com	2014-10-26 10:48:23

The following examples delete the record with id=3 in the "MyGuests" table

```
// sql to delete a record
$sql = "DELETE FROM MyGuests WHERE id=3";

if ($conn->query($sql) === TRUE) {
    echo "Record deleted successfully";
} else {
    echo "Error deleting record: " . $conn->error;
}
```

❖ PHP Update Data in MySQL

The UPDATE statement is used to update existing records in a table:

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

If we have the "MyGuests" table as follow:

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Moe	mary@example.com	2014-10-23 10:22:30

The following examples update the record with id=2 in the "MyGuests" table:

```
$sql = "UPDATE MyGuests SET lastname='Doe' WHERE id=2";

if ($conn->query($sql) === TRUE) {
    echo "Record updated successfully";
} else {
    echo "Error updating record: " . $conn->error;
}
```

Note: Notice the WHERE clause in the UPDATE syntax: The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

❖ PHP Limit Data Selections From MySQL

- MySQL provides a LIMIT clause that is used to specify the number of records to return.
- The LIMIT clause makes it easy to code multi page results or pagination with SQL, and is very useful on large tables. Returning a large number of records can impact on performance.
- Assume we wish to select all records from 1 - 30 (inclusive) from a table called "Orders". The SQL query would then look like this:

```
$sql = "SELECT * FROM Orders LIMIT 30";
```

- What if we want to select records 16 - 25 (inclusive), Mysql also provides a way to handle this: by using **OFFSET**.
- The SQL query below says "return only 10 records, start on record 16 (OFFSET 15)":

```
$sql = "SELECT * FROM Orders LIMIT 10 OFFSET 15";
```

You could also use a shorter syntax to achieve the same result:

```
$sql = "SELECT * FROM Orders LIMIT 15, 10";
```

Notice that the numbers are reversed when you use a comma.

Chapter 6: Advanced PHP

❖ PHP Date and Time

The PHP date() function is used to format a date and/or a time.

Syntax

```
date(format,timestamp)
```

Parameter	Description
format	Required. Specifies the format of the timestamp
timestamp	Optional. Specifies a timestamp. Default is the current date and time

Note: A timestamp is a sequence of characters, denoting the date and/or time at which a certain event occurred.

➢ Get a Simple Date

Here are some characters that are commonly used for dates:

- d - Represents the day of the month (01 to 31)
- m - Represents a month (01 to 12)
- Y - Represents a year (in four digits)
- l (lowercase 'L') - Represents the day of the week

Other characters, like"/", ".", or "-" can also be inserted between the characters to add additional formatting.

Ex:

```php
<?php
echo "Today is " . date("Y/m/d") . "<br>";
echo "Today is " . date("Y.m.d") . "<br>";
echo "Today is " . date("Y-m-d") . "<br>";
echo "Today is " . date("l");
?>
```

➢ Automatic Copyright Year

The date() function can be used to automatically update the copyright year on your website:

Ex:

```
&copy; 2010-<?php echo date("Y");?>
```

➢ Get a Simple Time

Here are some characters that are commonly used for times:

- h - 12-hour format of an hour with leading zeros (01 to 12)
- i - Minutes with leading zeros (00 to 59)
- s - Seconds with leading zeros (00 to 59)
- a - Lowercase Ante meridiem and Post meridiem (am or pm)

Ex:

```
<?php
echo "The time is " . date("h:i:sa");
?>
```

Note: the PHP date() function will return the current date/time of the server!

➢ Get Your Time Zone

If the time you got back from the code is not the right time, it's probably because your server is in another country or set up for a different timezone.

So, if you need the time to be correct according to a specific location, you can set a timezone to use.

The example below sets the timezone to "America/New_York", then outputs the current time in the specified format:

Ex:

```
<?php
date_default_timezone_set("America/New_York");
echo "The time is " . date("h:i:sa");
?>
```

❖ PHP Include Files

- ❖ The include (or require) statement takes all the text/code/markup that exists in the specified file and copies it into the file that uses the include statement.
- ❖ Including files is very useful when you want to include the same PHP, HTML, or text on multiple pages of a website.

➢ include and require Statements

The include and require statements are identical, except upon failure:

- require will produce a fatal error (E_COMPILE_ERROR) and stop the script
- include will only produce a warning (E_WARNING) and the script will continue

Syntax

```
include 'filename';

or

require 'filename';
```

Ex: Assume we have a standard footer file called "footer.php", that looks like this:

```
<?php
echo "<p>Copyright &copy; 1999-" . date("Y") . " W3Schools.com</p>";
?>
```

To include the footer file in a page, use the include statement:

```
<html>
<body>

<h1>Welcome to my home page!</h1>
<p>Some text.</p>
<p>Some more text.</p>
<?php include 'footer.php';?>

</body>
</html>
```

Notes:

- ✓ Use require when the file is required by the application.
- ✓ Use include when the file is not required and application should continue when file is not found.

❖ PHP File Upload

With PHP, it is easy to upload files to the server. However, with ease comes danger, so always be careful when allowing file uploads!

➢ Configure The "php.ini" File

- ➢ First, ensure that PHP is configured to allow file uploads.
- ➢ In your "php.ini" file, search for the file_uploads directive, and set it to On:

```
file_uploads = On
```

➢ Create The HTML Form

Create an HTML form that allow users to choose the image file they want to upload:

```
<!DOCTYPE html>
<html>
<body>

<form action="upload.php" method="post" enctype="multipart/form-data">
    Select image to upload:
    <input type="file" name="fileToUpload" id="fileToUpload">
    <input type="submit" value="Upload Image" name="submit">
</form>

</body>
</html>
```

Some rules to follow for the HTML form above:

- Make sure that the form uses method="post"
- The form also needs the following attribute: enctype="multipart/form-data". It specifies which content-type to use when submitting the form

Without the requirements above, the file upload will not work.

Other things to notice:

- The type="file" attribute of the <input> tag shows the input field as a file-select control, with a "Browse" button next to the input control

The form above sends data to a file called "upload.php", which we will create next

The Upload File PHP Script

The "upload.php" file contains the code for uploading a file:

```
<?php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));
// Check if image file is a actual image or fake image
if(isset($_POST["submit"])) {
    $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
    if($check !== false) {
        echo "File is an image - " . $check["mime"] . ".";
        $uploadOk = 1;
    } else {
        echo "File is not an image.";
        $uploadOk = 0;
    }
}
?>
```

PHP script explained:

- $target_dir = "uploads/" - specifies the directory where the file is going to be placed
- $target_file specifies the path of the file to be uploaded
- $uploadOk=1 is not used yet (will be used later)
- $imageFileType holds the file extension of the file (in lower case)
- Next, check if the image file is an actual image or a fake image

Note: You will need to create a new directory called "uploads" in the directory where "upload.php" file resides. The uploaded files will be saved there.

Check if File Already Exists

Check if the file already exists in the "uploads" folder. If it does, an error message is displayed, and $uploadOk is set to 0:

```
// Check if file already exists
if (file_exists($target_file)) {
    echo "Sorry, file already exists.";
    $uploadOk = 0;
}
```

Limit File Size

Check the size of the file. If the file is larger than 500KB, an error message is displayed, and $uploadOk is set to 0:

```
// Check file size
if ($_FILES["fileToUpload"]["size"] > 500000) {
    echo "Sorry, your file is too large.";
    $uploadOk = 0;
}
```

Limit File Type

The code below only allows users to upload JPG, JPEG, PNG, and GIF files. All other file types gives an error message before setting $uploadOk to 0:

```
// Allow certain file formats
if($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType !=
"jpeg"
&& $imageFileType != "gif" ) {
    echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";
    $uploadOk = 0;
}
```

Complete Upload File PHP Script

```php
<?php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));
// Check if image file is a actual image or fake image
if(isset($_POST["submit"])) {
    $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
    if($check !== false) {
        echo "File is an image - " . $check["mime"] . ".";
        $uploadOk = 1;
    } else {
        echo "File is not an image.";
        $uploadOk = 0;
    }
}
// Check if file already exists
if (file_exists($target_file)) {
    echo "Sorry, file already exists.";
    $uploadOk = 0;
}
// Check file size
if ($_FILES["fileToUpload"]["size"] > 500000) {
    echo "Sorry, your file is too large.";
    $uploadOk = 0;
}
// Allow certain file formats
if($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType != "jpeg"
&& $imageFileType != "gif" ) {
    echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";
    $uploadOk = 0;
}
// Check if $uploadOk is set to 0 by an error
if ($uploadOk == 0) {
    echo "Sorry, your file was not uploaded.";
// if everything is ok, try to upload file
} else {
    if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file)) {
        echo "The file ". basename( $_FILES["fileToUpload"]["name"]). " has been
uploaded.";
    } else {
        echo "Sorry, there was an error uploading your file.";
    }
}
?
```

Chapter 7: Advanced PHP

❖ PHP Cookies

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

A cookie is created with the setcookie() function.

Syntax

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

Only the *name* parameter is required. All other parameters are optional.

➢ Create/Retrieve Cookies With PHP

The following example creates a cookie named "user" with the value "John Doe". The cookie will expire after 30 days (86400 * 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer).

We then retrieve the value of the cookie "user" (using the global variable $_COOKIE). We also use the isset() function to find out if the cookie is set:

Ex:

```
<?php
$cookie_name = "user";
$cookie_value = "John Doe";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400
= 1 day
?>
<html>
<body>
<?php
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
}
?>
</body>
</html>
```

Note: The setcookie() function must appear BEFORE the <html> tag.

Note: The value of the cookie is automatically URLencoded when sending the cookie, and automatically decoded when received (to prevent URLencoding, use setrawcookie() instead).

➢ Modify a Cookie Value

To modify a cookie, just set (again) the cookie using the setcookie() function:

➢ Delete a Cookie

To delete a cookie, use the setcookie() function with an expiration date in the past:

Ex:

```
?php
// set the expiration date to one hour ago
setcookie("user", "", time() - 3600);
?>
<html>
<body>

<?php
echo "Cookie 'user' is deleted.";
?>

</body>
</html>
```

❖ PHP Sessions

- A session is a way to store information (in variables) to be used across multiple pages.
- Unlike a cookie, the information is not stored on the users computer.

➢ What is a PHP Session?

When you work with an application, you open it, do some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet, there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.

Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.

So; Session variables hold information about one single user and are available to all pages in one application.

➢ Start a PHP Session

- A session is started with the session_start() function.
- Session variables are set with the PHP global variable: $_SESSION.

Ex: In this example we create a new page called "demo_session1.php". In this page, we start a new PHP session and set some session variables:

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>
</body>
</html>
```

Note: The session_start() function must be the very first thing in your document. Before any HTML tags.

➢ Get PHP Session Variable Values

we create another page called "demo_session2.php". From this page, we will access the session information we set on the first page ("demo_session1.php").

Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (session_start()).

Ex:

```
<?php session_start(); ?>
<!DOCTYPE html>
<html>
<body>
<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . ".<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>
</body>
</html>
```

Another way to show all the session variable values for a user session is to run the following code:

Ex:

```
<?php session_start(); ?>
<!DOCTYPE html>
<html>
<body>
<?php print_r($_SESSION); ?>
</body>
</html>
```

How does it work? How does it know it's me?

Most sessions set a user-key on the user's computer that looks something like this: 765487cf34ert8dede5a562e4f3a7e12. Then, when a session is opened on another page, it scans the computer for a user-key. If there is a match, it accesses that session, if not, it starts a new session.

- **Modify a PHP Session Variable**

  To change a session variable, just overwrite it:

- **Destroy a PHP Session**

  To remove all global session variables and destroy the session, use session_unset() and session_destroy():

Ex:

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// remove all session variables
session_unset();

// destroy the session
session_destroy();
?>

</body>
</html>
```

❖ PHP Filters

- **Validating data**: Determine if the data is in proper form.
- **Sanitizing data**: Remove any illegal character from the data.

➢ The PHP Filter Extension

- PHP filters are used to validate and sanitize external input.
- The PHP filter extension has many of the functions needed for checking user input, and is designed to make data validation easier and quicker.
- The filter_list() function can be used to list what the PHP filter extension offers:

```
<!DOCTYPE html>
<html>
<head>
<style>
table, th, td {
    border: 1px solid black;
    border-collapse: collapse;
}
th, td {
    padding: 5px;
}
</style>
</head>
<body>

<table>
  <tr>
    <td>Filter Name</td>
    <td>Filter ID</td>
  </tr>
  <?php
  foreach (filter_list() as $id =>$filter) {
      echo '<tr><td>' . $filter . '</td><td>' .
                  filter_id($filter) . '</td></tr>';
  }
  ?>
</table>

</body>
</html>
```

Filter Name	Filter ID
int	257
boolean	258
float	259
validate_regexp	272
validate_url	273
validate_email	274
validate_ip	275
string	513
stripped	513
encoded	514
special_chars	515
full_special_chars	522
unsafe_raw	516
email	517
url	518
number_int	519
number_float	520
magic_quotes	521
callback	1024

➢ Why Use Filters?

Many web applications receive external input. External input/data can be:

- User input from a form
- Cookies
- Web services data
- Server variables
- Database query results

You should always validate external data!

Invalid submitted data can lead to security problems and break your webpage. By using PHP filters you can be sure your application gets the correct input.

➢ PHP filter_var() Function

The filter_var() function both validate and sanitize data.

The filter_var() function filters a single variable with a specified filter. It takes two pieces of data:

- The variable you want to check
- The type of check to use

Sanitize a String

Ex: The following example uses the filter_var() function to remove all HTML tags from a string:

```
<?php
$str = "<h1>Hello World!</h1>";
$newstr = filter_var($str, FILTER_SANITIZE_STRING);
echo $newstr;
?>
```

Validate an Integer

Ex: The following example uses the filter_var() function to check if the variable $int is an integer. If $int is an integer, the output of the code below will be: "Integer is valid". If $int is not an integer, the output will be: "Integer is not valid":

```
<?php
$int = 100;
if (!filter_var($int, FILTER_VALIDATE_INT) === false) {
    echo("Integer is valid");
} else {
    echo("Integer is not valid");
}
?>
```

Tip: filter_var() and Problem With 0

In the example above, if $int was set to 0, the function above will return "Integer is not valid". To solve this problem, use the code below:

```
<?php
$int = 0;
if (filter_var($int, FILTER_VALIDATE_INT) === 0 || !filter_var($int,
FILTER_VALIDATE_INT) === false) {
    echo("Integer is valid");
} else {
    echo("Integer is not valid");
}
?>
```

Validate an IP Address

Ex: The following example uses the filter_var() function to check if the variable $ip is a valid IP address:

```
?php
$ip = "127.0.0.1";

if (!filter_var($ip, FILTER_VALIDATE_IP) === false) {
    echo("$ip is a valid IP address");
} else {
    echo("$ip is not a valid IP address");
}
?>
```