

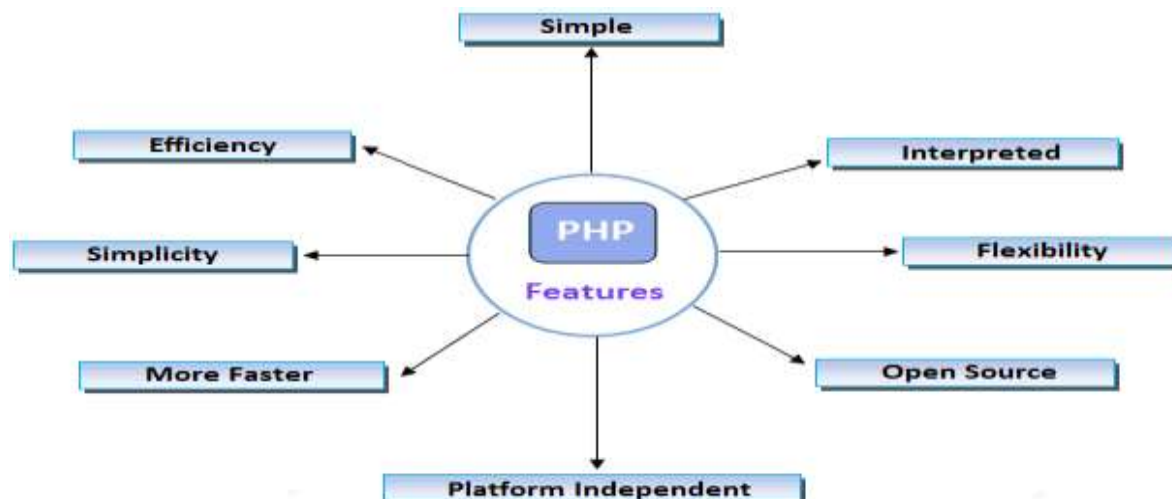
# Programming with PHP and MySQL

## UNIT -1

### 1.1 History of PHP

- PHP (PHP: Personal Home Page) was created by Rasmus Lerdorf in It was initially developed for HTTP usage logging and server-side form generation in Unix.
- PHP 2 (1995) transformed the language into a Server-side embedded scripting language. Added database support, file uploads, variables, arrays, recursive functions, conditionals, iteration, regular expressions, etc.
- PHP 3 (1998) added support for ODBC data sources, multiple platform support, protocols and new parser written by Zeev Suraski and Andi Gutmans .
- PHP 4 (2000) became an independent component of the web server for added efficiency. The parser was renamed the Zend Engine. Many security features were added.
- PHP 5 (2004) adds Zend Engine II with object oriented programming, robust XML support using the libxml2 library, SOAP extension for interoperability with Web Services, SQLite has been bundled with PHP

### 1.2 Features of PHP



➤ **Simple**

It is very simple and easy to use, compared to another scripting language. This is widely used all over the world.

➤ **Interpreted**

It is an interpreted language, i.e. there is no need for compilation.

➤ **Faster**

It is faster than other scripting languages e.g. asp and jsp.

➤ **Open Source**

Open source means you no need to pay for using PHP, you can free download and use.

➤ **Platform Independent**

PHP code will be run on every platform, Linux, Unix, Mac OS X, Windows.

➤ **Flexibility**

PHP is known for its flexibility and embedded nature as it can be well integrated with HTML, XML, Javascript and many more. PHP can run on multiple operating systems like Windows, Unix, Mac OS, Linux, etc.

➤ **Case Sensitive**

PHP is case sensitive scripting language at the time of variable declaration. In PHP, all keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are NOT case-sensitive.

### **1.3 Variables**

- Variables are used to store both numeric and non numeric data.
- The content of variable can be altered during program execution
- variables can be compared and manipulated using operators.

- All variables in PHP are denoted with a leading dollar sign (\$).
- The variable name must begin with a letter or the underscore character.
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_)
- **PHP has a total of eight data types which we use to construct our variables -**

**1. Integers** - are whole numbers, without a decimal point, like 4195.

**2. Doubles** - are floating-point numbers, like 3.14159 or 49.1.

**3. Booleans** - have only two possible values either true or false.

**4. NULL** - is a special type that only has one value: NULL.

**5. Strings** - are sequences of characters, like 'PHP supports string operations.'

**6. Arrays** - are named and indexed collections of other values.

**7. Objects** - are instances of programmer-defined classes, which can package up both other kinds of values and functions that are specific to the class.

**8. Resources** - are special variables that hold references to resources external to PHP (such as database connections).

- **Assigning and Using Variables Values**

- To assign a value to a variable , use assign operator (=) symbols.

- = operator assigns a value to a variable.

```
<?php
$today= "Aug 8 2020";
Echo "Today is $today";
?>
```

- PHP supports number of specialize functions to check if a variable or value belong to a specific type

1. is\_bool()
2. is\_string()
3. is\_numeric()
4. is\_float()
5. is\_int()
6. is\_null()
7. is\_array()

- **Echo()** function is used to print data to standard output.

### **Example of Integer values**

```
<html>
<head><title>Example</title></head>
<body>
<?php
// define variable
$first = 100;
$second = 200;
$third = $first + $second;
// print output
echo "Sum = "$third;
?>
</body>
</html>
```

**Output:** Sum = 300

### **Example of string values**

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

Output:

I love W3Schools.com!

### **NULL Value**

- Null is a special data type which can have only one value: NULL.
- A variable of data type NULL is a variable that has no value assigned to it.
- If a variable is created without a value, it is automatically assigned a value of NULL.

```
<?php
$x = "Hello world!";
$x = null;
echo $x;
?>
```

**Output:** NULL

### **1.4 Statement Operators**

- Operators are used to perform operations on variables and values.
- PHP divides the operators in the following groups:
  1. Arithmetic operators
  2. Assignment operators
  3. Comparison operators
  4. Increment/Decrement operators
  5. Logical operators

## 6. String operators Value

### 1.Arithmetic Operators

- The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.

Operator	Name	Example
+	Addition	$\$x + \$y$
-	Subtraction	$\$x - \$y$
*	Multiplication	$\$x * \$y$
/	Division	$\$x / \$y$
%	Modulus	$\$x \% \$y$
**	Exponentiation	$\$x ** \$y$

### 2.Assignment Operators

- The PHP assignment operators are used with numeric values to write a value to a variable.
- The basic assignment operator in PHP is "=".
- It means that the left operand gets set to the value of the assignment expression on the right.

## Program :

```
<?php  
$x = 10;  
$y = 4;
```

## OutPut:

```
echo($x + $y);
```

14  
6

```
echo($x - $y);
```

40  
2.5

```
echo($x * $y);
```

2

```
echo($x / $y);
```

```
echo($x % $y);
```

```
?>
```

### 3. Comparison Operators

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

Operator	Name	Example
==	Equal	\$x == \$y
===	Identical	\$x === \$y
<>	Not equal	\$x <> \$y
!==	Not identical	\$x !== \$y
>	Greater than	\$x > \$y
<	Less than	\$x < \$y
>=	Greater than or equal to	\$x >= \$y
<=	Less than or equal to	\$x <= \$y

#### 4. Increment / Decrement Operators

- The PHP increment operators are used to increment a variable's value.
- The PHP decrement operators are used to decrement a variable's value.

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

#### 5. Logical Operators

- The PHP logical operators are used to combine conditional statements.

Name	Example	Result
And	<code>\$x and \$y (or)</code> <code>\$x &amp;&amp; \$y</code>	True if both <code>\$x</code> and <code>\$y</code> are true
Or	<code>\$x or \$y (or)</code> <code>\$x    \$y</code>	True if either <code>\$x</code> or <code>\$y</code> is true
Xor	<code>\$x xor \$y</code>	True if either <code>\$x</code> or <code>\$y</code> is true, but not both
And		True if both <code>\$x</code> and <code>\$y</code> are true
Not	<code>!\$x</code>	True if <code>\$x</code> is not true

#### 6. String Operators

- PHP has two operators that are specially designed for strings.

Operator	Name	Example
<code>.</code>	Concatenation	<code>\$txt1 . \$txt2</code>



**Example :**

```
<?php
$txt1 = "Hello";
$txt2 = " world!";
echo $txt1 . $txt2;
?>
```

**Output:**

Hello world!

**1.5 Conditional Statement**

- A Conditional Statement enables you to test whether a specific condition is true or false and to perform different actions on the basis of the test result.

**1. Using if () Statement,**

In PHP, the simplest form of conditional statement is the if() statement, which looks like this:

```
<?php
If(conditional test) ,,3 which evaluate either true or false
{
do this; ,,3 if true within the curly braces is executed
}
?>
```

**Example**

```
<?php
$mark = 120;
if($mark >= 80)
{
echo "you have an A";
}
?>
```

**Output:**

you have an A

- PHP also offer if - else(),used to defines an alternate block of code that get executed when the conditional expression in if () statement evaluates as false.

```
<?php
```

```
If(conditional test) ,,3 which evaluate either true or false
```

```
{
```

```
do this; ,,3 if true within the curly braces is executed
```

```
}
```

```
else
```

```
{
```

```
do this ,,3 if false within the curly braces is executed
```

```
} ?>
```

**Example**

```
<?php
```

```
$mark = 60;
```

```
if($mark >= 80)
```

```
{
```

```
echo "you have an A";
```

```
}
```

```
else
```

```
{
```

```
echo " you have an B";
```

```
}
```

```
?>
```

**Output:**

you have an B

- PHP also provide you with a way handling multiple possibilities the if-else is-else() construct.
- This construct consists of listing number of possible results, one after another, specifying the action to be taken for each.

```
if (condition1)
```

```
{
```

```
//code 1 to be executed
```

```
}
```

```
elseif(condition2)
```

```
{
```

```
//code 2 to be executed
```

```
}
```

```
else
```

```
{
```

```
//code to be executed if code 1 and code 2 are not
```

```
}
```

# Example

```
<?php
//defining a variable
$marks = 75;
if ($marks>79)
{
echo "A";
}
elseif ($marks<=79&&
$marks>60)
{
echo "B";
}
```

```
elseif($marks<=60&&
$marks>50)
{
echo "C";
}
elseif($marks=50)
{
echo "D";
}
Else
{
echo "F";
}
?>
Output : B
```

## 2. Switch() Statement

- The **switch statement** is very similar to the **if...else statement**.
- But in the cases where your conditions are complicated like you need to check a condition with multiple constant values, a **switch statement** is preferred to an **if...else**.
- The examples below will help us better understand the **switch statements**.

```
switch (n)
{
case constant1:
// code to be executed if n is equal to constant1;
break;
case constant2:
// code to be executed if n is equal to constant2;
```

```
break;
```

```
. . . default:
```

```
// code to be executed if n doesn't match any
```

```
constant
```

```
}
```

## Example:

```
<?php
//variable definition
$gender = 'M';
switch ($gender)
{
  case 'F':
    echo 'F is FEMALE';
```

```
break;
  case 'M':
    echo 'M is MALE';
    break;
  default:
    echo 'Invalid choice';
}
?>
```

## Output :

M is MALE

### 1.6 Nesting Conditional Statement

- To handle multiple conditions, you can “nest” conditional statements include each other.
- Its structure will look like

```
if (expression 1 )
{
  if (expression 2 )
  {
    if (expression 3 )
    {
      // statements 1
    }
  }
}
```

# Example

```
<?PHP

$country = "India";
$state = "Tamilnadu";
$city = "Kovilpatti";

if ($country == "India")
{
    if ($state == "Tamilnadu")
    {
        if ($city == "Kovilpatti")
        {
            echo "nice country";
        }
        else
        {
            echo "city not match";
        }
    }
    else
    {
        echo "state not match";
    }
}
else
{
    echo "country not match";
}
?>
```

## 1.7 Merging Forms and Their Result

- Normally, when creating and processing forms in PHP, you would place the HTML form in one file and handle form processing through a separate PHP Script.
- Example you have seen so far have worked.
- With the power of conditional statement at your disposal, you can combine both pages into one.
- To do this, assign a name to the form's SUBMIT control and then check whether the special `$_POST` container variable contains that name when script first loads up.
- You can use a single PHP script to generate both initial form and post submission output.

### Example:

```
<html>
<head><title>Merging Form</title>
</head>
<body>
<? PHP
//if the SUBMIT variable does not exist
// FORM has not been submitted
//Display initial page
If (!$_POST['submit'])
{
?>
<form action =“<?=$_SERVER['PHP_SELF']?>” method=“post”>
Enter Number: <input name=“number” size=“2”>
<input type=“submit” name=“submit” value=“Go”>
</form>
```



</body>

</html>

## **1.8 Repeating Action with LOOPS**

- A LOOP is a control structure that enables you to repeat the same set of statement or commands over and over again.
- The actual number of repetitions may be dependent on a number you specify or fulfillment of a certain condition or set of conditions.

### **1. Using While() Loop:**

- The first and simplest to loop learn in PHP is called While() Loop.
- With this loop type, so long as conditional expression specified evaluates to true, the loop will continue to execute.
- When Condition become false, the loop broken and the statement will executed.

<b>Syntax:</b> <pre>&lt;?php While (condition is true) { Do this; } ?&gt;</pre>	<b>Example:</b> <pre>&lt;?php   \$x = 1;    while(\$x &lt;= 5)   {     echo "The number is: \$x &lt;br&gt;";     \$x++;   } ?&gt;</pre> <b>Output:</b> The number is: 1 The number is: 2 The number is: 3 The number is: 4 The number is: 5
--	---

## 2. Using the do() Loop

- The do...while loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true.

### Syntax:

```
<?php
do
{
code to be executed;
} while (condition is true);
?>
```

### Example

```
<?php
```

```
$x = 1;
do {
echo "The number is: $x <br>";
$x++;
} while ($x <= 5);
?>
```

### **Output:**

The number is: 1  
The number is: 2  
The number is: 3  
The number is: 4  
The number is: 5

### **3. Using the for() Loop**

- The for loop - Loops through a block of code a specified number of times.

### **Syntax**

```
<?php
for (init counter; test counter; increment counter)
{
code to be executed for each iteration;
}
?>
```

### **Parameters:**

- ***init counter***: Initialize the loop counter value
- ***test counter***: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- ***increment counter***: Increases the loop counter value

## Example:

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

## Output:

```
The number is: 0  
The number is: 1  
The number is: 2  
The number is: 3  
The number is: 4  
The number is: 5  
The number is: 6  
The number is: 7  
The number is: 8  
The number is: 9  
The number is: 10
```

## 1.9 Controlling Loop Iteration With Break And Continue

### 1. Break

- The break statement can be used to jump out of a loop.

## Example of Break:

```
<?php
for ($x = 0; $x < 10; $x++)
{
    if ($x == 4)
    {
        break;
    }
    echo "The number is: $x
    <br>";
}
?>
```

## Output:

The number is: 0  
The number is: 1  
The number is: 2  
The number is: 3



## 2. Continue

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

## Example of Continue:

```
<?php
for ($x = 0; $x < 10; $x++)
{
    if ($x == 4) {
        continue;
    }
    echo "The number is: $x
    <br>";
}
?>
```

### Output:

```
The number is: 0
The number is: 1
The number is: 2
The number is: 3
The number is: 5
The number is: 6
The number is: 7
The number is: 8
The number is: 9
```

## Unit – II

### 2.1 Arrays

- An array is a data structure that stores one or more similar type of values in a single variables.
- If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
$cars1 = "Volvo";
```

```
$cars2 = "BMW";
```

```
$cars3 = "Toyota";
```

- An array can be created using the array() language construct.
- There are three different kind of arrays:

**1. Indexed arrays** - These arrays can store numbers, strings and any object but their index will be represented by numbers. By default array index starts from zero.

**ex:** \$car[0]="Nano",

```
$car[1]="Audi",
```

```
$car[2]="Hundai"
```

**2. Associative arrays** - Associative array will have their index as string so that you can establish a strong association between key and values.

**ex:** \$car[N]="Nano", \$car[A]="Audi", \$car[H]="Hundai"

**3. Multidimensional arrays** - Arrays containing one or more arrays

**ex:** \$myarray = array(  
array("Ankit", "Ram", "Shyam"),  
array("Unnao", "Trichy", "Kanpur")  
);

## 2.2 Creating an array

- To define an array variables, name it using standard PHP variables rules and populate it with elements using array() function as follow:

```
<?php
//define an array
$flavors =
array('strawberry','grape','vanilla','chocolate')
?>
```

- An alternative way to defines an array is by specifying values for each element using index notation like this:

```
<?php
//define an array
$flavors[0] = 'strawberry',
$flavors[1] ='grape'
$flavors[2] ='vanilla'
$flavors[3] ='chocolate'
?>
```

- To create an associative array, use keys instead of numeric indices:

```
<?php
//define an array
$flavors[S] = 'strawberry',
$flavors[G] ='grape'
$flavors[V] ='vanilla'
$flavors[C] ='chocolate'
?>
```

## 2.3 Modifying Arrays

- You can modify the values in arrays as easily as other variables. One way is to access an element in an array simply by referring to it by index.



- For example, say you have this array:

```
$fruits[0] = "pineapple";
```

```
$fruits[1] = "pomegranate";
```

```
$fruits[2] = " apple";
```

- Now we want to change the value

of \$fruits[1] to "watermelon" then give as follow:

```
$fruits[0] = "pineapple";
```

```
$fruits[1] = "pomegranate";
```

```
$fruits[2] = " apple ";
```

```
$fruits[1] = "watermelon";
```

- If you want to add a new element, "grapes", to the end

of the array as follows:

```
$fruits[0] = "pineapple";
```

```
$fruits[1] = "pomegranate";
```

```
$fruits[2] = " apple ";
```

```
$fruits[1] = "watermelon";
```

```
$fruits[] = "grapes";
```

- Example program as follows:

<HTML>	for (\$index = 0;
<HEAD>	\$index < count(\$fruits);
<TITLE>	\$index++)
Modifying an array	{
</TITLE>	echo \$fruits[\$index], " ";
</HEAD>	}
	?>
<BODY>	</BODY>
<H1>	</HTML>
Modifying an array	<b>Output:</b>
</H1>	<b>Modifying an array</b>
<?php	pineapple
\$fruits[0] = "pineapple";	pomegranate
\$fruits[1] = "pomegranate";	watermelon
\$fruits[2] = "tangerine";	grapes

## 2.4 Processing Arrays with Loops

- The process the data in PHP array with loop over it using any loop constructs.
- The for () loop is used through the array, extract the elements from it using index and display them one after other in an unordered list.
- The sizeof () function is to return the size of array.

## Example program :

```
<?php
    $shoppinglist=array('eye','wing','tail','leg');
    for ($x=0;$x<sizeof($shoppinglist);$x++)
        {
            echo "<li>$shoppinglist[$x]";
        }
?>
```

### OUTPUT:

- Eye
- Wing
- Tail
- leg

### 1.The foreach() Loop

- The foreach() loop runs once for each element of array, moving forward through the array on each iteration.
- On each run, the statements within curly braces are executed and the currently selected array element is made available through a loop variable.
- Foreach() loop doesn't need a counter or call to sizeof().
- It keeps track of its position in the array automatically.

### Example:

```
<?php
$shoppinglist=array('eye','wing','tail','leg');
foreach ($shoppinglist as $item)
{
```

```
echo "<li>$item";  
}  
?>
```

**OUTPUT:**

```
. eye  
. wing  
. tail  
. leg
```

## **2.5 Grouping Form Selections with Arrays**

- Uses arrays and loops also come in handy when processing form in PHP.
- If you have a group of related checkboxes and a multiselect list, you can use an array to capture all the selected form values in a single variable.

## Example

```
<?php
If (isset($_POST['submit']))
{
    Foreach($_POST['option']
            as $o)
    {
        Echo "<i>$o</i>";
    }
}
Else
{
    If(is_array($_POST['option']))
    {
        Echo "nothing selected";
    }
}
```

### Using Array Functions

- The `array_keys()` and `array_values()` functions come in handy to get a list of all keys and values within array.
- The `print_r()` function prints the information about a variable in a more human-readable way.

The following examples

```
<?php
//define an array
$a=array("Volvo"=>"XC90","BMW"=>"X5","Toyota"=>"Highlander");
echo "array key <br>";
print_r(array_keys($a));
echo "<br>array values<br>";
print_r(array_values($a));
```

?>

## OUTPUT:

array key

```
Array ( [0] => Volvo [1] => BMW [2] => Toyota )
```

array values

```
Array ( [0] => XC90 [1] => X5 [2] => Highlander )
```

- The `is_array()` function checks whether a variable is an array or not.
- This function returns true (1) if the variable is an array, otherwise it returns false/nothing.

## Syntax

```
is_array(variable);
```

*Variable* <sup>3</sup>Required. Specifies the variable to check

## Example

```
<?php
$a=array("Volvo"=>"XC90","BMW"=>"X5","Toyota"=>"Highlander");
echo is_array($a);
?>
```

## OUTPUT:

1

- The `list()` function assigns array elements to variables.
- **Example** of `list()` function

```
<?php
$a=array("Volvo","BMW","Toyota");
list($a1,$a2,$a3)=$a;
echo $a2;
?>
```

## OUTPUT:

### BMW

- The extract() function imports variables into the local symbol table from an array.
- This function uses array keys as variable names and values as variable values.

- **Syntax**

extract(array)

Array --> Required. Specifies the array to use

### Example:

```
<?php
$my_array = array("a" => "Cat","b" =>
"Dog", "c" => "Horse");
extract($my_array);
echo "\$a = $a; \$b = $b; \$c = $c";
?>
```

**Output:** \$a = Cat; \$b = Dog; \$c = Horse

## 2.6 Creating User Defined Functions

- A **Function** is nothing but a 'block of statements' which generally performs a specific task and can be used repeatedly in our program.
- This 'block of statements' is also given a **name** so that whenever we want to use it in our program/script, we can call it by its **name**.
- In PHP there are thousands of built-in functions which we can directly use in our program/script.
- PHP also supports **user defined functions**, where we can define our own functions.
- we can define our own functions in our program and use those functions.
- **Syntax:**

```
function function_name()
{
    // function code statements
}
```

### ➤ Few Rules to name Functions

1. A **function name** can only contain alphabets, numbers and underscores. No other special character is allowed.
2. The name should start with either an alphabet or an underscore. It should not start with a number.
3. And last but not least, function names are not case-sensitive.
4. The opening curly brace { after the function name marks the start of the function code, and the closing curly brace } marks the end of function code.

**<?php**

**// defining the function**

```
function greetings()
{
    echo "Merry Christmas and a Very Happy New Year";
}
echo "Hey Martha <br/>";
```

**// invoking the function**

```
greetings();
echo "Hey Jon <br/>";
```

**// invoking the function again**

```
greetings();
```

**?>**

**Example program to defining and invoking functions**

**Output:**



Hey Martha

Merry Christmas and a Very Happy New Year

Hey Jon

Merry Christmas and a Very Happy New Year

## 2.7 Advantages of User-defined Functions

- **Reusable Code:** As it's clear from the example above, you write a function once and can use it for a thousand times in your program.
- **Less Code Repetition:** Rather than repeating all those lines of code again and again, we can just create a function for them and simply call the

function.

**Easy to Understand:** Using functions in your program, makes the code more readable and easy to understand.

### Using Arguments and Return Values

- We can even pass data to a function, which can be used inside the function block.
- This is done using arguments.
- An argument is nothing but a variable.
- Arguments are specified after the function name, in parentheses, separated by comma.
- When we define a function, we must define the number of arguments it will accept and only that much arguments can be passed while calling the function.

### Syntax:

```
<?php
```

```
/* we can have as many arguments as we
```

```
want to have in a function */
```

```
function function_name(argument1,
```

```
argument2)
```

```
{  
// function code statements  
}  
?>  
Example  
<?php  
// defining the function with argument  
function greetings($festival)  
{  
echo "Wish you a very Happy $festival";  
}  
echo "Hey Jai <br/>";  
// invoking the function  
greetings("Diwali");  
// next line echo "<br/>";  
echo "Hey Jon <br/>";  
// invoking the function again  
greetings("New Year");  
?>
```

Output:

Hey Jai

Wish you a very Happy Diwali

Hey Jon

Wish you a very Happy New Year

## **2.8 Default Function Arguments**

➤ Sometimes function arguments play an important role in the function code execution.

- In such cases, if a user forgets to provide the argument while calling the function, it might lead to some error.
- To avoid such errors, we can provide a default value for the arguments which is used when no value is provided for the argument when the function is called.
- **Example**

**<?php**

```
// defining the function with default argument function
```

```
greetings($festival = "Life")
```

```
{
```

```
echo "Wish you a very Happy $festival";
```

```
}
```

```
echo "Hey Jai <br/>";
```

```
greetings("Diwali");
```

```
echo "<br/>";
```

```
echo "Hey Jon <br/>";
```

```
greetings();
```

```
?>
```

## **2.9 Function Overloading**

- Function overloading allows you to have multiple different variants of one function, differentiated by the number and type of arguments they take.
- For example, we defined the function add() which takes two arguments, and return the sum of those two. What if we wish to provide support for adding 3 numbers.

### **Example**

**<?php**

```
// add function with 2 arguments
```

```
function add($a, $b)
```

```
{
$sum = $a + $b;
return $sum;
}
function add1($a, $b, $c)
{
$sum1 = $a + $b + $c;
return $sum1;
}
echo "5 + 10 = " . add(5, 10) . "<br/>";
// calling add with 3 arguments
echo "5 + 10 + 15 = " . add1(5, 10, 15) . "<br/>";
?>
```

Output:

5 + 10 = 15

5 + 10 + 15 = 30

## 2.10 Using Files

- File handling is an important part of any web application. You often need to open and process a file for different tasks.
- PHP has several functions for
  - i. fopen()** function is unable to open the specified file.
  - ii. fwrite()** function is used to write to a file.
  - iii. fread()** function reads from an open file.
  - iv. fclose()** function is used to close an open file.

### **i. fopen():**

**Syntax:**

**fopen(filename, mode)**

- **Filename-** Required. Specifies the file or URL to open

- **Mode** - Required. Specifies the type of access you require to the file/stream. There are different types of mode are listed below:

Mode	Description
r	<ul style="list-style-type: none"> <li>• Read file from beginning.</li> <li>• Returns false if the file doesn't exist.</li> <li>• Read only</li> </ul>
r+	<ul style="list-style-type: none"> <li>• Read file from beginning</li> <li>• Returns false if the file doesn't exist.</li> <li>• Read and write</li> </ul>
w	<ul style="list-style-type: none"> <li>• Write to file at beginning</li> <li>• truncate file to zero length</li> <li>• If the file doesn't exist attempt to create it.</li> <li>• Write only</li> </ul>
w+	<ul style="list-style-type: none"> <li>• Write to file at beginning, truncate file to zero length</li> <li>• If the file doesn't exist attempt to create it.</li> <li>• Read and Write</li> </ul>
a	<ul style="list-style-type: none"> <li>• Append to file at end</li> <li>• If the file doesn't exist attempt to create it.</li> <li>• Write only</li> </ul>
a+	<ul style="list-style-type: none"> <li>• <u>Php</u> append to file at end</li> <li>• If the file doesn't exist attempt to create it</li> <li>• Read and write</li> </ul>

## ii. fread()

### Syntax

**fread** (*file*, *length*)

- **File** - Required. Specifies the open file to read from
- **length** - Required. Specifies the maximum number of bytes to read

**Ex:** fread (\$file,"10");

```
<?php
```

```
$myfile =
```

```
fopen("webdictionary.txt", "r")
```

```
echo fread($myfile,filesize("webdictionary.txt"));
```

```
fclose($myfile);
```

```
?>
```

### iii. fwrite()

#### Syntax:

**fwrite(*file*, *string*, *length*)**

- **File** - Required. Specifies the open file to write to
- **string** - Required. Specifies the string to write to the open file
- **length** -Optional. Specifies the maximum number of bytes to write
- Ex: echo fwrite(\$file,"Hello World. Testing!");

### iv. fclose()

#### Syntax:

**fclose(*file*)**

- **file** - Required. Specifies the file to close

**Ex:** fclose(\$file);

#### Example:

```
<?php
```

```
$file = fopen("test.txt","w");
```

```
echo fwrite($file,"HelloWorld. Testing! ");
```

```
fclose($file);
```

```
?>
```

## 2.11 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.
- 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`.
- The `session_start()` function must be the very first thing in your document. Before any HTML tags.

### **Example**

```
<?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>
```

### **Get PHP Session Variable Values**

- 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()).

- All session variable values are stored in the global \$\_SESSION variable.

### **Example**

```
<?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>
```

### **Destroy a PHP Session**

- To remove all global session variables and destroy the session, use session\_unset() and session\_destroy():

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
// remove all session variables
```



```
session_unset();  
// destroy the session  
session_destroy();  
?>  
</body>  
</html>
```

## 2.12 Cookie

- 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 use both create and retrieve cookie values.
- **Create Cookies With PHP**
- A cookie is created with the `setcookie()` function.
- **Syntax:**
- `setcookie(name, value, expire, path, domain, secure);`
- Only the *name* parameter is required. All other parameters are optional.
- Here is the detail of all the arguments -
- **Name** - Set of name and values of the cookie
- **Value** - Sets the value of the named variable and is the content that you actually want to store.
- **Expiry** - Sets the date and time at which the cookie expires.
- **Path** - This specifies the directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories.
- **Domain** - This can be used to specify the domain name in very large domains and must contain at least two periods to be valid.

- **Security** - This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

- Example

```
<?php
setcookie("name", "John Watkin", time()+3600, "/", "", 0);
setcookie("age", "36", time()+86400, "/", "", 0);
?>
```

#### **OUTPUT:**

**Time: 60 sec \* 60 mins = 3600**

**Time: 60sec \* 60 mins\* 24 hours = 86,400**

#### **Retrieving Cookie Data**

- Once cookie has been sent for a domain, it becomes available in the special `$_COOKIE` array, and its value may be accessed using array notation.

#### **Example:**

```
<? Php
If($_COOKIE['name'])
{
Echo "Welcome to " . $_COOKIE['name'];
}
Else
{
Echo "Cookie not found";
}
```

#### **Deleting Cookie**

- To delete a cookie, simply use `setcookie()` with its name to set the cookie's expiry date to a value in the past.

```
<?php
Setcookie(`name`,``, time()-10000,`/`);
?>
```

## 2.13 Dealing with Dates and Times

- The date/time functions allow you to get the date and time from the server where your PHP script runs.
- You can then use the date/time functions to format the date and time in several ways.
- The PHP date() function is used to format a date and/or a time.
- The PHP date() function formats a timestamp to a more readable date and time.

- **Syntax :**

*(format,timestamp)*

format - Required. Specifies the format of the timestamp

timestamp - Optional. Specifies a timestamp. Default is the current date and time.

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

### Get a Date

- The required *format* parameter of the date() function specifies how to format the date (or time).
- 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.

- **The example below formats today's date in three different ways:**

```
<?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");
?>
```

### **Output:**

Today is 2020/10/07

Today is 2020.10.07

Today is 2020-10-07

Today is Wednesday

### **Create a Date With mktime()**

- The optional *timestamp* parameter in the date() function specifies a timestamp. If omitted, the current date and time will be used.
- The PHP mktime() function returns the Unix timestamp for a date. The Unix timestamp contains the number of seconds between the Unix Epoch

(January 1 1970 00:00:00 GMT) and the time specified.

- **Syntax**

mktime(*hour, minute, second, month, day, year*)

- The example below creates a date and time with the date() function from a number of parameters in the mktime() function:

#### ➤ **Example**

```
<?php
$d=mktime(11, 14, 54, 8, 12, 2014);
echo "Created date is " . date("Y-m-d h:i:sa", $d);
?>
```

## ➤ **getdate() Function**

- The getdate() function returns date/time information of a timestamp or the current local date/time.
- Syntax

`getdate(timestamp)`

*Timestamp* - Optional. Specifies an integer Unix timestamp. Default is the current local time (time())

**Return Value:** Returns an associative array with information related to the timestamp:

- [seconds] - seconds
- [minutes] - minutes
- [hours] - hours
- [mday] - day of the month
- [wday] - day of the week (0=Sunday, 1=Monday,...)
- [mon] - month
- [year] - year
- [yday] - day of the year
- [weekday] - name of the weekday
- [month] - name of the month
- [0] - seconds since Unix Epoch

## **Example**

```
<?php
$current = getdate();
$current_time=$current['hours'] . ':' . $current['minutes'] .
':' . $current['seconds'];
$current_date=$current['mday'] . ':' . $current['mon'] . ':' .
```

```
$current['year'];  
echo "Time $current_time</br>";  
echo "Date $current_date";  
?>
```

### Output:

```
Time 2:27:15  
Date 7:10:2020
```

## 2.14 Executing External Programs

- To run an external program from your Php script, place the program command line within back tricks (` `)
- The output of the command can also be assigned to a variable for further use within the script.
- Example which runs the unix du command to calculate disk usage:

```
<?php  
$output = `/bin/du -s/tmp`;  
echo $output;  
?>
```

- **Escapeshellcmd** - Escape shell metacharacters

Syntax

**escapeshellcmd** ( string \$command )

- **escapeshellcmd()** escapes any characters in a string that might be used to trick a shell command into executing arbitrary commands.
- This function should be used to make sure that any data coming from user input is escaped before this data is passed to the `exec()` or `system()` functions, or to the backtick operator.

*Command* -- >The command that will be escaped.

- Example

```
<?php
```

```
$escaped_command = escapeshellcmd($command);  
system($escaped_command);
```

?>

- **Escapeshellarg** - Escape a string to be used as a shell argument
- Syntax

**escapeshellarg** ( string \$arg )

- **escapeshellarg()** adds single quotes around a string and quotes/escapes any existing single quotes allowing you to pass a string directly to a shell function and having it be treated as a single safe argument.
- This function should be used to escape individual arguments to shell functions coming from user input.
- *Arg* - The argument that will be escaped.

## UNIT-3

### FILE HANDLING

#### 3.1 Opening files using fopen

- **fopen()** function is Used to open the specified file.

**fopen(filename, mode)**

- **Filename**- Required. Specifies the file or URL to open
- **Mode** - Required. Specifies the type of access you require to the file/stream.
- There are different types of mode are listed below:

Mode	Description
r	<ul style="list-style-type: none"><li>• Read file from beginning.</li><li>• Returns false if the file doesn't exist.</li><li>• Read only</li></ul>
r+	<ul style="list-style-type: none"><li>• Read file from beginning</li><li>• Returns false if the file doesn't exist.</li><li>• Read and write</li></ul>
w	<ul style="list-style-type: none"><li>• Write to file at beginning</li><li>• truncate file to zero length</li><li>• If the file doesn't exist attempt to create it.</li><li>• Write only</li></ul>
w+	<ul style="list-style-type: none"><li>• Write to file at beginning, truncate file to zero length</li><li>• If the file doesn't exist attempt to create it.</li><li>• Read and Write</li></ul>
a	<ul style="list-style-type: none"><li>• Append to file at end</li><li>• If the file doesn't exist attempt to create it.</li><li>• Write only</li></ul>
a+	<ul style="list-style-type: none"><li>• <u>Php</u> append to file at end</li><li>• If the file doesn't exist attempt to create it</li><li>• Read and write</li></ul>



### 3.2 Looping over a files content with

#### Feof()

- The feof() function is used for looping through the content of a file if the size of content is not known beforehand.
- The feof() function returns True if end-of-file has been reached or if an error has occurred. Else it returns False.
- **Syntax:**

feof( \$file )

**Parameters:** The feof() function in PHP accepts only one parameter which is \$file. This parameter specifies the file which has to be checked for end-of-file.

**Return Value:** It returns TRUE if end-of-file has been reached or if an error has occurred. Else it returns False.

### 3.3 Reading text from a file using fgets

#### fgets() function

- The fgets() function in PHP is an inbuilt function which is used to return a line from an open file.
- It is used to return a line from a file pointer and it stops returning at a specified length, on end of file(EOF) or on a new line, whichever comes first.
- The file to be read and the number of bytes to be read are sent as parameters to the fgets() function and it returns a string of length -1 bytes from the file pointed by the user.
- It returns False on failure.
- **Syntax:**

fgets(file, length)

**Parameters Used:** The fgets() function in PHP accepts two parameters.

**file :** It specifies the file from which characters have to be extracted.

**length :** It specifies the number of bytes to be read by the fgets() function. The default value is 1024 bytes.

**Return Value :** It returns a string of length -1 bytes from the file pointed by the user or False on failure.

### 3.4 Closing A File

- The fclose() function closes an open file.
- Syntax

fclose(*file*)

Parameter Values :

*File* - Required. Specifies the file to close

Example : Open and close file "test.txt":

```
<?php
$file = fopen("test.txt", "r");
fclose($file);
?>
```

#### **EXAMPLE:**

```
<?php
// a file is opened using fopen() function
$check = fopen("singleline.txt", "r");
$seq = fgets($check);
// Outputs a line of the file until
// the end-of-file is reached
while(! feof($check))
{
echo $seq ;
$seq = fgets($check);
}
// file is closed using fclose() function
```

```
fclose($check);
```

```
?>
```

**OUTPUT:**

This file consists of only a single line.

### **Example**

The singleline.txt contain ” hai how are you?”

```
<html>
```

```
<body>
```

```
<?php
```

```
$check = fopen("singleline.txt","r");
```

```
while(!feof($check))
```

```
{
```

```
echo fgets($check);
```

```
}
```

```
fclose($check);
```

```
?>
```

```
<html>
```

```
<body>
```

### **3.5 READING CHARACTER WITH fgets () IN PHP**

- The fgets() function in PHP is an inbuilt function which is used to return a single character from an open file. It is used to get a character from a given file pointer.
- The file to be checked is used as a parameter to the fgets() function and it returns a string containing a single character from the file which is used as a parameter.

#### **Syntax:**

```
fgets($file)
```

**Parameters:** The `fgetc()` function in PHP accepts only one parameter *\$file*. It specifies the file from which character is needed to be extracted.

**Return Value:** It returns a string containing a single character from the file which is used as a parameter.

### Example

- The file named *gfg.txt* contains the following text.

This is the first line.

This is the second line.

This is the third line.

### Program:

```
<?php
$my_file = fopen("gfg.txt", "rw");
echo fgetc($my_file);
fclose($my_file);
?>
```

### Output:

T

### Program:

```
<?php
$my_file = fopen("gfg.txt", "rw");
while (! feof ($my_file))
{
echo fgetc($my_file);
}
fclose($my_file);
?>
```

### Output:

**This is the first line.**

**This is the second line.**

**This is the third line.**

### **3.6 file\_get\_contents() Function**

- The file\_get\_contents() function in PHP is an inbuilt function which is used to read a file into a string.
- The function uses memory mapping techniques which are supported by the server and thus enhances the performances making it a preferred way of reading contents of a file.

The path of the file to be read is sent as a parameter to the function and it returns the data read on success and FALSE on failure.

#### **Syntax:**

**file\_get\_contents(\$path, \$include\_path, \$context, \$start, \$max\_length)**

- **\$path:** It specifies the path of the file or directory you want to check.
- **\$include\_path:** It is an optional parameter which searches for a file in the file in the include\_path (in php.ini) also if it is set to 1.
- **\$context:** It is an optional parameter which is used to specify a custom context.
- **\$start:** It is an optional parameter which is used to specify the starting point in the file for reading.
- **\$max\_length:** It is an optional parameter which is used to specify the number of bytes to be read.
- **Return Value:** It returns the read data on success and FALSE on failure.

#### **Example**

- The test.txt contain the data:

```
abcdefghijklmnopqrstuvwxyz0123456789
```

```
<?php
```

```
echo file_get_contents("test.txt");
```

```
?>
```

**Output:**

```
abcdefghijklmnopqrstuvwxy0123456789
```

```
<html>
```

```
<body>
```

```
<?php
```

```
// Read 14 characters starting from the 21st
```

```
character
```

```
echo file_get_contents("test.txt", FALSE,
```

```
NULL, 20, 14);
```

```
?>
```

```
</body>
```

```
</html>
```

**Output:**

```
vwxyz01234567
```

**3.7 file\_exists() Function**

- The file\_exists() function in PHP is an inbuilt function which is used to check whether a file or directory exists or not.
- The path of the file or directory you want to check is passed as a parameter to the file\_exists() function which returns True on success and False on failure.
- **Syntax:**

```
file_exists($path)
```

**Parameters:** The file\_exists() function in PHP accepts only one parameter *\$path*.

It specifies the path of the file or directory you want to check.

**Return Value:** It returns True on success and False on failure.

Example:

```
<?php
```

```
// checking whether file exists or not
```

```
echo file_exists("test.txt");
```

```
?>
```

**Output : 1**

```
<?php
```

```
$myfile = "test.txt";
```

```
if (file_exists($myfile)) {
```

```
echo "$myfile exists!";
```

```
} else {
```

```
echo "$myfile does not exist!";
```

```
}
```

```
?>
```

**Output:** test.txt exists

### 3.8 fscanf() function

➤ The fscanf() function parses input from an open file according to a specified format. It returns the values parsed as an array, if only two parameters were passed.

#### Syntax

```
fscanf(file_pointer, format, mixed)
```

#### Parameters

**file\_pointer** - A file system pointer resource created using fopen().

**format** - Specify the format. Here are the values:

- %% - Returns a percent
- %b - Binary number
- %c - The character according to the ASCII value
- %f - Floating-point number
- %F - Floating-point number
- %o - Octal number
- %s - String
- %d - Signed decimal number
- %e - Scientific notation
- %u - Unsigned decimal number
- %x - Hexadecimal number for lowercase letters
- %X - Hexadecimal number for uppercase letters
- **mixed** – Specify the assigned values. Optional.

### **Return :**

The fscanf() function returns the values parsed as an array, if only two parameters were passed.

### **Example**

- **The new.txt contain the text :**

ram 5

siva 3

```
<?php
```

```
$handle = fopen("new.txt", "r");
```

```
while ($playerrank = fscanf($handle, " %s\t%d\n "))
```

```
{
```

```
list ($name, $rank) = $playerrank;
```



```
echo "$name got rank $rank.<br>";  
}  
fclose($handle);  
?>
```

**Output:**

**ram got rank 5.**

**siva got rank 3.**

### **3.9 parse\_ini\_file() Function**

➤ The parse\_ini\_file() function parses a configuration (ini) file and returns the settings.

➤ This function can be used to read in your own configuration files, and has nothing to do with the php.ini file.

➤ The following reserved words must not be used as keys for ini files: null, yes, no,true, false, on, off, none. Furthermore, the following reserved characters must not be used in the key: {}|&~!()^".

➤ **Syntax**

*parse\_ini\_file(file, process\_sections, scanner\_mode)*

Parameter	Description
<i>file</i>	Required. Specifies the ini file to parse
<i>process_sections</i>	Optional. If set to TRUE, it returns is a multidimensional array with section names and settings included. Default is FALSE
<i>scanner_mode</i>	Optional. Can be one of the following values: <ul style="list-style-type: none"> <li>• INI_SCANNER_NORMAL (default)</li> <li>• INI_SCANNER_RAW (means option values will not be parsed)</li> <li>• INI_SCANNER_TYPED (means that boolean, null and integer types are preserved when possible. "true", "on", "yes" are converted to TRUE. "false", "off", "no", "none" are converted to FALSE. "null" is converted to NULL. Numeric strings are converted to integer type if possible)</li> </ul>

**Return-** The `parse_ini_file()` function returns the settings as an associative array success. It returns FALSE on failure.

➤ Let's say the content of our "demo.ini" is -

[names]

one = Anne

two = Katie

three = Tom

[urls]

host1 = "https://www.example.com"

host2 = "https://www.example2.com"

„H Let's say the content of our "parse ini.php" is

```
<?php
```

```
print_r(parse_ini_file("demo.ini"));
```

```
?>
```

**Output:**

Array ( [one] => Anne

[two] => Katie

[three] => Tom

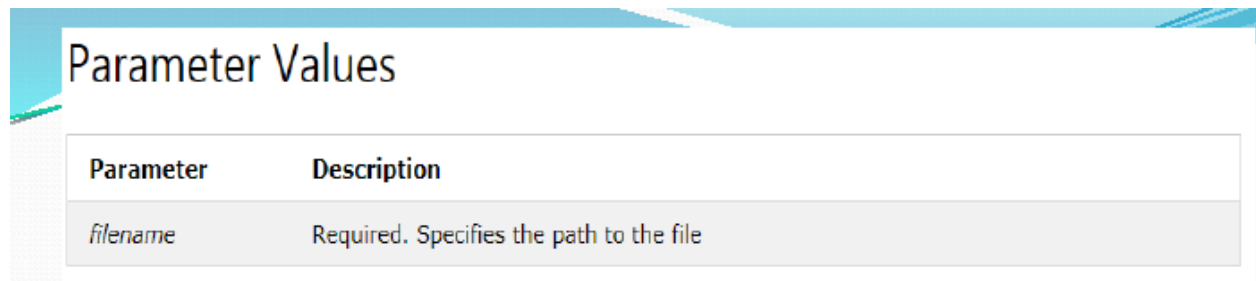
[host1] => https://www.example.com

[host2] => https://www.example2.com )

### 3.10 Getting file information with stat()

- The stat() function returns information about a file.
- The results from this function will differ from server to server.
- The array may contain the number index, the name index, or both.
- The result of this function is cached.
- Use clearstatcache() to clear the cache.
- **Syntax**

*stat(filename)*



Parameter	Description
<i>filename</i>	Required. Specifies the path to the file

- The function returns an array with the below given elements.
- [0] or [dev] - Device number
- [1] or [ino] - Inode number
- [2] or [mode] - Inode protection mode
- [3] or [nlink] - Number of links
- [4] or [uid] - User ID of owner
- [5] or [gid] - Group ID of owner
- [6] or [rdev] - Inode device type
- [7] or [size] - Size in bytes

- [8] or [atime] - Last access time as Unix timestamp
- [9] or [mtime] - Last modified time as Unix timestamp
- [10] or [ctime] - Last inode change time as Unix timestamp
- [11] or [blksize] - Blocksize of filesystem IO
- [12] or [blocks] - Number of blocks allocated

Example:

```
<?php
print_r(stat("demo.txt"));
?>
```

**OUTPUT:**

```
Array
(
    [0] => 0
    [1] => 0
    [2] => 33206
    [3] => 1
    [4] => 0
    [5] => 0
    [6] => 0
    [7] => 120
    [8] => 17128173529
    [9] => 1984185875
    [10] => 1294322653
    [11] => -1
    [12] => -1
    [dev] => 0
    [ino] => 0
    [mode] => 33206
    [nlink] => 1
    [uid] => 0
    [gid] => 0
    [rdev] => 0
    [size] => 120
    [atime] => 1718173529
    [mtime] => 1984185875
    [ctime] => 1294322653
    [blksize] => -1
    [blocks] => -1
)
```

Example:

```
<?php
```

```
$test = stat('new.txt');  
echo 'Access time: ' . $test['atime'];  
echo '<br />Modification time: ' . $test['mtime'];  
echo '<br />Device number: ' . $test['dev'];  
?>
```

### **Output:**

```
Access time: 1602748187  
Modification time: 1603208669  
Device number: 2
```

### **3.11 fseek( ) Function**

- The *fseek()* function in PHP is an inbuilt function which is used to seek in an open file.
- It moves the file pointer from its current position to a new position, forward or backward specified by the number of bytes.
- It returns 0 on success, else returns -1 on failure.
- **Syntax:**

```
fseek ( $file, $offset, $whence)
```

**Parameters:** The *fseek()* function in PHP accepts three parameters as described below.

***\$file:*** It is a mandatory parameter which specifies the file.

***\$offset:*** It is a mandatory parameter which specifies the new position of the pointer. It is measured in bytes from the beginning of the file.

***\$whence:*** It is an optional parameter which can have the following possible values-

SEEK\_SET: It sets position equal to offset.

SEEK\_CUR: It sets position to current location plus offset.

SEEK\_END: It sets position to EOF plus offset. To move to a position before EOF, the offset must be a negative value.

**Example:**

```
<?php
// Opening a file
$myfile = fopen("gfg.txt", "w");
// reading first line
fgets($myfile);
// moving back to the beginning of the file
echo fseek($myfile, 0);
// closing the file
fclose($myfile);
?>
```

**Output: 0****3.12 Copying Files With Copy**

- The copy() function in PHP is an inbuilt function which is used to make a copy of a specified file.
- It makes a copy of the source file to the destination file and if the destination file already exists, it gets overwritten.
- The copy() function returns true on success and false on failure.

**Syntax:**

*copy(from\_file, to\_file, context)*

Parameter	Description
<i>from_file</i>	Required. Specifies the path to the file to copy from
<i>to_file</i>	Required. Specifies the path to the file to copy to
<i>context</i>	Optional. Specifies a context resource created with <code>stream_context_create()</code>

**Example:**

```
<?php
echo
copy("test.txt","new.txt");
?>
```

**Output:**

```
1
<?php
// Copying gfg.txt to geeksforgeeks.txt
$srcfile = "test.txt";
$destfile = "copy.txt";
if (!copy($srcfile, $destfile))
{
echo "File cannot be copied! \n";
}
else
{
echo "File has been copied!";
}
?>
```

Output: File has been copied!

### 3.13 Deleting files

- To delete a file by using PHP is very easy.
- Deleting a file means completely erase a file from a directory so that the file is no longer exist.

PHP has an unlink () function is used to delete a file.

#### Syntax

```
unlink( $filename, $context );
```

#### Example

```
<?php
$myFile = "test5.txt";
unlink($myFile) or die("Couldn't delete file");
?>

<?php
$file_pointer = fopen("gfg.txt", "w");
fwrite($file_pointer, 'A computer science portal
for geeks!');
fclose($file_pointer);
if (!unlink($file_pointer))
{
echo (" $file_pointer cannot be deleted due to an
error");
}
else {
echo (" $file_pointer has been deleted");
} ?>
```

Output:

gfg.txt has been deleted



### 3.14 Reading and Writing Binary Files

- With the basics of reading and writing text files complete, let's now turn our attention to working with binary files.
- Unlike text files, binary files can be much harder both to work with and debug because they are by their very nature unreadable by anything but a computer.
- In PHP, writing binary files is done in the same manner as writing text files (via the `fputs()` function) and therefore requires no explanation.
- In fact, the only difference (which has already been mentioned) is the use of the `b` mode when the file is opened via `fopen()`.
- Hence, this section will focus primarily on those functions relating to reading binary data from a file and converting it into a form usable by PHP.
- Specifically, we will be constructing a function that will read the header from a Zip-compressed file and determine the minimum version number required to decompress the data.
- To accomplish this, we'll be examining the `fseek()`, `fread()`, and `unpack()` functions.
- Already `fseek()`, `fread()`, is given write that notes here.

#### **Unpack() function:**

- The `unpack()` function is an inbuilt function in PHP which is used to unpack from a binary string into the respective format.
- **Syntax:**

```
unpack( $format, $data, $offset )
```

Parameter	Description
<i>format</i>	<p>Required. Specifies the format to use when unpacking data.</p> <p>Possible values:</p> <ul style="list-style-type: none"> <li>• a - NUL-padded string</li> <li>• A - SPACE-padded string</li> <li>• h - Hex string, low nibble first</li> <li>• H - Hex string, high nibble first</li> <li>• c - signed char</li> <li>• C - unsigned char</li> <li>• J - unsigned long long (always 64 bit, big endian byte order)</li> <li>• P - unsigned long long (always 64 bit, little endian byte order)</li> <li>• f - float (machine dependent size and representation)</li> <li>• g - float (machine dependent size, little endian byte order)</li> <li>• G - float (machine dependent size, big endian byte order)</li> <li>• d - double (machine dependent size and representation)</li> <li>• e - double (machine dependent size, little endian byte order)</li> <li>• E - double (machine dependent size, big endian byte order)</li> <li>• x - NUL byte</li> <li>• X - Back up one byte</li> <li>• Z - NUL-padded string</li> <li>• @ - NUL-fill to absolute</li> </ul>
<i>data</i>	Required. Specifies the binary data to be unpacked
<i>offset</i>	Optional. Specifies where to start unpacking from. Default is 0.

### Example

```
<?php
var_dump ( unpack("C*", "GEEKSFORGEEKS"));
?>
```

### Output:

```
array(13) { [1]=> int(71) [2]=> int(69) [3]=> int(69)
[4]=> int(75) [5]=> int(83) [6]=> int(70) [7]=>
int(79) [8]=> int(82) [9]=> int(71) [10]=> int(69)
[11]=> int(69) [12]=> int(75) [13]=> int(83) }
```

### 3.15 Locking files

- The flock() function locks and releases a file.

#### Syntax

`flock(file, lock, block)`

## Parameter Values

Parameter	Description
<i>file</i>	Required. Specifies an open file to lock or release
<i>lock</i>	Required. Specifies what kind of lock to use.  Possible values: <ul style="list-style-type: none"><li>• LOCK_SH - A shared lock (reader). Allow other processes to access the file</li><li>• LOCK_EX - An exclusive lock (writer). Prevent other processes from accessing the file</li><li>• LOCK_UN - Release the lock</li><li>• LOCK_NB - Avoid blocking other processes while locking</li></ul>
<i>block</i>	Optional. Set to 1 to block other processes while locking

### Example

```
<?php
$file = fopen("test3.txt","w+");
if (flock($file,LOCK_EX)) {
fwrite($file,"Add some text to the file.");
fflush($file);
// release lock
echo "successfully done";
}
else
{
echo "Error locking file!";
}
fclose($file);
?>
```

## Unit – 4

### MySQL

- With PHP, you can connect to and manipulate databases.
- MySQL is the most popular database system used with PHP.
- The data in a MySQL database are stored in tables.
- A table is a collection of related data, and it consists of columns and rows.

#### What is MySQL?

- 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 developed, distributed, and supported by Oracle Corporation
- MySQL is named after co-founder Monty Widenius's daughter: My

#### 4.1 Effectiveness of MySQL

- MySQL is a free-to-use, open-source database that facilitates effective management of databases by connecting them to the software.

- It is a stable, reliable and powerful solution with advanced features like the following:

### **1. Data Security**

- MySQL is globally renowned for being the most secure and reliable database management system used in popular web applications like WordPress, Facebook ,Twitter etc.
- The data security and support for transactional processing that accompany the recent version of MySQL, can greatly benefit any business especially if it is an eCommerce business that involves frequent money transfers.

### **2. On-Demand Scalability**

- MySQL offers unmatched scalability to facilitate the management of deeply embedded apps using a smaller footprint even in massive warehouses that stack terabytes of data.
- On-demand flexibility is the star feature of MySQL. This open source solution allows complete customization to eCommerce businesses with unique database server requirements.

### **3. High Performance**

- MySQL features a distinct storage-engine framework that facilitates system administrators to configure.
- Whether it is an eCommerce website that receives a million queries every single day or a high-speed transactional processing system.

### **4. Round-the-clock Uptime**

- MySQL comes with the assurance of 24X7 uptime and offers a wide range of high availability solutions like specialized cluster servers and master/slave replication configurations.

## **5. Comprehensive Transactional Support**

- MySQL tops the list of robust transactional database engines available on the market.
- With features like complete atomic, consistent, isolated, durable transaction support, multi-version transaction support, and unrestricted row-level locking, it is the go-to solution for full data integrity.
- It guarantees instant deadlock identification through server-enforced referential integrity.

## **6. Complete Workflow Control**

- With the average download and installation time being less than 30 minutes, MySQL means usability from day one.
- Whether your platform is Linux, Microsoft, Macintosh or UNIX, MySQL is a comprehensive solution with self-management features that automate everything from space expansion and configuration to data design and database administration.

## **7. Reduced Total Cost of Ownership**

- By migrating current database apps to MySQL, enterprises are enjoying significant cost savings on new projects.

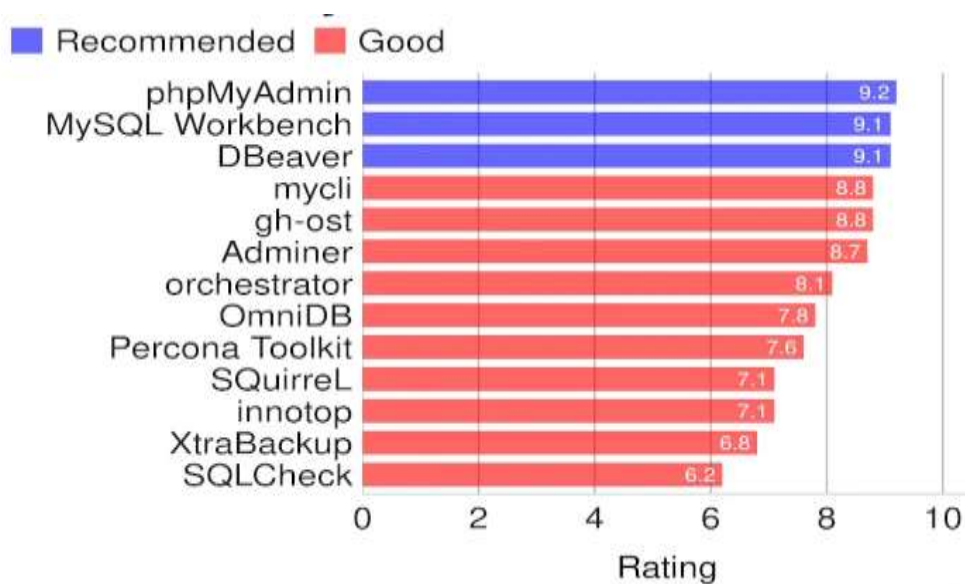
## **8. The Flexibility of Open Source**

- The secure processing and trusted software of MySQL combine to provide effective transactions for large volume projects.
- It makes maintenance, debugging and upgrades fast and easy while enhancing the end-user experience.

## **4.2 Tools**

- MySQL is a relational database management system. It provides a very fast, multi-threaded, multi-user, and robust SQL (Structured Query Language) database server.
- MySQL is the most popular open source database, and is the database component of the LAMP software stack. LAMP consists of the Apache web server, MySQL and PHP, the essential building blocks to run a general purpose web server.
- MySQL is used and championed by many large organizations including Google, Facebook, the BBC, Intel, Sun, SAP, Dell, AMD, Novell, Veritas and many others.
- With the increasing popularity of MySQL, it is not surprising that developers have written useful tools which help users to monitor, query, administer, troubleshoot, and optimize MySQL databases.
- This article identifies 13 open source tools which help to reduce the complexity associated with the powerful database software.
- One tool which is not featured in this article but warrants a mention is sqlyog.

- It is an excellent utility to manage and administer MySQL.
- To provide an insight into the quality of software that is available, we have compiled a list of 13 excellent MySQL tools.
- Hopefully, there will be something of interest for anyone interested in managing MySQL databases with the minimum of fuss.



### **4.3 Prerequisites of MYSQL**

You must satisfy the following prerequisites to create a connection with the MySQL Adapter:

- Ensure that you have write permissions on the database.
- Ensure that you have the required permissions to run stored procedures and packages and SQL statements against the MySQL Database.
- Know the database hostname or IP address and the port number.
- Know the database name.
- Know the username and password for connecting to the database.



- Download the mysql-connector-java-commercial-5.1.22-bin.jar to the host on which the connectivity agent is installed.

1. Download the mysql-connector-java-commercial-5.1.22-bin.jar from the MySQL Database site. Check the Oracle Integration Adapters Certification matrix to identify the certified versions of the MySQL Database.
2. Copy the JAR file to agenthome/thirdparty/lib.
3. Restart the connectivity agent.

If you do not download and install this JAR file, you receive a Check agent status error when testing the connection on the Connections page in Oracle Integration.

#### **4.4 Databases**

- Databases are useful for storing information categorically. A company may have a database with the following tables:
  - Employees
  - Products
  - Customers
  - Orders
- PHP + MySQL Database System
- PHP combined with MySQL are cross-platform (you can develop in Windows and serve on a Unix platform)

#### **Database Queries**

- A query is a question or a request.
- We can query a database for specific information and have a recordset returned.

- Look at the following query (using standard SQL):  
SELECT LastName FROM Employees
- The query above selects all the data in the "LastName" column from the "Employees" table.

### **Tables:**

- Every database is composed of one or more *tables*.
- These tables, which structure data into rows and columns, are what lend organization to the data.
- Here's an example of what a typical table looks like:

```

+-----+-----+-----+
| mid | mtitle |      | myear |
+-----+-----+-----+
|  1 | Rear Window |      | 1954 |
|  2 | To Catch A Thief |      | 1955 |
|  3 | The Maltese Falcon |      | 1941 |
|  4 | The Birds |      | 1963 |
|  5 | North By Northwest |      | 1959 |
|  6 | Casablanca |      | 1942 |
|  7 | Anatomy Of A Murder |      | 1959 |
+-----+-----+-----+

```

- A table divides data into rows, with a new entry (or record) on every row.
- The data in each row is further broken down into columns (or fields), each of which contains a value for a particular attribute of that data.

### **4.5 MySQL - Data Types**

- You should use only the type and size of field you really need to use.

- For example, do not define a field 10 characters wide, if you know you are only going to use 2 characters.
- These type of fields (or columns) are also referred to as data types, after the **type of data** you will be storing in those fields.
- MySQL uses many different data types broken into three categories –

## 1. Numeric

- MySQL uses all the standard ANSI SQL numeric data types, so if you're coming to MySQL from a different database system, these definitions will look familiar to you.
- The following list shows the common numeric data types and their descriptions
  - **INT** – A normal-sized integer that can be signed or unsigned. You can specify a width of up to 11 digits.
  - **TINYINT** – A very small integer that can be signed or unsigned. If signed, the allowable range is from -128 to 127.
  - **SMALLINT** – A small integer that can be signed or unsigned. You can specify a width of up to 5 digits.
  - **MEDIUMINT** – A medium-sized integer that can be signed or unsigned. You can specify a width of up to 9 digits.
  - **BIGINT** – A large integer that can be signed or unsigned. You can specify a width of up to 20 digits.
  - **FLOAT(M,D)** – A floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D).

- **DOUBLE(M,D)** – A double precision floating-point number that cannot be unsigned. You can define the display length (M) and the number of decimals (D).
- **DECIMAL(M,D)** – An unpacked floating-point number that cannot be unsigned. Defining the display length (M) and the number of decimals (D) is required. NUMERIC is a synonym for DECIMAL.

## 2. Date and Time

➤ The MySQL date and time datatypes are as follows –

- **DATE** – A date in YYYY-MM-DD format, between 1000-01-01 and 9999-12-31.
- **DATETIME** – A date and time combination in YYYY-MM-DD HH:MM:SS format.
- **TIME** – Stores the time in a HH:MM:SS format.
- **YEAR(M)** – Stores a year in a 2-digit or a 4-digit format. If the length is specified as 2. If the length is specified as 4, then YEAR can be 1901 to 2155. The default length is 4.

## 3. String Types

➤ Although the numeric and date types are fun, most data you'll store will be in a string format. This list describes the common string datatypes in MySQL.

- **CHAR(M)** – A fixed-length string between 1 and 255 characters in length right-padded with spaces to the specified length when stored. Defining a length is not required, but the default is 1.

- **VARCHAR(M)** – A variable-length string between 1 and 255 characters in length. For example, VARCHAR(25). You must define a length when creating a VARCHAR field.
- **ENUM** – An enumeration, which is a fancy term for list. When defining an ENUM, you are creating a list of items from which the value must be selected

#### **4.6 CREATE TABLE**

- To create a table in MySQL, a **CREATE TABLE statement** is used.
- This statement is very complex because it requires defining all columns, data types and other parameters that make a column.
- However, a table may be created with basic information only, and that is *column name(s)* and *data type(s)*.
- In such as case, other parameters will be set to their default values. Below you can see a basic syntax for MySQL Table Creation.

```
CREATE TABLE emp (  
emp_id INT AUTO_INCREMENT PRIMARY KEY,  
emp_name VARCHAR(255),  
salary INT  
);
```

- This SQL statement creates the table employee with three fields, or columns, with commas separating the information about each column.

- 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

### Insert Rows in table:

- 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 emp (emp_id, emp_name, salary)
VALUES (E01, 'Ram', 20000);
```

```
INSERT INTO emp (emp_id, emp_name, salary)
VALUES (E02, 'Siva', 50000);
```

Emp_id	Emp_name	salary
E01	Ram	20000
E02	Siva	50000

### **Update Data in table**

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

```
“UPDATE emp SET emp_name='rama' WHERE id=E01”
```

- The above examples update the record the emp\_name as Rama where id=E02 in the "emp" table.

Emp_id	Emp_name	salary
E01	Rama	20000
E02	Siva	50000

### **Delete Data in table**

- The Delete statement is used to remove the data in the table.

DELETE FROM emp WHERE id=E02

- The above statement is used to delete the data where the id= 2

Emp_id	Emp_name	salary
E01	Rama	20000

### **Retrieving data**

- The SQL **SELECT** command is used to fetch data from the MySQL database. You can use this command at mysql> prompt as well as in any script like PHP.
- Syntax  
SELECT field1, field2,...fieldN FROM table\_name1, table\_name2...  
[WHERE Clause]
- You can use one or more tables separated by comma to include various conditions using a WHERE clause, but the WHERE clause is an optional part of the SELECT command.
- You can fetch one or more fields in a single SELECT command.
- You can specify star (\*) in place of fields. In this case, SELECT will return all the fields.
- You can specify any condition using the WHERE clause.
- Example:

```
SELECT * FROM emp where emp_id=E01;
```



Emp_id	Emp_name	salary
E01	Rama	20000

## 4.7 Sorting And Filtering Retrieved Data

### Sorting Data:

- We have seen the SQL **SELECT** command to fetch data from a MySQL table.
- When you select rows, the MySQL server is free to return them in any order, unless you instruct it otherwise by saying how to sort the result.
- But, you sort a result set by adding an **ORDER BY** clause that names the column or columns which you want to sort.
- **Syntax:**
- The following code block is a generic SQL syntax of the **SELECT** command along with the **ORDER BY** clause to sort the data from a MySQL table.

```
SELECT field1, field2,...fieldN table_name1, table_name2...  
ORDER BY field1, [field2...] [ASC [DESC]]
```

- You can sort the returned result on any field, if that field is being listed out.
- You can sort the result on more than one field.
- You can use the keyword **ASC** or **DESC** to get result in ascending or descending order. By default, it's the ascending order.
- You can use the **WHERE...LIKE** clause in the usual way to put a condition.

➤ **Example:**

```
SELECT * FROM emp ORDER BY salary DESC';
```

- The above query display the data in the table emp orderby desc as follow.

Emp_id	Emp_name	salary
E02	Siva	50000
E01	Rama	20000

#### **4.8 Aggregate functions**

- An aggregate function performs a calculation on multiple values and returns a single value.
- For example, you can use the **AVG()** aggregate function that takes multiple numbers and returns the average value of the numbers.
- The following illustrates the syntax of an aggregate function:

```
function_name(DISTINCT | ALL expression)
```

- In this syntax:
- First, specify the name of the aggregate function e.g., **AVG()**. See the list of aggregate functions in the following section.
- Second, use **DISTINCT** if you want to calculate based on distinct values or **ALL** in case you want to calculate all values including duplicates. The default is **ALL**.

- Third, specify an expression that can be a column or expression which involves column and arithmetic operators.
- The aggregate functions are often used with the GROUP BY clause to calculate an aggregate value for each group e.g., the average value by the group or the sum of values in each group.

Aggregate function	Description
AVG()	Return the average of non-NULL values.
BIT_AND()	Return bitwise AND.
BIT_OR()	Return bitwise OR.
BIT_XOR()	Return bitwise XOR.
COUNT()	Return the number of rows in a group, including rows with NULL values.
GROUP_CONCAT()	Return a concatenated string.
JSON_ARRAYAGG()	Return result set as a single JSON array.
JSON_OBJECTAGG()	Return result set as a single JSON object.
MAX()	Return the highest value (maximum) in a set of non-NULL values.
MIN()	Return the lowest value (minimum) in a set of non-NULL values.

➤ **Example:**

```
SELECT name , AVG(salary) FROM emp;
```

- The COUNT() function returns the number of the value in a set.
- For example, you can use the COUNT() function to get the number of products in the products table as shown in the following query:

```
SELECT
COUNT(*) AS salary
```

**FROM**

emp;

- The SUM() function returns the sum of values in a set.
- The SUM() function ignores NULL. If no matching row found, the SUM() function returns NULL.
- To get the total order value of each product, you can use the SUM() function in conjunction with the GROUP BY clause as follows:

**SELECT**

**SUM(salary) total**

**FROM**

Emp

### **Grouping data**

- The GROUP BY clause groups a set of rows into a set of summary rows by values of columns or expressions.
- The GROUP BY clause returns one row for each group.
- In other words, it reduces the number of rows in the result set.
- You often use the GROUP BY clause with aggregate functions such as SUM, AVG, MAX, MIN, and COUNT.
- The aggregate function that appears in the SELECT clause provides information about each group.
- Syntax:

**SELECT**

c1, c2,..., cn, aggregate\_function(ci)

**FROM**

table

**WHERE**

where\_conditions

**GROUP BY** c1 , c2,...,cn;

- The **GROUP BY** clause must appear after the **FROM** and **WHERE** clauses.
- Following the **GROUP BY** keywords is a list of comma-separated columns or expressions that you want to use as criteria to group rows.
- Example:

**SELECT** Emp\_name **FROM** emp **GROUP BY** salary;

### **Subquery**

- A MySQL subquery is a query nested within another query such as **SELECT**, **INSERT**, **UPDATE** or **DELETE**. In addition, a subquery can be nested inside another subquery.
- A MySQL subquery is called an inner query while the query that contains the subquery is called an outer query. A subquery can be used anywhere that expression is used and must be closed in parentheses.
- The following query returns employees who work in offices located in the USA.

**SELECT**

lastName, firstName

**FROM**

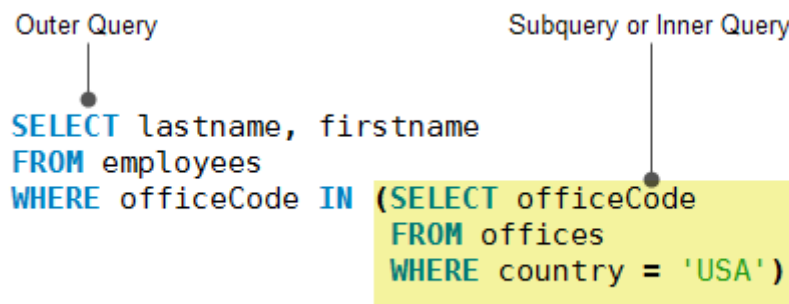
employees

**WHERE**

```
officeCode IN (SELECT
    officeCode
FROM
    offices
WHERE
    country = 'USA');
```

➤ In this example:

- The subquery returns all *office codes* of the offices located in the USA.
- The outer query selects the last name and first name of employees who work in the offices whose office codes are in the result set returned by the subquery.

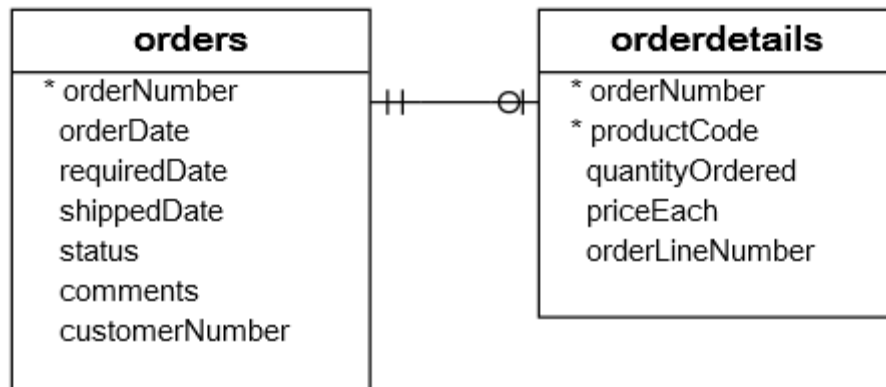


- When the query is executed, the subquery runs first and returns a result set. Then, this result set is used as an input for the outer query.

## 4.9 Joining Tables

- A relational database consists of multiple related tables linking together using common columns which are known as foreign key columns.
- Because of this, data in each table is incomplete from the business perspective.

- For example, in the sample database, we have the orders and orderdetails tables that are linked using the orderNumber column:



- A join is a method of linking data between one (self-join) or more tables based on values of the common column between the tables.
- MySQL supports the following types of joins:
- Inner join : The INNER JOIN matches each row in one table with every row in other tables and allows you to query rows that contain columns from both tables.

**Example:**

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
```

```
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

**Output:**

OrderID	CustomerName
10248	Wilman Kala
10249	Tradição Hipermercados
10250	Hanari Carnes

- Left join : The **LEFT JOIN** allows you to query data from two or more tables. Similar to the **INNER JOIN** clause, the **LEFT JOIN** is an optional clause of the **SELECT** statement, which appears immediately after the **FROM** clause.

**Example:**

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

**Output:**

CustomerName	OrderID
Alfreds Futterkiste	<i>null</i>
Ana Trujillo Emparedados y helados	10308
Antonio Moreno Taquería	10365

- Right join : MySQL **RIGHT JOIN** is similar to **LEFT JOIN**, except that the treatment of the joined tables is reversed.

**Example:**



```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

**Output:**

OrderID	LastName	FirstName
	West	Adam
10248	Buchanan	Steven
10249	Suyama	Michael

**4.10 Set operators**

- Set operations which can be performed on the table data.
- These are used to get meaningful results from data stored in the table, under different special conditions.
- In this tutorial, we will cover 4 different types of SET operations, along with example:

1. UNION
2. UNION ALL
3. INTERSECT
4. MINUS

**1. UNION Operator**

- The UNION operator is used to combine the result-set of two or more SELECT statements.

- Each SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in each SELECT statement must also be in the same order

➤ **UNION Syntax**

SELECT *column\_name(s)* FROM *table1*

UNION

SELECT *column\_name(s)* FROM *table2*;

**Example:**

The **First** table,

ID	Name
1	abhi
2	adam

The **Second** table,

ID	Name
2	adam
3	Chester

Select \* from First

Union

Select \* from second;

The result of the above query is

ID	NAME
1	abhi
2	adam
3	Chester

## 2. UNION ALL

- This operation is similar to Union. But it also shows the duplicate rows.

### Example:

Select \* from First

Union All

Select \* from second;

- The result of the above query is

ID	NAME
1	abhi
2	adam
2	adam
3	Chester

## 3. INTERSECT

- Intersect operation is used to combine two **SELECT** statements, but it only returns the records which are common from both **SELECT** statements.
- In case of **Intersect** the number of columns and datatype must be same.
- Example:

Select \* from First

Intersect

Select \* from second;

- The following are the result of above query

ID	NAME
2	adam

#### 4. Minus

- The Minus operation combines results of two **SELECT** statements and return only those in the final result, which belongs to the first set of the result.

- Example:

Select \* from First

Minus

Select \* from second;

- The following are the result of above query

ID	NAME
1	abhi

### 4.11 Full-Text Search

- Full-Text Search in MySQL server lets users run full-text queries against character-based data in MySQL tables.

- You must create a full-text index on the table before you run full-text queries on a table.
- The full-text index can include one or more character-based columns in the table.
- FULLTEXT is the index type of full-text index in MySQL.
- InnoDB or MyISAM tables use Full-text indexes.
- Full-text indexes can be created only for VARCHAR, CHAR or TEXT columns.
- A FULLTEXT index definition can be given in the CREATE TABLE statement or can be added later using ALTER TABLE or CREATE INDEX.
- Large data sets without FULLTEXT index is much faster to load data into a table than to load data into a table which has an existing FULLTEXT index.
- Therefore create the index after loading data.

➤ **Syntax:**

```
MATCH (col1,col2,col3...) AGAINST (expr [search_modifier])
```

- col1, col2, col3 - Comma-separated list that names the columns to be searched
  - AGAINST() takes a string to search, and an optional modifier that indicates what type of search to perform.
  - The search string must be a string value. The value is constant during query evaluation.
- There are three types of full-text searches :

- Natural Language Full-Text Searches
- Boolean Full-Text searches
- Query expansion searches
- The minimum length of the word for full-text searches as follows :
  - Three characters for InnoDB search indexes.
  - Four characters for MyISAM search indexes.
- Stop words are words that are very common such as 'on', 'the' or 'it', appear in almost every document. These type of words are ignored during searching.
- **Example:**

```
SELECT * FROM table_name WHERE MATCH(col1, col2)
```

```
AGAINST('search terms' IN NATURAL LANGUAGE MODE)
```

## Unit – 5

### PHP with MySQL

#### **5.1 Working MySQL with PHP:**

- PHP will work with virtually all database software, including Oracle and Sybase but most commonly used is freely available MySQL database.
- You have gone through MySQL tutorial to understand MySQL Basics.
- Downloaded and installed a latest version of MySQL.
- Created database user **guest** with password **guest123**.
- If you have not created a database then you would need root user and its password to create a database.

We have divided this chapter in the following sections –

- Connecting to MySQL database – how to use PHP to open and close a MySQL database connection.
- Create MySQL Database Using PHP – This explains how to create MySQL database and tables using PHP.
- Delete MySQL Database Using PHP – This part explains how to delete MySQL database and tables using PHP.
- Insert Data To MySQL Database – Once you have created your database and tables then you would like to insert your data into created tables. This session will take you through real example on data insert.
- Retrieve Data From MySQL Database – how to fetch records from MySQL database using PHP.

- Using Paging through PHP – This one explains how to show your query result into multiple pages and how to create the navigation link.
- Updating Data Into MySQL Database – This explains how to update existing records into MySQL database using PHP.
- Deleting Data From MySQL Database – This explains how to delete or purge existing records from MySQL database using PHP.

## **5.2 DATABASE CONNECTIVITY**

### 1. Opening Database Connection

- PHP provides **mysql\_connect** function to open a database connection. This function takes five parameters and returns a MySQL link identifier on success, or FALSE on failure.

- **Syntax**

```
connection mysql_connect(server,user,passwd,new_link,client_flag);
```



Sr.No	Parameter & Description
1	<p><b>server</b></p> <p>Optional – The host name running database server. If not specified then default value is <b>localhost:3306</b>.</p>
2	<p><b>user</b></p> <p>Optional – The username accessing the database. If not specified then default is the name of the user that owns the server process.</p>
3	<p><b>passwd</b></p> <p>Optional – The password of the user accessing the database. If not specified then default is an empty password.</p>
4	<p><b>new_link</b></p> <p>Optional – If a second call is made to <code>mysql_connect()</code> with the same arguments, no new connection will be established; instead, the identifier of the already opened connection will be returned.</p>
5	<p><b>client_flags</b></p> <p>Optional – A combination of the following constants –</p> <ul style="list-style-type: none"> <li>▣ <b>MYSQL_CLIENT_SSL</b> – Use SSL encryption</li> <li>▣ <b>MYSQL_CLIENT_COMPRESS</b> – Use compression protocol</li> <li>▣ <b>MYSQL_CLIENT_IGNORE_SPACE</b> – Allow space after function names</li> <li>▣ <b>MYSQL_CLIENT_INTERACTIVE</b> – Allow interactive timeout seconds of inactivity before closing the connection</li> </ul>

## Closing Database Connection

- Its simplest function **mysql\_close** PHP provides to close a database connection.

- This function takes connection resource returned by `mysql_connect` function.
- It returns `TRUE` on success or `FALSE` on failure.
- Syntax

```
bool mysql_close ( resource $link_identifier );
```

- If a resource is not specified then last opened database is closed.

- Example:

```
<?php

$dbhost = 'localhost:3036';
$dbuser = 'guest';
$dbpass = 'guest123';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);

if(! $conn ) {
    die('Could not connect: ' . mysql_error());
}

echo 'Connected successfully';
mysql_close($conn);
?>
```

### 5.3 Usage Of Mysqlcommands In PHP

- MySQL works very well in combination of various programming languages like PERL, C, C++, JAVA and PHP.
- Out of these languages, PHP is the most popular one because of its web application development capabilities.
- PHP provides various functions to access the MySQL database and to manipulate the data records inside the MySQL database.
- You would require to call the PHP functions in the same way you call any other PHP function.
- The PHP functions for use with MySQL have the following general format –

`mysql_function(value,value,...);`

- The second part of the function name is specific to the function, usually a word that describes what the function does. The following are the functions, which we will use in our tutorial –

`mysqli_connect($connect);`

`mysqli_query($connect,"SQL statement");`

The following are the functions of MYSQL are:

<b>Function</b>	<b>Description</b>
<code>mysqli_affected_rows()</code>	Returns the number of affected rows in the previous MySQL operation
<code>mysqli_close()</code>	Closes a previously opened database connection
<code>mysqli_connect()</code>	Opens a new connection to the MySQL server
<code>mysqli_errno()</code>	Returns the last error code for the most recent function call
<code>mysqli_error()</code>	Returns the last error description for the most recent

	function call
<u>mysqli_fetch_all()</u>	Fetches all result rows as an associative array, a numeric array, or both
<u>mysqli_fetch_array()</u>	Fetches a result row as an associative, a numeric array, or both
<u>mysqli_fetch_assoc()</u>	Fetches a result row as an associative array
<u>mysqli_fetch_row()</u>	Fetches one row from a result-set and returns it as an enumerated array
<u>mysqli_free_result()</u>	Frees the memory associated with a result
<u>mysqli_num_rows()</u>	Returns the number of rows in a result set
<u>mysqli_query()</u>	Performs a query against the database
<u>mysqli_real_escape_string()</u>	Escapes special characters in a string for use in an SQL statement
<u>mysqli_select_db()</u>	Changes the default database for the connection

- MySQL command-line client commands
- This command allows us to connect with MySQL Server with a username and passwords using below syntax.

**mysql -u [username] -p;**

If you want to connect with a particular database, use this syntax:

**mysql -u [username] -p [database];**

If you want to set a new password, use this syntax:

**mysqladmin -u root password your\_password;**

We can clear the console window in Linux using the below command:

**mysql> system clear;**

## **5.4 Processing Result Sets**

- The return value of a successful `mysql_query()` invocation can be processed in a number of different ways, depending on the type of query executed.

### **Queries Which Return Data**

- For SELECT-type queries, a number of techniques exist to process the returned data.
- The simplest is the `mysql_fetch_row()` function, which returns each record as a numerically indexed PHP array.
- Individual fields within the record can then be accessed using standard PHP-array notation. The following example illustrates this:

```
<?php
// open connection to MySQL server
$connection = mysql_connect('localhost', 'guest', 'pass')
or die ('Unable to connect!');
// select database for use
mysql_select_db('db2') or die ('Unable to select database!');

// create and execute query
$query = 'SELECT itemName, itemPrice FROM items';
$result = mysql_query($query)
or die ('Error in query: $query. ' . mysql_error());
// check if records were returned
if (mysql_num_rows($result) > 0)
{
// iterate over record set
```

```
// print each field
while($row = mysql_fetch_row($result))
{
echo $row[0] . " - " . $row[1] . "\n";
}
}
else
{
// print error message
echo 'No rows found!';
}
// once processing is complete
// free result set
mysql_free_result($result);
// close connection to MySQL server
mysql_close($connection);
?>
```

➤ Notice, in the previous listing, how the call to `mysql_fetch_row()` is wrapped in a `mysql_num_rows()` conditional test.

➤ The `mysql_num_rows()` function returns the number of records in the result set and comes in handy to check whether the query returned any records at all.

## **5.5 Handling Errors**

➤ Before you go out there and start building data-driven web sites, you should be aware that PHP's MySQL API also comes with some powerful error-tracking functions that can reduce debugging time.

- Take a look at the following example, which contains a deliberate error in the SELECT query string:

```
<?php
// open connection to MySQL server
$connection = mysql_connect('localhost', 'guest', 'pass') ⚡
or die ('Unable to connect!');
// select database for use
mysql_select_db('db2') or die ('Unable to select database!');
// create and execute query
$query = 'SELECT FROM items';
$result = mysql_query($query);
// if no result
// print MySQL error message
if(!$result)
{
echo 'MySQL error ' . mysql_errno() . ': ' . mysql_error();
mysql_close($connection);
}
?>
```

- The `mysql_errno()` function displays the error code returned by MySQL if there's an error in your SQL statement, while the `mysql_error()` function returns the actual error message.

### **Using Ancillary Functions**

- PHP's MySQL API comes with a number of ancillary functions that may be used to find out more about the databases and tables on the MySQL server or to obtain server status information.

- The below Table lists the important functions in this category.

Function	What It Does
<code>mysql_get_server_info()</code>	Returns the version number of the MySQL server
<code>mysql_get_proto_info()</code>	Returns the version number of the MySQL protocol
<code>mysql_get_client_info()</code>	Returns the version number of the MySQL client
<code>mysql_get_host_info()</code>	Returns information on the MySQL host
<code>mysql_thread_id()</code>	Returns the thread ID for the current MySQL connection
<code>mysql_list_dbs()</code>	Returns a list of databases available on the MySQL server
<code>mysql_list_tables()</code>	Returns a list of tables available in a specified MySQL database
<code>mysql_list_fields()</code>	Returns information about the fields of a specified MySQL table
<code>mysql_stat()</code>	Returns status information about the MySQL server
<code>mysql_info()</code>	Returns information about the last executed query
<code>mysql_db_name()</code>	Returns a name of a database from the list generated by <code>mysql_list_dbs()</code>
<code>mysql_tablename()</code>	Returns a name of a table from the list generated by <code>mysql_list_tables()</code>
<code>mysql_ping()</code>	Tests the server connection

Table: Useful Debugging and Diagnostic Functions

## **5.6 Validating User Input Through Database Layer And Application Layer**

### **Setting Input Constraints at the Database Layer:**

- When it comes to maintaining the integrity of your database, a powerful tool is provided by the database system itself: the capability to restrict the type of data entered into a field or make certain fields mandatory, using field definitions or constraints.

### **Using the NULL Modifier**

- MySQL enables you to specify whether a field is allowed to be empty or if it must necessarily be filled with data, by placing the NULL and NOT NULL modifiers after each field definition.



- This is a good way to ensure that required fields of a record are never left empty, because MySQL will simply reject entries that do not have all the necessary fields filled in.
- Here's an example of this in action:

```
mysql> CREATE TABLE products (  
-> id int(4),  
-> name varchar(50)  
-> );
```

Query OK, 0 rows affected (0.06 sec)

- Here, the name field can hold NULL values, which means the following INSERT will go unchallenged,

```
mysql> INSERT INTO products VALUES (NULL, NULL);
```

Query OK, 1 row affected (0.06 sec)

and create the following nonsense entry in the table:

- ```
mysql> SELECT * FROM products;
```

```
+-----+-----+  
| id    | name  |  
+-----+-----+  
| NULL  | NULL  |  
+-----+-----+
```

- Now, look what happens if you make the name field mandatory:

```
mysql> CREATE TABLE products (  
-> id int(4),  
-> name varchar(50) NOT NULL  
-> );
```

Query OK, 0 rows affected (0.05 sec)

```
mysql> INSERT INTO products VALUES (NULL, NULL);
```

ERROR 1048: Column 'name' cannot be null

- Of course, because MySQL makes a distinction between a NULL value and an empty string ("), the following record—which is also meaningless—would be accepted.

```
mysql> INSERT INTO products VALUES ('', '');  
Query OK, 1 row affected (0.05 sec)
```

```
mysql> SELECT * FROM products;
```

```
+-----+-----+  
| id   | name |  
+-----+-----+  
|    0 |     |  
+-----+-----+
```

- Thus, while the NOT NULL modifier can help reduce the incidence of empty or incomplete records in a database, it is not a comprehensive solution.
- It needs to be supplemented by application-level verification to ensure that empty strings are caught before they get to the database.

### Validating Input at the Application Layer

- When it comes to catching errors in user input, the best place to do this is at the point of entry—the application itself.
- you can use to catch common input errors and ensure that they don't get into your database.

### Checking for Required Values

- One of the most common mistakes a novice programmer makes is forgetting to check for required field values.

This can result in a database with numerous empty records, and these empty records can, in turn, affect the accuracy of your queries.

- To see what I mean by this, consider the following users table:

```
mysql> CREATE TABLE users (
```

-> **username varchar(8) NOT NULL DEFAULT ''**,

-> **password varchar(8) NOT NULL DEFAULT ''**

-> ) **TYPE=MyISAM;**

➤ Query OK, 0 rows affected (0.05 sec)

➤ EXAMPLE

//connect to database

// open connection

```
$connection = mysql_connect('localhost', 'guest', 'pass') ⚡
```

```
or die ('Unable to connect!');
```

// select database

```
mysql_select_db('db2') or die ('Unable to select database!');
```

// create query

```
$query = "INSERT INTO users (username, password) ⚡
```

```
VALUES ('$username', '$password)";
```

// execute query

```
$result = mysql_query($query) ⚡
```

```
or die ("Error in query: $query. " . mysql_error());
```

// close connection

➤ `mysql_close($connection);`

## **5.7 Formatting Character Data**

- A lot of your MySQL data is going to be stored as strings or text blocks, in CHAR, VARCHAR, or TEXT fields.
- It's essential that you know how to manipulate this string data and adjust it to fit the requirements of your application user interface.
- Both PHP and MySQL come equipped with numerous string manipulation functions (in fact, they overlap in functionality in many places).

### **Concatenating String Values**

- It's pretty simple—just string together the variables you want to concatenate using the PHP concatenation operation, a period (.).
- Concatenating fields from a MySQL result set is equally simple—just assign the field values to PHP variables and concatenate the variables together in the normal manner.

### **Padding String Values**

- you read about the PHP trim() function, used to strip leading and trailing white space from string values prior to testing them for validity or inserting them into a database.
- However, PHP also comes with the str\_pad() function, which does just the reverse: it pads strings to a specified length using either white space or a user-specified character sequence.
- This can come in handy if you need to artificially elongate string values for display or layout purposes.

### **Altering String Case**

- If you need case manipulation, just reach for PHP's string manipulation API again.
- Four useful functions are here:
  - strtolower(), which converts all characters in a string to lowercase;
  - strtoupper(), which converts all characters to uppercase;
  - ucfirst(), which converts the first character of a string to uppercase, and
  - ucwords(), which converts the first character of all the words in a string to uppercase.

### **5.8 Dealing with Special Characters**

- When it comes to displaying large text blocks on a web page, a PHP developer must grapple with a number of issues.
- Special characters need to be protected, white space and line breaks must be preserved, and potentially malicious HTML code must be defanged.

- PHP comes with a number of functions designed to perform just these tasks.
- The revised listing uses three new functions.
  - The `htmlspecialchars()` function takes care of replacing special characters like `"`, `&`, `<`, and `>` with their corresponding HTML entity values. This function is useful to defang user-supplied HTML text and render it incapable of effecting the display or functionality of your web page.
  - Next, the `wordwrap()` function wraps text to the next line once it reaches a particular, user-defined size, by inserting the `\n` newline character at appropriate points in the text block (these are then converted into HTML line breaks by the next function).
  - Finally, the `nl2br()` function automatically preserves newlines in a text block, by converting them to HTML `<br />` elements. This makes it possible to reproduce the original formatting of the text when it is displayed.

## **5.9 Formatting Numeric Data**

- Just as you can massage string values into a number of different shapes, so, too, can you format numeric data.
- Both PHP and MySQL come with a full set of functions to manipulate integer and floating-point numbers, and to format large numeric values for greater readability.

### **Using Decimal and Comma Separators**

- When it comes to formatting numeric values in PHP, there are only two functions:

1. number\_format()

2. sprintf()

- It can be used to control both the visibility and the appearance of the decimal digits, as well as the character used as the thousands separator.
- To see how this works, consider the following table:

```
mysql> SELECT accountNumber, accountName, accountBalance FROM accounts;
```

| accountNumber | accountName | accountBalance |
|---------------|-------------|----------------|
| 1265489921    | James D     | 2346.00000     |
| 2147483647    | Timothy J   | 56347.50000    |
| 5739304575    | Harish K    | 996564.87500   |
| 2173467271    | Kingston X  | 634238.00000   |
| 2312934021    | Sue U       | 34.67000       |
| 1248954638    | Ila T       | 5373.81982     |
| 2384371001    | Anil V      | 72460.00000    |
| 9430125467    | Katrina P   | 100.00000      |
| 1890192554    | Pooja B     | 17337.11914    |
| 2388282010    | Sue U       | 388883.12500   |
| 2374845291    | Jacob N     | 18410.00000    |

➤ Here's a PHP script that displays this information on a web page, using number\_format() to display account balances with two decimal places and commas as thousand separators.

The PHP sprintf() function is similar to the sprintf() function that

C programmers are used to. To format the output, you need to use *field*

*templates*, templates that represent the format you'd like to display. Common field

templates are listed in Table

| Template        | What It Represents |
|-----------------|--------------------|
| <code>%s</code> | string             |
| <code>%d</code> | decimal number     |
| <code>%x</code> | hexadecimal number |
| <code>%o</code> | octal number       |
| <code>%f</code> | float number       |

```
<?php
// returns 00003
echo sprintf("%05d", 3);
// returns $25.99
echo sprintf("$%2.2f", 25.99);
// returns ****56
printf("%*6d", 56);
?>
```

## 5.10 Formatting Dates and Times

- you can use PHP's `mktime()` function to obtain a UNIX timestamp for any arbitrary date/time value.
- PHP offers the `date()` function, which accepts two arguments: one or more *format specifiers*, which indicates how the timestamp should be formatted, and the timestamp itself (optional; PHP assumes the current time if this second argument is not provided).
- To see a few examples of the `date()` function in action, create and run the following script:

```
<?php
// retrieve current date and time
// prints a date and time like "09:18 pm 19 Jun 2004"
echo date("h:i a d M Y", mktime());
```

```
// returns just the date "27 April 2003"
echo date("d F Y", mktime(0, 0, 0, 04, 27, 2003));
// returns the time in 24-hr format "21:18"
echo date("H:i", mktime());
?>
```

- The following table display Common Format Specifiers Supported by the date() Function

| Specifier | What It Means                                |
|-----------|----------------------------------------------|
| d         | Day of the month; numeric                    |
| D         | Day of the week; short string                |
| F         | Month of the year; long string               |
| h         | Hour; numeric 12-hour format                 |
| H         | Hour; numeric 24-hour format                 |
| i         | Minute; numeric                              |
| l         | Day of the week; long string                 |
| L         | Boolean indicating whether it is a leap year |
| m         | Month of the year; numeric                   |
| M         | Month of the year; short string              |
| s         | Seconds; numeric                             |
| T         | Timezone                                     |
| Y         | Year; numeric                                |
| z         | Day of the year; numeric                     |

### Example:

```
SELECT DATE_FORMAT(NOW(), '%W, %D %M %Y %r');
```

### Output:

```
+-----+
| DATE_FORMAT(NOW(), '%W, %D %M %Y %r') |
```



+-----+  
| Thursday, 18th November 2004 12:07:55 PM |

- The following tables have MySQL Date/Time Formatting Codes

| Symbol | What It Means                               |
|--------|---------------------------------------------|
| %a     | Short weekday name (Sun, Mon . . .)         |
| %b     | Short month name (Jan, Feb . . .)           |
| %d     | Day of the month                            |
| %H     | Hour (01, 02 . . .)                         |
| %I     | Minute (00, 01 . . .)                       |
| %j     | Day of the year (001, 002 . . .)            |
| %m     | 2-digit month (00, 01 . . .)                |
| %M     | Long month name (January, February . . . .) |
| %p     | AM/PM                                       |
| %r     | Time in 12-hour format                      |
| %S     | Second (00, 01 . . .)                       |
| %T     | Time in 24-hour format                      |
| %w     | Day of the week (0,1 . . .)                 |
| %W     | Long weekday name (Sunday, Monday . . .)    |
| %Y     | 4-digit year                                |

Example:

```
mysql> SELECT TIME_FORMAT(19690609140256, '%h:%i %p');
```

**Output:**

+-----+  
| TIME\_FORMAT(19690609140256, '%h:%i %p') |  
+-----+  
| 02:02 PM |

+-----+

➤ The following table display the More MySQL Date Functions

| Function       | What It Does                                                            |
|----------------|-------------------------------------------------------------------------|
| DAYOFWEEK ( )  | Returns a number (1 to 7) representing the day of the week for a date   |
| DAYOFMONTH ( ) | Returns the day component (1 to 31) of a date                           |
| DAYOFYEAR ( )  | Returns a number (1 to 366) representing the day of the year for a date |
| DAYNAME ( )    | Returns the weekday name for a date                                     |
| HOUR ( )       | Returns the hour component (0–23) of a time                             |
| MINUTE ( )     | Returns the minute component (0–59) of a time                           |
| MONTH ( )      | Returns the month component (1 to 12) for a date                        |
| MONTHNAME ( )  | Returns the month name for a date                                       |
| QUARTER ( )    | Returns the quarter (1–2) in which a date falls                         |
| WEEK ( )       | Returns the week number (0–53) for a date                               |
| YEAR ( )       | Returns the year component (1000–9999) of a date                        |

### Example:

```
SELECT DAYOFMONTH(NOW()), DAYOFYEAR('1979-01-02');
```

+-----+

```
| DAYOFMONTH(NOW()) | DAYOFYEAR('1979-01-02') |
```

+-----+

```
          | 23 |                2 |
```