CHAPTER 5

Programming in PHP and MYSQL

Pralhad Kumar Shrestha

Contents:

1. Origins and Uses of PHP

- Origins - Rasmus Lerdorf - 1994
  - Developed to allow him to track visitors to his Web site.
- PHP is an open-source product.
- PHP is an acronym for Personal Home Page, or PHP: Hypertext Preprocessor.
- PHP is used for form handling, file processing, and database access.
- PHP supports the common electronic mail protocols POP3 and IMAP.

2. Overview of PHP

- PHP is a server-side scripting language whose scripts are embedded in HTML documents.
  - Similar to JavaScript, but on the server side.
- PHP is an alternative to CGI, ASP.NET and Java servlets.
- The PHP processor has two modes:
  - copy (HTML) and interpret (PHP)
- PHP syntax is similar to that of JavaScript.
- PHP is dynamically typed.
- PHP is purely interpreted.

2. Overview of PHP

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

<h1>My first PHP page</h1>

<?php
echo "Hello World!";
?>

</body>
</html>
```

Syntactic Characteristics

- PHP code can be specified in an HTML document internally or externally:
- Internally: <?php ... ?>
- Externally: include ("myScript.inc")
  - the file can have both PHP and HTML
  - If the file has PHP, the PHP must be in <?php .. ?>, even if the include is already in <?php .. ?>
- Every variable name begins with a $
  - case sensitive
- PHP statements terminated with ;
- Compound statements (block of statements) are formed with braces.
- Compound statements cannot define locally scoped variables (except functions).

Syntactic Characteristics

- Comments - three different types

  // …

  # …

  /* … */

```
<?php
// This is a single-line comment

# This is also a single-line comment
?>
```

- Reserved words are not case sensitive.

  While, WHILE, While, wHiLe => while

and	else	global	require	virtual
break	elseif	if	return	xor
case	extends	include	static	while
class	false	list	switch	
continue	for	new	this	
default	foreach	not	true	
do	function	or	var	

3. Primitives, Operations, and Expressions

- Four scalar types:
  - Boolean
  - integer
  - double (floating point numbers)
  - string
- Two compound types:
  - array and object
- Two special types:
  - resource and NULL
- Integer & double are like those of other languages
- Boolean - values are true and false (case insensitive)

3. Primitives, Operations, and Expressions

- Resource is a special data type that refers to any external resource.
- A resource variable acts as a reference to external source of data such as stream, file, database etc.
- PHP uses relevent functions to create these resources.

```
<?php
$fp = fopen("test.txt","w");
var_dump($fp);
?>
```

```
//returns
resource(5) of type (stream)
```

Variables

- There are no type declarations.
- An unassigned (unbound) variable has the value *NULL.*
- The *unset* function sets a variable to NULL.
- The *IsSet* function is used to determine whether a variable is *NULL.*
- *error_reporting(15*); the interpreter will report when an unbound variable is referenced.

Variables

- Rules for variable names:
  - 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.
  - A variable name cannot start with a number.
  - A variable name 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).

String Type

- Characters are single bytes.
- String literals use single or double quotes.
- Single-quoted string literals
  - Embedded variables are NOT interpolated and embedded escape sequences are NOT recognized.
- Double-quoted string literals
  - Embedded variables ARE interpolated and embedded escape sequences ARE recognized.

```php
<?php
  $txt = "W3Schools.com";
  echo "I love $txt!";
?>
```

I love W3Schools.com!

String Type

```php
<?php
  $txt = "W3Schools.com";
  echo ‘I love $txt! <br />’;
  echo "I love $txt! <br />";
?>
```

I love $txt!
I love W3Schools.com!

Boolean Type

- Only two possible values: (case insensitive)
  - TRUE and FALSE
- Integer expression:
  - TRUE => 1
  - FALSE => 0
- String expression:
  - “” or “0” => FALSE
  - Otherwise TRUE

Predefined functions

- Arithmetic functions
  - *floor*, *ceil*, *round*, *abs*, *min*, *max*, *rand*, etc.
- String functions
  - *strlen*, *strcmp*, *strpos*, *substr*
  - *chop* – remove whitespace from the right end
  - *trim* – remove whitespace from both ends
  - *ltrim* – remove whitespace from the left en
  - *strtolower*, *strtoupper*

String Operations

```
<?php
  $string = “Apples";
  echo $string{4};
?>
```

// returns
e

```
<?php
  $string = “Apples are good";
  echo substr($string, 7, 1);
?>
```

// returns
a

Syntax
substr(*string*,*start*,*length*)

Scalar Type Conversions

- Implicit type conversion – coercions
  - String to numeric
    - If the string contains an e or an E, it is converted to double; otherwise to integer.
    - If the string does not begin with a sign or a digit, zero is used.
    - If double is converted to an integer, the fractional part is dropped; rounding is not done.
- Explicit conversions – typecasting
  - e.g., (*int*)$*total* or *intval*($*total*) or *settype*($*total*, "*integer*")
- The type of a variable can be determined with *gettype* or *is_type* function e.g.:
  - *gettype*($*total*) may return "unknown“.
  - *is_integer*($*total*) returns Boolean value.

4. Output and Control Statements

- Output from a PHP script is HTML that is sent to the browser.
- HTML is sent to the browser through standard output.
- There are three ways to produce output: *echo*, *print*, and *printf*
  - *echo* and *print* take a string, but will coerce other values to strings
  - echo “Hello there!"; echo(“Hello there!”); echo $sum;
  - print "Welcome!"; print(“Welcome”); print (46);

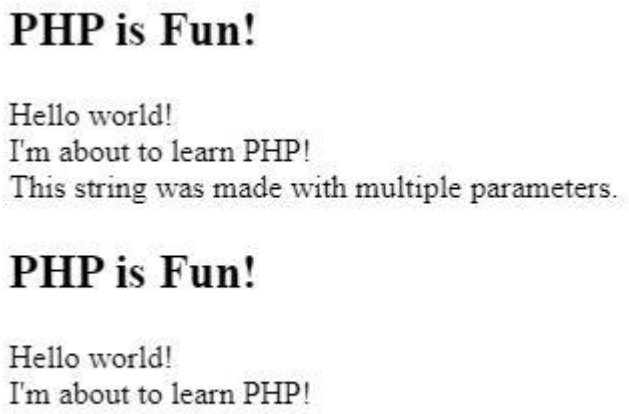
Learn More at:
https://www.w3schools.com/php/php_echo_print.asp

4. Output and Control Statements

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

<?php
echo "<h2>PHP is Fun!</h2>";
echo "Hello world!<br>";
echo "I'm about to learn PHP!<br>";
echo "This ", "string ", "was ", "made ", "with multiple
parameters.";
print "<h2>PHP is Fun!</h2>";
print "Hello world!<br>";
print "I'm about to learn PHP!";
?>

</body>
</html>
```



4. Output and Control Statements

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

<?php
    $number = 9;
    $str = "Beijing";
    printf("There are %u million bicycles in %s.",$number,$str);
?>

</body>
</html>
```

```
//result
There are 9 million bicycles in Beijing.
```

Control Statements

- Arithmetic operators
  - +, -, *, /, %, ** (exponentiation / power)
- Assignment operators
  - =, +=, -=, *=, /=, %=
- Increment / Decrement operators
  - ++$x, $x++, --$x, $x—
- Logical operators
  - and, or, xor
  - &&, ||, !
- String operators
  - . ➔ concatenation
  - .= ➔ concatenation assignment

Control Statements

Relational Operators	Descriptions
>	greater than
<	less than
>=	greater than or equal
<=	less than or equal
!=	not equal
==	equal
===	identical ➔ returns true / false (checks for same operands)
!==	not identical

Control Statements

- Selection statements
  - if, else, else if
  - switch
    - The switch expression type must be integer, double, or string
- Loops
  - while, do-while, for

5. Arrays, Functions, Basic Pattern Matching

- Arrays
- Functions
- Basic Pattern Matching
- Form Handling
- Files Handling

Arrays

- An array is a special variable, which can hold more than one value at a time.
- PHP array is really a mapping of keys to values, where the keys can be numbers (to get a traditional array) or strings (to get a hash).
- Can be created using *array()* or *[]* .

```
array("Volvo", "BMW", "Toyota");

["Volvo", "BMW", "Toyota“];
```

- The *array()* construct takes one or more key => value pairs as parameters and returns an array of them.
  - The keys are non-negative integer literals or string literals.
  - The values can be anything.

Array creation

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

$array2 = array();

$array3 = array(0 => "Volvo", 1 => "BMW", 2 => "Toyota");

$array4 = ["Andy" => 50, "Jane" => 20, "Kelie" => 17];
```

Arrays

- Indexed Arrays

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

- The above array can be assigned manually as:

```
$cars[0] = "Volvo";
$cars[1] = "BMW";
$cars[2] = "Toyota";
```

- Associative Arrays

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

OR,

```
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
```

Arrays

- Multidimensional Arrays
- A multidimensional array is an array containing one or more arrays.
- PHP supports multidimensional arrays that are two, three, four, five, or more levels deep. However, arrays more than three levels deep are hard to manage for most people

```
$cars = array (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
);
```

```
//Access the value of multidimensional array
<?php
  echo $cars[0][0];  //"Volvo"
?>
```

Accessing array elements

- Individual array elements can be accessed through subscripting

```
$list[4] = 7;
$list["day"] = "Tuesday";
$list[] = 17;
```

  - If an element with the specified key does not exist, it is created.
  - If the array does not exist, the array is created.

Functions for Dealing with Arrays

- Whole array can be deleted with unset, as with a scalar variable.

```
$list = [2, 4, 6, 8, 10];
unset($list);  // deletes whole array.
```

- Individual elements of an array can also be deleted with unset.

```
$list = [2, 4, 6, 8, 10];
unset($list[2]);  // deletes 6 from array.
```

- The keys or values can be extracted from an array

```
$highs = array("Mon" => 74, "Tue" => 70, "Wed" => 67, "Thu" => 62, "Fri" => 65);
$days = array_keys($highs);
$temps = array_values($highs);
```

Functions for Dealing with Arrays

- is_array($list) ➔ returns true if *$list* is an array.
- in_array(17, $list) ➔ returns true if 17 is an element of $list.
- explode(" ", $str) ➔ creates an array with the values of the words from *$str*, split on a space.
- implode(" ", $list) ➔ creates a string of the elements from *$list*, separated by a space.

Functions for Dealing with Arrays

Functions	Description
array_keys	returns an array containing the keys.
array_values	returns an array containing all the values of an array.
array_key_exists	checks an array for a specified key, and returns true if the key exists and false if the key does not exist.
is_array	checks whether a variable is an array or not.
sizeof	returns the number of elements in an array.
explode	breaks a string into an array.
implode	returns a string from the elements of an array.

Sequential access to array elements

- next and prev functions

  $cities = array(“London”, “Paris”, “Chicago”);

  $city = next($cities); // returns second element of an array.

Function	Description
prev	moves the internal pointer to, and outputs, the previous element in the array
current	returns the value of the current element in an array
end	moves the internal pointer to, and outputs, the last element in the array
reset	moves the internal pointer to the first element of the array
each	returns the current element key and value, and moves the internal pointer forward

Sequential access to array elements

- while loop

```
<?php

$salaries = [ 'Mike' => 42500, 'Jerry' => 51250, 'Fred' => 37950 ];

while($employee = each($salaries)) {
    $name = $employee['key'];
    $salary = $employee['value'];
    echo "The salary of $name is $salary <br />";
}

?>
```

// output

The salary of Mike is 42500
The salary of Jerry is 51250
The salary of Fred is 37950

Sequential access to array elements

- foreach statement – to build loops that process all of the elements in an array

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

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

//Output

```
red
green
blue
yellow
```

Sorting arrays

- *sort* - to sort the values of an array, leaving the keys in their present order - intended for traditional arrays. (sort arrays in ascending order)

  e.g., sort($list); //Works for both strings and numbers, even mixed strings and numbers

  $list = ('h', 100, 'c', 20, 'a');

  sort($list); // Produces ('a', 'c', 'h‘, 20, 100)

- *asort* - to sort the values of an array, but keeping the key/value relationships - intended for hashes

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

```
//result
Array ( [Peter] => 35 [Ben] => 37 [Joe] => 43 )
```

Sorting arrays

- *ksort* – sort associative arrays in ascending order, according to the key.

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

```
//result
Array ( [Ben] => 37 [Joe] => 43 [Peter] => 35 )
```

Functions

- Functions need not be defined before they are called.
- Function overloading is not supported.
  - If you try to redefine a function, it is an error.
- Function names are NOT case sensitive.
- The return statement is used to return a value.
  - If there is no return, there is no returned value.

```
function function_name([parameters]) {
        …
}
```

Function parameters

- If the caller sends too many actual parameters, the function ignores the extra ones.
- If the caller does not send enough parameters, the unmatched formal parameters are unbound.
- The default parameter passing method is pass by value (one-way communication).
- To specify pass-by-reference, prepend an ampersand to the formal parameter

```
function addOne(&$param) {
  $param++;
}
$val = 16;
addOne($val);  // $val is now 17
```

Scope of Variables

- Variables defined in a function have local scope.
- To access a non-local variable in a function, it must be declared to be global (within the function code).

```
global $sum;
```

```
<?php //local variable
$x = 0;
function myTest() {
  $x = 5; // local scope
  echo "<p>Variable x inside function is: $x</p>";
}
myTest();

// the $x outside function will not be affected with the value from function.
echo "<p>Variable x outside function is: $x</p>";
?>
```

```
//result
Variable x inside function is: 5
Variable x outside function is: 0
```

Scope of Variables

```
<?php
$x = 5;
$y = 10;

function myTest() {
  global $x, $y;
  $y = $x + $y;
}

myTest();  // run function
echo $y; // output the new value for variable $y
?>
```

```
//result
15
```

The Lifetime of Variables

- Normally, the lifetime of a variable in a function is from its first appearance to the end of the function's execution.
- To support history sensitivity a function must have static local variables.

  static $sum = 0;

  – Its lifetime ends when the browser leaves the document in which the PHP script is embedded.

```
function do_it($param) {
    static $count = 0;
    $count++;
    print "do_it has now been called $count times <br />";
}

do_it(1);
do_it(1);
do_it(1);
```

```
//result
do_it has now been called 1 times
do_it has now been called 2 times
do_it has now been called 3 times
```

Pattern Matching

- PHP includes two different kinds of string pattern matching using RegEx.
  - based on POSIX RegEx – already compiled into PHP
  - based on Perl RegEx – needs to be compiled before Perl RegEx can be used (always enabled) – used in JS RegEx.

```
<?php
$str = "Visit W3Schools";
$pattern = "/w3schools/i";
echo preg_match($pattern, $str);  //returns 1
?>
```

```
preg_match(pattern, input, matches, flags, offset)
// Returns 1 if a match was found, 0 if no matches were found and false if an error occurred.
```

Learn More at:
https://www.w3schools.com/php/func_regex_preg_match.asp

Pattern Matching

```
<?php
$str = "PHP is my name.";

if (preg_match("/^PHP/", $str)) {
    print "\$str begins with PHP <br />";
} else {
    print "\$str does not begins with PHP <br />";
}
?>
```

```
//returns
$str begins with PHP
```

```
<?php
$fruit_string = "apple : orange : banana";

$fruits = preg_split("/ : /", $fruit_string);
var_dump($fruits);
?>
```

```
//returns
array(3) { [0]=> string(5) "apple" [1]=> string(6)
"orange" [2]=> string(6) "banana" }
```

Form Handling

- Forms could be handled by the same document that creates the form, but that may be confusing.
  - A separate document to handle the form can be specified as the value of the action attribute.
- It does not matter whether GET or POST method is used to transmit the form data.
- Superglobals $_GET and $_POST are used to collect form-data.
- PHP builds an array of the form values
  - $_GET for the GET method
  - $_POST for the POST method
- Developers prefer POST for sending form data.

Form Handling

- Both GET and POST create an array

  e.g. array(key1 => value1, key2 => value2, key3 => value3, ...)
- Superglobals means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.
- $_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.
- 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.

Form Handling

```
<form action="login.php" method="post">
    <input type="text" name="name"><br>
    <input type="text" name="email"><br>
    <input type="submit">
</form>
```

```
//login.php
<html>
<body>

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

</body>
</html>
```

```
//result
Welcome John
Your email address is john.doe@example.com
```

Form Handling

```
<form action=“login.php" method=“get">
    <input type="text" name="name"><br>
    <input type="text" name="email"><br>
    <input type="submit">
</form>
```

```
//login.php
<html>
<body>

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

</body>
</html>
```

```
//result
Welcome John
Your email address is john.doe@example.com
```

Files Handling

- PHP is a server-side technology, and is able to create, read and write files on the server system.
- All file operations in PHP are implemented as functions.
- **Opening and closing a file:**
  - Prepares file for use and associates a variable with the file for future reference.

    $fptr = fopen(filename, use_indicator)
  - Every open file has an internal pointer (where the next file operations should take place)
  - Because *fopen* could fail, use it with die

    $file_var = fopen (“test.dat”, “r”) or die (“Error – test.dat can’t be opened”);

Files Handling

Use Indicator	Description
`"r"`	Read only. The file pointer is initialized to the beginning of the file.
`"r+"`	Read and write an existing file. The file pointer is initialized to the beginning of the file; if a read operation precedes a write operation, the new data is written just after where the read operation left the file pointer.
`"w"`	Write only. Initializes the file pointer to the beginning of the file; creates the file if it does not exist.
`"w+"`	Read and write. Initializes the file pointer to the beginning of the file; creates the file if it does not exist. Always initializes the file pointer to the beginning of the file before the first write, destroying any existing data.
`"a"`	Write only. If the file exists, initializes the file pointer to the end of the file; if the file does not exist, creates it and initializes the file pointer to its beginning.
`"a+"`	Read and write a file, creating the file if necessary; new data is written to the end of the existing data.

Files Handling

- Use *file_exists(filename)* to determine whether file exists before trying to open it
- Use *fclose(file_var)* to close a file

```php
<?php
  $myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
  echo fread($myfile,filesize("webdictionary.txt"));
  fclose($myfile);
?>
```

```
//result
AJAX = Asynchronous JavaScript and XML CSS = Cascading Style Sheets HTML = Hyper Text
Markup Language PHP = PHP Hypertext Preprocessor SQL = Structured Query Language SVG =
Scalable Vector Graphics XML = EXtensible Markup Language
```

Files Handling

- **Reading from a file:**
  1. Read all or part of the file into a string variable.
     - $str = fread($file_var, #bytes)
     - To read the whole file, use *filesize(file_name)* as the second parameter
     - $file_string = fread ($file_var, filesize("test.dat"));
     - Large collection of data are often stored in database, so, usually small data sets are stored in files.
  2. Read the lines of the file into an array(each new line is a separate array value).
     - $file_lines = file(file_name);
     - Need not open or close the file

Files Handling

3. Read one line from the file
   - $line = fgets(file_var, #bytes)
   - Reads characters until newline, eof, or #bytes characters have been read
4. Read one character at a time
   - $ch = fgetc(file_var)
   - Control reading lines or characters with eof detection using feof (TRUE for eof; FALSE otherwise)

```
while(!feof($file_var)) {
        $ch = fgetc($file_var);
  }
```

Files Handling

- **Writing to a file:**
  - $bytes_written = fwrite($file_var, $out_data);
  - *fwrite* returns the number of bytes it wrote

```
<?php
$file = fopen("test.txt","w");
echo fwrite($file,"Hello World. Testing!");
fclose($file);
?>
```

- **Locking files:**
- Files can be locked (to avoid interference from concurrent accesses) with flock
  - Takes 2 parameters – file variable and intreger that specifies particular operation
    - 1 – file can be read by others
    - 2 – no other access
    - 3 – unlocks file

6. Cookies, Sessions and Database Access

- Cookies
- Session Tracking
- Database Access with PHP and MySQL

Cookies

- The HTTP protocol is stateless – no means to store information about a session that is available to a subsequent session.
- Cookies helps to store information about sessions on the browser itself.
- Create a cookie with *setcookie*.

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

  setcookie("voted", "true", time() + 86400);
- Cookies must be created before any other HTML is created by the PHP document.
  - Because cookies stored in HTTP header.
- Cookies are obtained in a script the same way form values are gotten, using the *$_COOKIES* array (cookie names as keys).
  - Use *IsSet* to check if a particular cookie came with the request.

Cookies

```
<!DOCTYPE html>
<?php
$cookie_name = "user";
$cookie_value = "John Doe";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/");?>
<html>
<body>
<?php
if(!isset($_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];
}?>
<p><strong>Note:</strong> You might have to reload the page to see the
value of the cookie.</p>
</body>
</html>
```

Session Tracking

- A session is the time span during which a browser interacts with a particular server.
- An alternative to cookies.
- For session tracking, PHP creates and maintains a session tracking id.
- Create the id with a call to *session_start* with no parameters.
- Subsequent calls to *session_start* retrieve any session variables that were previously registered in the session (in $_SESSION array).
- Session variable are created or changed by assignments to the $_SESSION array.
- Remove all the session variables using *session_unset* and destroy the session using *session_destroy* function.

Session Tracking

- Example: count number of pages visited

  Put the following code in all documents

```
session_start();
 if (!IsSet($_SESSION[“page_number”})) {
       $_SESSION[“page_number”] = 1;
}

$page_num = $_SESSION[“page_number”];

 print("You have now visited $page_num page(s) <br>");
 $_SESSION[“page_number”]++;
```

Session Tracking

```php
<?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>
```

// result
Session variables are set.

Session Tracking

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

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

// destroy the session
session_destroy();
?>

</body>
</html>
```

Database Access with PHP and MySQL

- A database is a collection of data organized to allow relatively easy access for retrievals, additions, modifications, and deletions.
- The most widely used approach to structuring data are called relational database systems.
- A relational database is a collection of tables of data.
- Each table can have any number of rows and columns of data.
- The data itself can have a variety of different forms.

Database Access with PHP and MySQL

- MySQL is a freely available open source Relational Database Management System (RDBMS) that uses Structured Query Language (SQL).
- The Structured Query Language (SQL) is a standard language for specifying accesses and modifications to relational databases.
- Database queries: A query is a question or a request.
- We can query a database for specific information and have a record set returned.
- *mysqli_query()* is the function that executes the SQL queries.
- *mysqli_error()* function returns the last error description for the most recent function call, if any.

MySQL connection using PHP

- Before we can access data in the MySQL database, we need to be able to connect to the server.
- The PHP function *mysqli_connect* connects a script to a MySQL server.
- This function takes six parameters, all of which are optional.

Object Oriented Style:

$mysqli = new mysqli($host, $username, $passwd, $dbName, $port, $socket);

Or

$mysqli = new mysqli($host, $username, $passwd);

Procedural Style:

$mysqli = mysqli_connect(host, username, password, dbname, port, socket);

Or

$mysqli = mysqli_connect(host, username, password);

MySQL connection using PHP

Parameters	Descriptions	Default Value
$host	Specifies a host name or an IP address.	**localhost** or (localhost:3306)
$username	Specifies the MySQL username.	name of the user (root)
$password	Specifies the MySQL password.	empty password or blank
$dbName	Specifies the default database to be used.	“”
$port	Specifies the port number to attempt to connect to the MySQL server.	**3306**
$socket	Specifies the socket or named pipe to be used.	

MySQL connection using PHP

- If the connect operation is failed the value returned us false, otherwise it returns a reference to the database.
- Therefore, the call to *mysql_connect* usually is used in conjunction with die.
- The connection to a database is terminated with the *mysql_close* function.
- *mysql_select_db* function is used to select a particular database on the server.

```
$conn = mysqli_connect($servername, $username, $password) or die("Connection failed: " .
mysqli_connect_error());
```

MySQL connection using PHP

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

Object Oriented Style

Procedural Style

```php
<?php
// Create connection
$conn = mysqli_connect("localhost", "username", "password");
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}
mysqli_close($conn);
?>
```

SQL Database

- A database consists of one or more tables.
- Special **CREATE** privileges is required to create or to delete a MySQL database.
- The **CREATE DATABASE** statement is used to create a database in MySQL.

```php
<?php
// mysql connection codes
// Create database
$sql = "CREATE DATABASE myDB";
if ($conn->query($sql) === TRUE) {
  echo "Database created successfully";
} else {
  echo "Error creating database: " . $conn->error;
}
// Close connection
?>
```

SQL Database

// Full example to create a database

```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();
?>
```

SQL Database

```php
// Common code to reuse for database connection.
// filename => connector.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);
}
?>
```

SQL Database

// Create a database with common connection code

```php
<?php
require_once('connector.php');

// Create database
$sql = "CREATE DATABASE my_database";
if ($conn->query($sql) === TRUE) {
  echo "Database created successfully";
} else {
  echo "Error creating database: " . $conn->error;
}
$conn->close();
?>
```

SQL Table

- A database table has its own unique name and consists of columns and rows.
- The **CREATE TABLE** statement is used to create a table in MySQL.

```
// SQL syntax to create a table
CREATE TABLE table_name (
  column1 data_type constraints,column2 data_type constraints, ….., columnN data_type constraints
);

// SQL statement to create a table called MyGuests with columns and attributes
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 DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

SQL Table

- After the data type, you can specify other optional attributes for each column:
  - **NOT NULL** - Each row must contain a value for that column, null values are not allowed
  - **DEFAULT** value - Set a default value that is added when no other value is passed
  - **UNSIGNED** - Used for number types, limits the stored data to positive numbers and zero
  - **AUTO INCREMENT** - MySQL automatically increases the value of the field by 1 each time a new record is added
  - **PRIMARY KEY** - Used to uniquely identify the rows in a table. The column with PRIMARY KEY setting is often an ID number, and is often used with AUTO_INCREMENT

SQL Table

// Create a table with common connection code

```php
<?php
require_once('connector.php');
// 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 DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)";

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

SQL Insert Command

- After a database and a table have been created, we can start inserting data into them.
- Syntax rules to follow during writing queries:
  - 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

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


//INSERT Query Example:
"INSERT INTO MyGuests (firstname, lastname, email) VALUES ('John', 'Doe', 'john@example.com')"
```

SQL Insert Command

// Insert a data into table

```php
<?php
require_once('connector.php');
$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();
?>
```

SQL Select Command

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

```
//Select specific columns from tables
SELECT column_name(s) FROM table_name

//Select all columns from table
SELECT * FROM table_name
```

- The WHERE clause is used to filter records.
- The WHERE clause is used to extract only those records that fulfill a specified condition.

```
//Select specific columns from table that matches the criteria
SELECT column_name(s) FROM table_name WHERE column_name operator value

//INSERT Query Example:
"SELECT id, firstname, lastname FROM MyGuests WHERE lastname='Doe'";
```

SQL Select Command

// Select all data from table having specific columns

```
<?php
require_once('connector.php');
$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();
?>
```

SQL Select Command

// Select specific columns with data having lastname as “Doe” from table

```
<?php
require_once('connector.php');
$sql = "SELECT id, firstname, lastname FROM MyGuests WHERE lastname = 'Doe'";
$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();
?>
```

SQL Update Command

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

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

```
//UPDATE Query Example:
"UPDATE MyGuests SET lastname='Doe' WHERE id=2";
```

SQL Update Command

```php
<?php                                   // Update lastname of all data from table having specific condition
require_once('connector.php');
// sql to update a record
$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();
?>
```

SQL Delete Command

- The DELETE statement is used to delete records from a table.

```
//Delete command
DELETE FROM table_name
WHERE some_column = some_value
```

```
//DELETE Query Example:
"DELETE FROM MyGuests WHERE id=3";
```

SQL Delete Command

```php
<?php                                   // Delete a data from table having id = 3
require_once('connector.php');
// sql to delete a record
// 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();
?>
```

SQL Table

- The DROP TABLE statement is used to drop an existing table in a database.

```
//Drop command
DROP TABLE table_name;
```

```
//DROP Query Example:
DROP TABLE Shippers;
```

- The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

```
//Truncate command
TRUNCATE TABLE table_name;
```

```
//TRUNCATE Query Example:
TRUNCATE TABLE Shippers;
```

SQL Database

- The DROP DATABASE statement is used to drop an existing SQL database.

```
//Drop command
DROP DATABASE database_name;
```

```
//DROP Query Example:
DROP DATABASE my_db;
```

SQL JOIN

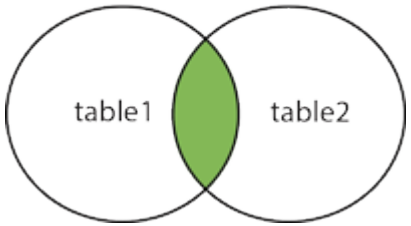
- A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

```
//JOIN command
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

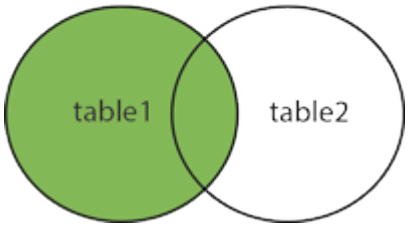
- Types of JOINS:
  - (INNER) JOIN: Returns records that have matching values in both tables
  - LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table
  - RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table
  - FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table

Types of SQL JOIN

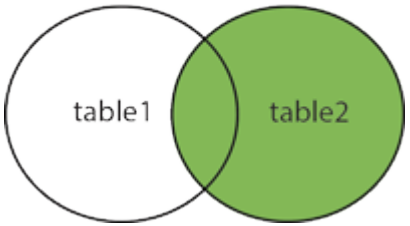
INNER JOIN



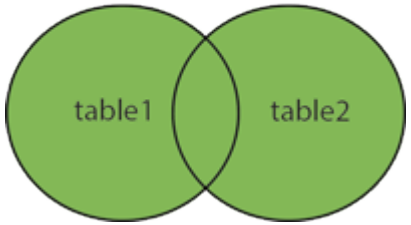
LEFT JOIN



RIGHT JOIN



FULL OUTER JOIN



Insert Form data to database

```
<?php
require('connector.php');

$name = $_POST['name'];
$regno = $_POST['regno'];
$address = $_POST['address'];

$database = "web_class";
$conn->select_db($database);
$table = 'users';
$sql = "INSERT INTO $table (name, regno, address) VALUES ('$name', '$regno',
'$address')";

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

End of chapter 5