Open Source Software Development

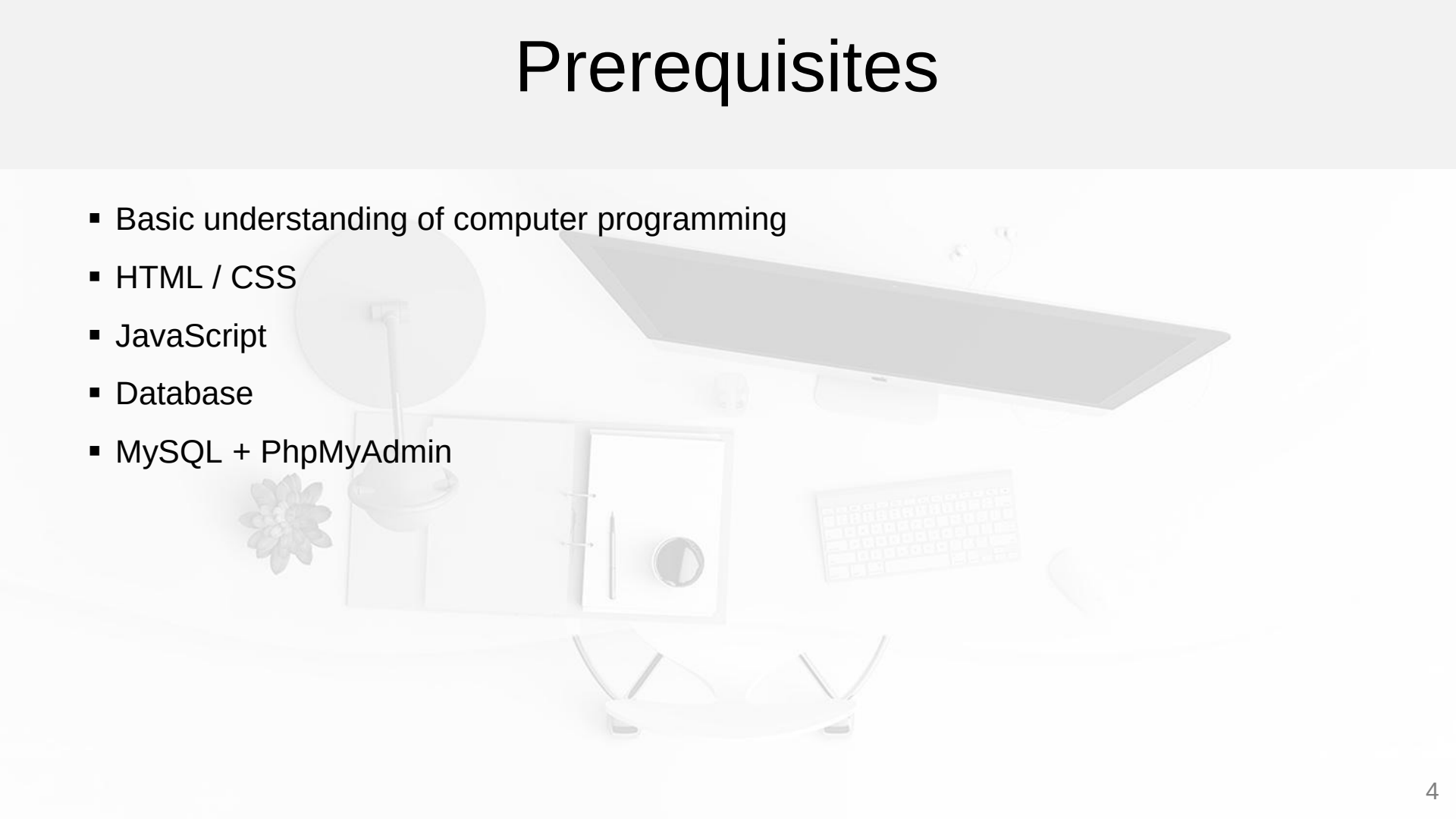
Ung Văn Giàu
Email: giau.ung@eiu.edu.vn

PHP: Hypertext Preprocessor

Contents

Prerequisites

- Basic understanding of computer programming
- HTML / CSS
- JavaScript
- Database
- MySQL + PhpMyAdmin



What is PHP?

- PHP is a **open source server scripting language** for making dynamic and interactive Web pages by interacting with databases
- PHP is a widely-used, **free**, and efficient alternative to competitors such as Microsoft’s ASP
- Fast, flexible and pragmatic, PHP powers everything from blogs to the most popular websites in the world

What is PHP?

- PHP is an amazing and popular language!
  - It is **powerful enough** to be at the **core of the biggest blogging system** on the web (WordPress)!
  - It is **deep enough** to **run the largest social network** (Facebook)!
  - It is also **easy enough** to be a beginner's first server-side language!

Why to Learn PHP?

- PHP runs on various platforms

  Windows, Linux, Unix, Mac OS X,…

- PHP is compatible with almost all servers

  Apache, IIS, nginx,…

- PHP is a server side scripting language embedded in HTML

  to manage dynamic content, databases, session tracking, even build entire e-commerce sites

Why to Learn PHP?

- It is integrated with a number of popular databases

  MySQL, PostgreSQL, Oracle, and Microsoft SQL Server

- PHP supports a large number of major protocols

  SMTP, POP3, IMAP, LDAP, Java and distributed object architectures (COM and CORBA)

- PHP Syntax is C-Like

What can PHP do?

- Generate dynamic page content
- Create, open, read, write, delete, and close files on the server
- Handle form data
- Access cookies variables and set cookies
- Add, delete, modify data in database
- Be used to control user-access

  Restrict users to access some pages
- Encrypt data

PHP Syntax

- A PHP script is **executed on the server**, and the **plain HTML result** is **sent back to the browser**
- A PHP script can **be placed anywhere** in the document
- A PHP script starts with **<?php** and ends with **?>**
- The default file extension for PHP files is ".php"
- A PHP file normally contains HTML tags, and some PHP scripting code

PHP Syntax

- **Note:** PHP statements end with a semicolon (**;**)

- PHP **Case Sensitivity**
  - Keywords (e.g., if, else, while, echo,…), classes, functions, and user-defined functions are **not case-sensitive**
  - All **variable names** are **case-sensitive**!

PHP Syntax

```
<!DOCTYPE html>
<html>
    <head>
        <title>PHP Example</title>
        <meta charset="utf-8">
    </head>
    <body>
        <h1>My first PHP page</h1>
        <?php
            echo "Hello World!";
        ?>
    </body>
</html>
```

PHP Comments

- A comment in PHP code is a line not executed as a part of the program

  Its only purpose is to be read by someone who is looking at the code

- PHP supports several ways of commenting:
  - Syntax for single-line comments

    // This is a single-line comment

    # This is also a single-line comment

  - Syntax for multiple-line comments

```
/*
        This is a multiple-lines comment block
        that spans over multiple lines
*/
```

PHP Variables

Declare PHP Variables

- A variable starts with the $ sign, followed by the name of the variable
  - $txt = “Hello world!”;
  - $x = 5;
  - $y = 10.5;

- When **assign a text value** to a variable, **put quotes** around the value

PHP Variables

PHP Variables

- Rules for PHP variables:
  - A variable starts with the $ sign, followed by the name of the variable
  - A variable name:
    - ✓must start with a letter or the underscore character
    - ✓cannot start with a number
    - ✓can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
  - Variable names are case-sensitive

    $age and $AGE are two different variables

PHP Variables

Output Variables

- The PHP echo statement is often used to output data to the screen

```
$txt = “W3Schools.com”;
echo “I love $txt!”;
echo “I love ” . $txt . “!”;


$x = 5;
$y = 4;
echo $x + $y;
```

PHP is a Loosely Typed Language

- PHP automatically associates a data type to the variable, depending on its value.
- Since the data types are not set in a strict sense, you can do things like adding a string to an integer without causing an error.
- In PHP 7, type declarations were added

PHP echo and print Statements

There are two basic ways to get output

- **echo** statement can be used with or without parentheses: echo or echo()

```
echo “Hello world!<br>”;
echo (“<h2>” . $txt1 . “</h2>”);
```

- **print** statement can be used with or without parentheses: print or print()

```
print “<h2>PHP is Fun!</h2>”;
print (“<h2>” . $txt1 . “</h2>”);
```

PHP Data Types

PHP supports the following data types:

- **String**
  - a sequence of characters
  - any text inside quotes (single or double)
  - E.g., $y = ‘Hello world!’;
- **Integer**
  - a non-decimal number
  - E.g., $x = 5985;

PHP Data Types

PHP supports the following data types:

- **Float** (floating point numbers - also called double)
  - a number with a decimal point or a number in exponential form
  - E.g., $x = 10.365;
- **Boolean**
  - represents two possible states: TRUE or FALSE
  - E.g.:

    $x = true;

    $y = false;

PHP Data Types

PHP supports the following data types:

- **Array**
  - stores multiple values in one single variable
  - E.g., $cars = array(“Volvo”, “BMW”, “Toyota”);
- **Object**
  - A class is a template for objects, and an object is an instance of a class
  - When the individual objects are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties

PHP Data Types

PHP supports the following data types:

- **NULL**
  - a special data type which can have only one value: NULL
  - A variable of data type NULL has no value assigned to it
  - If a variable is created without a value, it is automatically assigned a value of NULL
  - Variables can also be emptied by setting the value to NULL

PHP String Functions

- **strlen()**
  - return the length of a string
  - E.g., echo strlen(“Hello world!”);

- **str_word_count()**
  - count the number of words in a string
  - E.g., echo str_word_count(“Hello world!”);

PHP String Functions

- **strpos()**
  - searches for a specific text within a string
  - If a match is found, returns the character position of the first match
  - If no match is found, return FALSE
  - E.g., echo strpos(“Hello world!”, “world”);
- **str_replace()**
  - replaces some characters with some other characters
  - E.g., echo str_replace(“world”, “Dolly”, “Hello world!”);
- **PHP String Reference**

  https://www.w3schools.com/php/php_ref_string.asp

PHP Operators

PHP Operators

- Operators are used to **perform operations on variables and values**
- PHP divides the operators in the following groups:
  - Arithmetic operators
  - Assignment operators
  - Comparison operators
  - Increment/Decrement operators
  - Logical operators
  - String operators
  - Array operators
  - Conditional assignment operators

PHP Arithmetic Operators

The PHP arithmetic operators are **used with numeric values** to perform common arithmetical operations

Operator	Name	Example	Result
+	Addition	$x + $y	Sum of $x and $y
-	Subtraction	$x - $y	Difference of $x and $y
*	Multiplication	$x * $y	Product of $x and $y
/	Division	$x / $y	Quotient of $x and $y
%	Modulus	$x % $y	Remainder of $x divided by $y
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power

PHP Assignment Operators

Assignment	Same as...	Description
x = y	x = y	The left operand gets set to the value of the expression on the right
x += y	x = x + y	Addition
x -= y	x = x - y	Subtraction
x *= y	x = x * y	Multiplication
x /= y	x = x / y	Division
x %= y	x = x % y	Modulus

PHP Comparison Operators

The PHP comparison operators are used to compare two values (number or string)

Operator	Name	Example	Result
==	Equal	$x == $y	Returns true if $x is equal to $y
===	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type
!=	Not equal	$x != $y	Returns true if $x is not equal to $y
<>	Not equal	$x <> $y	Returns true if $x is not equal to $y
!==	Not identical	$x !== $y	Returns true if $x is not equal to $y, or they are not of the same type
>	Greater than	$x > $y	Returns true if $x is greater than $y
<	Less than	$x < $y	Returns true if $x is less than $y
>=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y
<=	Less than or equal to	$x <= $y	Returns true if $x is less than or equal to $y

PHP Increment / Decrement Operators

Operator	Name	Description
++$x	Pre-increment	Increments $x by one, then returns $x
$x++	Post-increment	Returns $x, then increments $x by one
--$x	Pre-decrement	Decrements $x by one, then returns $x
$x--	Post-decrement	Returns $x, then decrements $x by one

PHP Logical Operators

The PHP logical operators are used to combine conditional statements

Operator	Name	Example	Result
and	And	$x and $y	True if both $x and $y are true
or	Or	$x or $y	True if either $x or $y is true
xor	Xor	$x xor $y	True if either $x or $y is true, but not both
&&	And	$x && $y	True if both $x and $y are true
||	Or	$x || $y	True if either $x or $y is true
!	Not	!$x	True if $x is not true

PHP String Operators

Operator	Name	Example	Result
.	Concatenation	$txt1 . $txt2	Concatenation of $txt1 and $txt2
.=	Concatenation assignment	$txt1 .= $txt2	Appends $txt2 to $txt1

PHP Conditional Assignment Operators

The PHP conditional assignment operators are used to set a value depending on conditions

Operator	Name	Example	Result
?:	Ternary	$x = *expr1* ? *expr2* : *expr3*	Returns the value of $x. The value of $x is *expr2* if *expr1* = TRUE. The value of $x is *expr3* if *expr1* = FALSE
??	Null coalescing	$x = *expr1* ?? *expr2*	Returns the value of $x. The value of $x is *expr1* if *expr1* exists, and is not NULL. If *expr1* does not exist, or is NULL, the value of $x is *expr2*. Introduced in PHP 7

PHP if...else...elseif Statements

- if statement - executes some code if one condition is true
- if...else statement - executes some code if a condition is true and another code if that condition is false
- if...elseif...else statement - executes different codes for more than two conditions
- switch statement - selects one of many blocks of code to be executed

PHP if...else...elseif Statements

```
if (condition) {
    code to be executed if this condition is true;
} elseif (condition) {
    code to be executed if first condition is false and this condition is true;
} else {
    code to be executed if all conditions are false;
}
```

```
switch (n) {
    case label1:
        code to be executed if n=label1;
    break;
    case label2:
        code to be executed if n=label2;
    break;
    case label3:
        code to be executed if n=label3;
    break;
    ...
    default:
        code to be executed if n is different from all labels;
}
```

PHP Loops

- Loops are used to execute the same block of code again and again, as long as a certain condition is true

- We have the following loop types:
  - **while** - loops through a block of code as long as the specified condition is true
  - **do...while** - loops through a block of code once, and then repeats the loop as long as the specified condition is true
  - **for** - loops through a block of code a specified number of times
  - **foreach** - loops through a block of code for each element in an array

PHP for Loop

- The for loop - Loops through a block of code a specified number of times
- The for loop is used when you **know in advance how many times** the script should run
- **Syntax**

```
for (init counter; test counter; increment counter) {
    # code to be executed for each iteration;
}
```

- E.g.,

```
for ($x = 0; $x <= 10; $x++) {
    echo "The number is: $x <br>";
}
```

PHP foreach Loop

- Loops through a block of code for each element in an array
- The foreach loop works only on arrays, and is used to loop through each key/value pair in an array
- **Syntax**

```
foreach ($array as $value) {
    # code to be executed;
}
```

PHP foreach Loop

- E.g.,

```php
$colors = array("red", "green", "blue", "yellow");
foreach ($colors as $value) {
    echo "$value <br>";
}
```

```php
$age = array("Peter" => "35", "Ben" => "37", "Joe" => "43");
foreach($age as $x => $val) {
    echo "$x = $val<br>";
}
```

PHP Break and Continue

- **PHP Break**

  The break statement can also be used to **jump out of a loop**

- **PHP Continue**

  The continue statement **breaks one iteration** (in the loop), if a specified condition occurs, and continues with the next iteration in the loop

PHP Functions

- **PHP Built-in Functions**

  over 1000 built-in functions that can be called directly, from within a script, to perform a specific task

  E.g.:

  - isset(): determine if a variable is declared and is different than null
  - empty(): determine whether a variable is considered to be empty (if it does not exist or if its value equals false).

- **PHP User Defined Functions**
  - A function is a block of statements that can be used repeatedly in a program
  - A function will not execute automatically when a page loads
  - A function will be executed by a call to the function

Create a User Defined Function

- **Syntax**

```
function functionName() {
    # code to be executed;
}
```

- A function name must start with a letter or an underscore
- Function names are **NOT case-sensitive**
- E.g.,

```
function writeMsg() {
    echo "Hello world!";
}

writeMsg(); // call the function
```

PHP Function Arguments

- Information can be passed to functions through arguments

  An argument is just like a variable
- Arguments are specified after the function name, inside the parentheses.

  add as many arguments as you want, just separate them with a comma

```php
function familyName($fname) {
    echo "$fname Refsnes.<br>";
}

familyName("Jani");
```

PHP Arrays

- An array stores multiple values in one single variable
- Create an Array

  The array() function is used to create an array
- There are three types of arrays:
  - **Indexed arrays** - Arrays with a numeric index
  - **Associative arrays** - Arrays with named keys
  - **Multidimensional arrays** - Arrays containing one or more arrays
- Get The Length of an Array

  count() function is used to return the length (the number of elements) of an array

PHP Indexed Arrays

- There are two ways to create indexed arrays:
  - The index can be assigned automatically (index always starts at 0)

    $cars = array(“Volvo”, “BMW”, “Toyota”);

    $cars = [“Volvo”, “BMW”, “Toyota”];

  - The index can be assigned manually

    $cars[0] = “Volvo”;

    $cars[1] = “BMW”;

```php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
```

PHP Indexed Arrays

Loop Through an Indexed Array

```php
$cars = array("Volvo", "BMW", "Toyota");
$arrlength = count($cars);

for($x = 0; $x < $arrlength; $x++) {
    echo $cars[$x];
    echo "<br>";
}
```

PHP Associative Arrays

- Associative arrays are arrays that use named keys that you assign to them.
- There are two ways to create an associative array:
  - $age = array(“Peter” => “35”, “Ben” => “37”, “Joe” => “43”);

  Or

  - $age[‘Peter’] = “35”;
  - $age[‘Ben’] = “37”;
  - $age[‘Joe’] = “43”;
- The named keys can then be used in a script

```
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
echo "Peter is " . $age['Peter'] . " years old.";
```

PHP Associative Arrays

Loop Through an Associative Array

```php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

foreach($age as $x => $x_value) {
    echo "Key=" . $x . ", Value=" . $x_value;
    echo "<br>";
}
```

PHP Multidimensional Arrays

- A multidimensional array is **an array containing one or more arrays**

- The **dimension** of an array indicates the **number of indices** you need **to select** an element
  - For a two-dimensional array you need two indices to select an element
  - For a three-dimensional array you need three indices to select an element

PHP Multidimensional Arrays

Two-dimensional Arrays

```php
$cars = array (
    array("Volvo",22,18),
    array("BMW",15,13),
    array("Saab",5,2),
    array("Land Rover",17,15)
);
echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2].".<br>";
echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2].".<br>";
echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2].".<br>";
echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2].".<br>";
```

```php
for ($row = 0; $row < 4; $row++) {
    echo "<p><b>Row number $row</b></p>";
    echo "<ul>";
    for ($col = 0; $col < 3; $col++) {
        echo "<li>".$cars[$row][$col]."</li>";
    }
    echo "</ul>";
}
```

PHP Sorting Arrays

The elements in an array can be sorted in alphabetical or numerical order, descending or ascending

- **sort()** - sort arrays in ascending order
- **rsort()** - sort arrays in descending order
- **asort()** - sort associative arrays in ascending order, according to the value
- **ksort()** - sort associative arrays in ascending order, according to the key
- **arsort()** - sort associative arrays in descending order, according to the value
- **krsort()** - sort associative arrays in descending order, according to the key

PHP Sorting Arrays

```php
$cars = array("Volvo", "BMW", "Toyota");
sort($cars);

$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
ksort($age);
```

PHP Global Variables - Superglobals

- They are always accessible, regardless of scope
- The PHP superglobal variables are:
  - $GLOBALS
  - $_SERVER
  - $_REQUEST
  - $_POST
  - $_GET
  - $_FILES
  - $_ENV
  - $_COOKIE
  - $_SESSION

PHP $GLOBALS

- Be used to access global variables from anywhere in the PHP script (also from within functions or methods)
- PHP stores all global variables in an array called $GLOBALS[index]

  The index holds the name of the variable

```php
$x = 75;
$y = 25;
function addition() {
    $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
}
addition();
echo $z;
```

PHP $_SERVER

Holds information about headers, paths, and script locations

```php
echo $_SERVER['PHP_SELF'];
echo "<br>";
echo $_SERVER['SERVER_NAME'];
echo "<br>";
echo $_SERVER['HTTP_HOST'];
echo "<br>";
echo $_SERVER['HTTP_REFERER'];
echo "<br>";
echo $_SERVER['HTTP_USER_AGENT'];
echo "<br>";
echo $_SERVER['SCRIPT_NAME'];
```

Read more: https://www.w3schools.com/php/php_superglobals_server.asp

PHP $_REQUEST

Be used to collect data after submitting an HTML form

```
<!DOCTYPE html>
<html>
    <head>
    </head>
    <body>
        <form method="get" action="<?php echo $_SERVER['PHP_SELF'] ?>" >
            <input type="text" name="FullName" placeholder="Please type your full name">
            <button type="submit">Send</button>
        </form>
    <?php
        // Check if the variable is submitted
        if (isset($_REQUEST['FullName'])) {
            $fullName = $_REQUEST['FullName'];
            // Check if the value is empty
            if (empty($fullName)) {
                echo "The Name field is empty";
            } else {
                echo $fullName;
            }
        }
    ?>
    </body>
</html>
```

PHP $_POST

Be used to collect form data after submitting an HTML form with method = “post”

```
<!DOCTYPE html>
<html>
    <head>
    </head>
    <body>
        <form method="post" action="<?php echo $_SERVER['PHP_SELF'] ?>" >
            <input type="text" name="FullName" placeholder="Please type your full name">
            <button type="submit">Send</button>
        </form>
    <?php
        // Check method
        if ($_SERVER['REQUEST_METHOD'] == 'POST') {
            // Check if the variable is submitted
            if (isset($_POST['FullName'])) {
                $fullName = $_POST['FullName'];
                // Check if the value is empty
                if (empty($fullName)) {
                    echo "The Name field is empty";
                } else {
                    echo $fullName;
                }
            }
        }
    ?>
    </body>
</html>
```

PHP $_GET

- Be used to collect form data after submitting an HTML form with method = “get”
- $_GET can also collect data sent in the URL

```
<!DOCTYPE html>
<html>
    <head>
    </head>
    <body>
        <form method="get" action="<?php echo $_SERVER['PHP_SELF'] ?>" >
            <input type="text" name="FullName" placeholder="Please type your full name">
            <button type="submit">Send</button>
        </form>
    <?php
        // Check method
        if ($_SERVER['REQUEST_METHOD'] == 'GET') {
            // Check if the variable is submitted
            if (isset($_GET['FullName'])) {
                $fullName = $_GET['FullName'];
                // Check if the value is empty
                if (empty($fullName)) {
                    echo "The Name field is empty";
                } else {
                    echo $fullName;
                }
            }
        }
    ?>
    </body>
</html>
```

PHP $_GET

```
<!DOCTYPE html>
<html>
    <body>
        <a href="test_get.php?subject=PHP&web=W3schools.com">Test $GET</a>
    </body>
</html>

Page: test_get.php
<!DOCTYPE html>
<html>
    <body>
        <?php
            echo "Study " . $_GET['subject'] . " at " . $_GET['web'];
        ?>
    </body>
</html>
```

Should you use Post or Get?

- **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
  - GET may be used for sending non-sensitive data
  - **Note:** GET should NEVER be used for sending passwords or other sensitive information!

Should you use Post or Get?

- **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)
  - POST has **no limits** on the amount of information to send
  - POST **supports advanced functionality** such as support for multi-part binary input while uploading files to server
  - However, because the variables are not displayed in the URL, it is **not possible to bookmark** the page

PHP include and require Statements

- To insert the content of one PHP file into another PHP file (before the server executes it)

- They 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

PHP include and require Statements

- Including files saves a lot of work
  - create a standard header, footer, or menu file for all web pages
  - when the header needs to be updated, you can only update the header include file

- **Syntax**

```
include 'filename';
or
require 'filename';
```

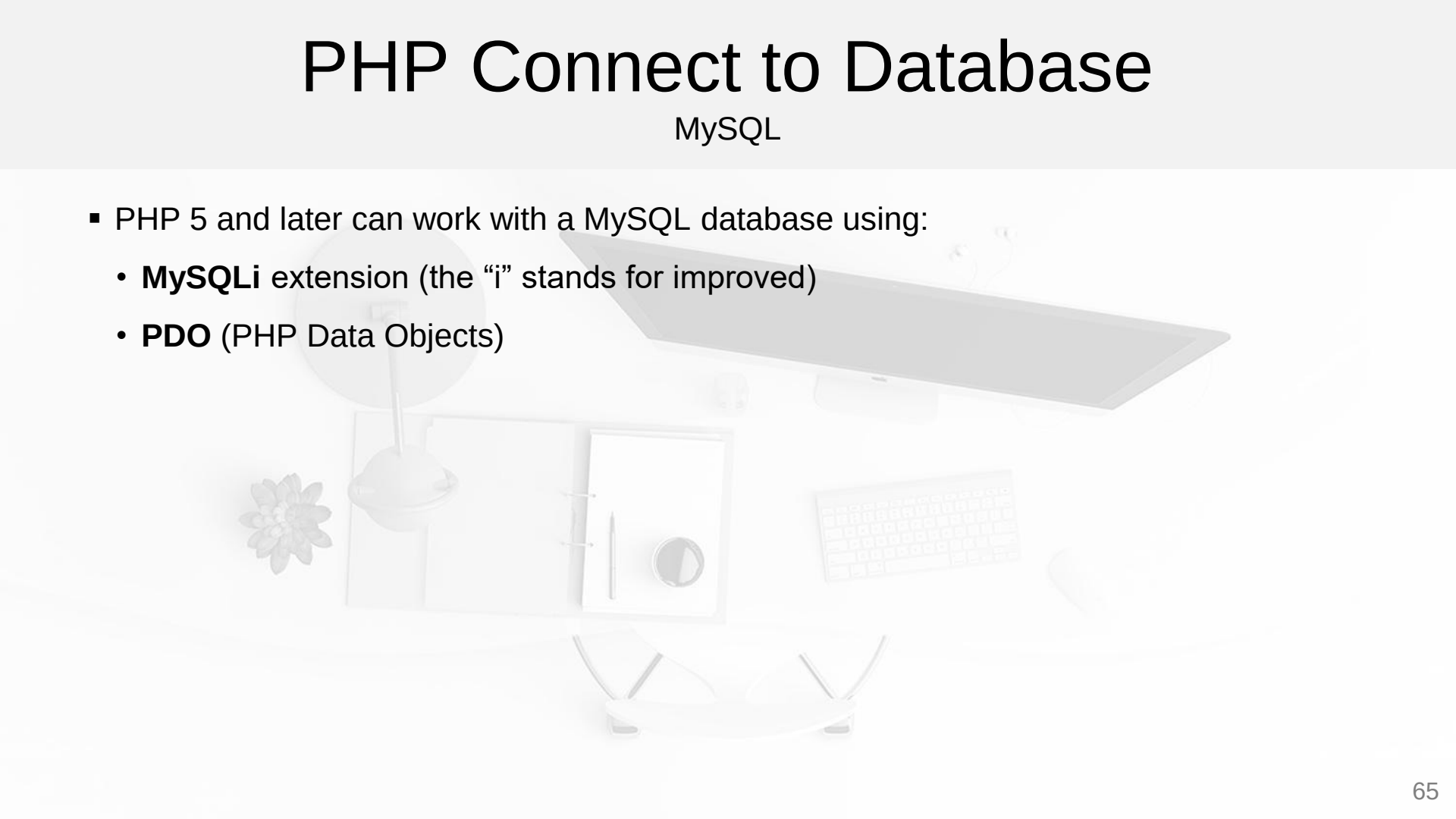
PHP Connect to Database

(MySQL)

PHP Connect to Database

MySQL

- PHP 5 and later can work with a MySQL database using:
  - **MySQLi** extension (the “i” stands for improved)
  - **PDO** (PHP Data Objects)



Should I Use MySQLi or PDO?

- Both MySQLi and PDO have their advantages:
  - PDO will work on 12 different database systems, whereas MySQLi will only work with MySQL databases

    If have to switch project to use another database,

    ✓PDO makes the process easy, only have to change the connection string and a few queries.

    ✓With MySQLi, need to rewrite the entire code - queries included
  - Both are object-oriented, but MySQLi also offers a procedural API
  - Both support **Prepared Statements**

    Prepared Statements protect from SQL injection, and are very important for web application security

Open a Connection to MySQL

Before we can access data in the MySQL database, we need to be able to connect to the server

Example (MySQLi Object-Oriented)

```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";
?>
```

Open a Connection to MySQL

Example (MySQLi Procedural)

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

// Create connection
$conn = mysqli_connect($servername, $username, $password);

// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully";
?>
```

Open a Connection to MySQL

Example (PDO)

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

try {
  $conn = new PDO("mysql:host=$servername;dbname=myDB", $username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  echo "Connected successfully";
} catch(PDOException $e) {
  echo "Connection failed: " . $e->getMessage();
}
?>
```

Close the Connection

MySQLi Object-Oriented:

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

MySQLi Procedural:

```
mysqli_close($conn);
```

PDO:

```
$conn = null;
```

PHP MySQL Insert Data

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

PHP MySQL Insert Data

MySQLi Object-oriented

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

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

PHP MySQL Insert Data

MySQLi Procedural

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

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

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

if (mysqli_query($conn, $sql)) {
  $last_id = mysqli_insert_id($conn);
  echo "New record created successfully. Last inserted ID is: " . $last_id;
} else {
  echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

PHP MySQL Insert Data

PDO

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

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $sql = "INSERT INTO MyGuests (firstname, lastname, email)
  VALUES ('John', 'Doe', 'john@example.com')";
  // use exec() because no results are returned
  $conn->exec($sql);
  $last_id = $conn->lastInsertId();
  echo "New record created successfully. Last inserted ID is: " . $last_id;
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

PHP MySQL set_charset() Function

- The function specifies the default character set to be used when sending data to and from the database server
- **Syntax:**
  - **Object oriented style:** $mysqli -> set_charset(charset)
  - **Procedural style:**

    mysqli_set_charset($connection, charset)
- Example:
  - $mysqli -> set_charset(“utf8”);
  - mysqli_set_charset($conn, “utf8”);

PHP MySQL Select Data

MySQLi Object-oriented

```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();
?>
```

PHP MySQL Select Data

MySQLi Procedural

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

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

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

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

mysqli_close($conn);
?>
```

PHP MySQL Select Data

PDO

```php
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "productmanagement";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $stmt = $conn->prepare("SELECT * FROM products");
    $stmt->execute();

    // set the resulting array to associative
    $result = $stmt->setFetchMode(PDO::FETCH_ASSOC);
    while ($row = $stmt->fetch()) {
        echo "ID: " . $row['ID'] . "<br>";
        // echo "<img src='" . $row['ProductImage'] . "'><br>";
        echo "Product Name: " . $row['ProductName'] . "<br>";
        echo "Product Price: " . $row['ProductPrice'] . "<br>";
        echo "Importe Date: " . $row['ImportedDate'] . "<br><br>";
    }
} catch(PDOException $e) {
    echo "Error: " . $e->getMessage();
}
$conn = null;
```

PHP MySQL Select Data

MySQLi Object-oriented - Fetch a result row as an associative array

```php
$mysqli = new mysqli("localhost","my_user","my_password","my_db");

if ($mysqli -> connect_errno) {
  echo "Failed to connect to MySQL: " . $mysqli -> connect_error;
  exit();
}

$sql = "SELECT Lastname, Age FROM Persons ORDER BY Lastname";
$result = $mysqli -> query($sql);

// Associative array
$row = $result -> fetch_assoc();
printf ("%s (%s)\n", $row["Lastname"], $row["Age"]);

// Free result set
$result -> free_result();

$mysqli -> close();
```

PHP MySQL Select Data

MySQLi Procedural - Fetch a result row as an associative array

```php
$con = mysqli_connect("localhost","my_user","my_password","my_db");
if (mysqli_connect_errno()) {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
  exit();
}

$sql = "SELECT Lastname, Age FROM Persons ORDER BY Lastname";
$result = mysqli_query($con, $sql);

// Associative array
$row = mysqli_fetch_assoc($result);
printf ("%s (%s)\n", $row["Lastname"], $row["Age"]);

// Free result set
mysqli_free_result($result);

mysqli_close($con);
```

PHP MySQL Delete Data

MySQLi Object-oriented

```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 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;
}

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

PHP MySQL Delete Data

MySQLi Procedural

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

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

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

if (mysqli_query($conn, $sql)) {
  echo "Record deleted successfully";
} else {
  echo "Error deleting record: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

PHP MySQL Delete Data

PDO

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

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

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

  // use exec() because no results are returned
  $conn->exec($sql);
  echo "Record deleted successfully";
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

PHP MySQL Update Data

MySQLi Object-oriented

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

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

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

PHP MySQL Update Data

MySQLi Procedural

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

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

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

if (mysqli_query($conn, $sql)) {
  echo "Record updated successfully";
} else {
  echo "Error updating record: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

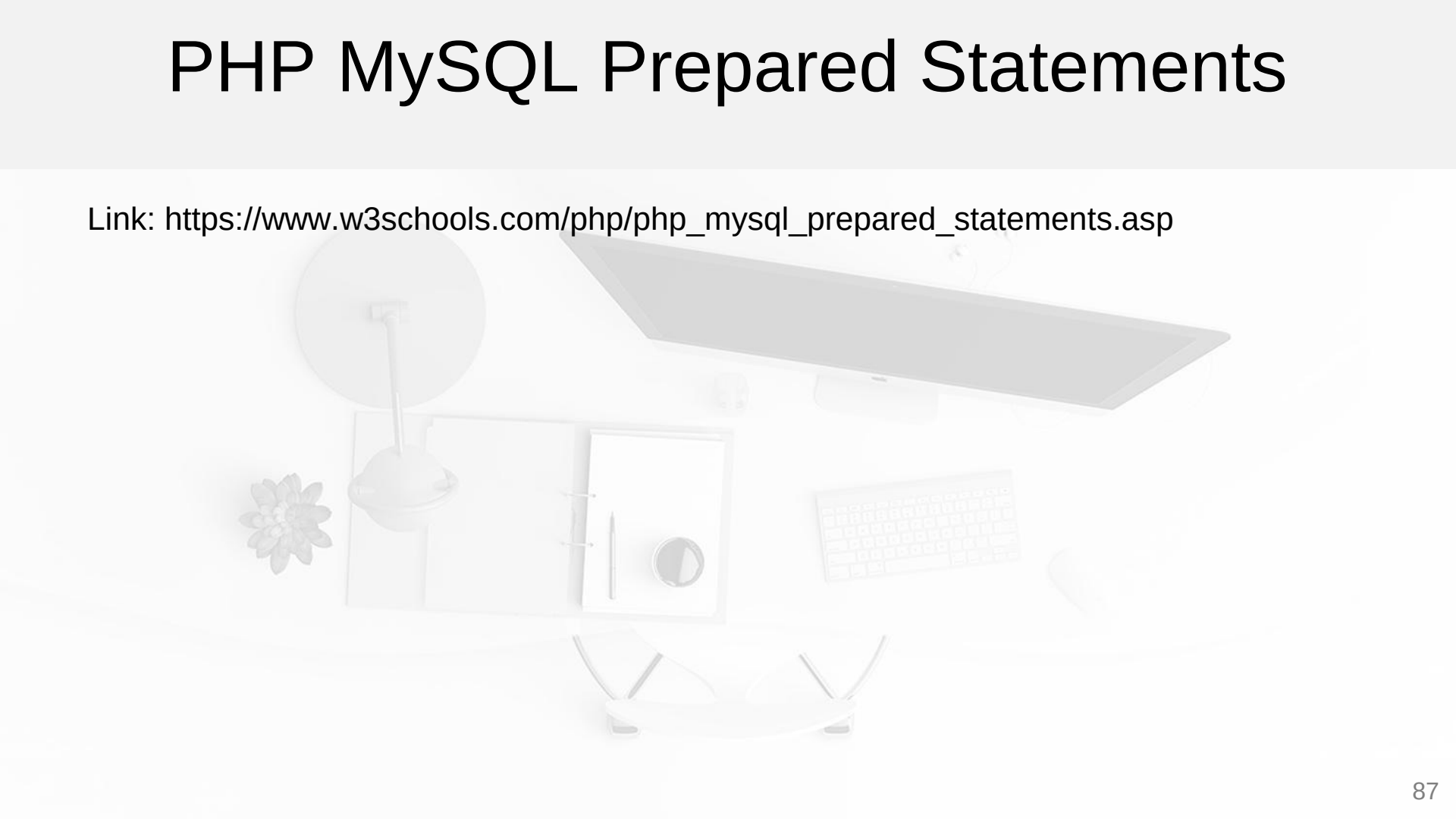
PHP MySQL Update Data

PDO

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

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

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

  // Prepare statement
  $stmt = $conn->prepare($sql);

  // execute the query
  $stmt->execute();

  // echo a message to say the UPDATE succeeded
  echo $stmt->rowCount() . " records UPDATED successfully";
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

PHP MySQL Prepared Statements

Link: https://www.w3schools.com/php/php_mysql_prepared_statements.asp




Q&A