5. Programming in PHP & MySQL

5.1. Origins and Uses of PHP

- Developed by Rasmus Lerdorf in 1994
- Developed to allow him to track visitors to his Web site
- PHP is an open-source product
- PHP is an acronym for Personal Home Page, or PHP: Hypertext Preprocessor
- PHP is used for form handling, file processing, and database access

5.2. Overview of PHP

PHP is a server-side scripting language whose scripts are embedded in HTML documents. It is similar to JavaScript, but on the server side. It is an alternative to CGI, Active Server Pages(ASP), and Java Server Pages (JSP).

1. The PHP processor has two modes:
    - copy (HTML) and
    - interpret (PHP)
2. PHP syntax is similar to that of JavaScript
3. PHP is dynamically typed

5.3. General Syntactic Characteristics

PHP code can be specified in an HTML document internally or externally:

Internally:

```
<?php
...
?>
```

Externally:

```
include ("myScript.inc")
```

The file can have both PHP and HTML. If the file has PHP, the PHP must be in <?php .. ?>, even if the include is already in <?php .. ?>. All variable names begin with dollar ($).

Comments - three different kinds (Java and Perl)

// ... --> single line comment

... --> single line comment

/* ... */ --> multiple line comment

Compound statements are formed with braces. Compound statements cannot be blocks.

5.4. Primitives, Operations, and Expressions

5.4.1. Variables

- There are no type declarations .
- An unassigned (unbound) variable has the value, NULL
- The ***unset*** function sets a variable to NULL
- The **isset** function is used to determine whether a variable is NULL
- PHP has many predefined variables, including the environment variables of the host operating system
- You can get a list of the predefined variables by calling phpinfo() in a script

There are eight primitive types:

1. Four scalar types: Boolean, integer, double, and string
2. Two compound types: array and object
3. Two special types: resource and NULL

Points to remember:

- Integer & double are typical , like in C
- Characters are single bytes
- String literals use single or double quotes

Difference between single quoted string literals and double qouted string literals

1. Single-quoted string literals
    - Embedded variables are NOT interpolated
    - Embedded escape sequences are NOT recognized

2. Double-quoted string literals
    - Embedded variables are interpolated
    - If there is a variable name in a double quoted string but you don't want it interpolated, it must be back-slashed
    - Embedded escape sequences are recognized

Note: For both single- and double-quoted literal strings, embedded delimiters must be back-slashed .

Boolean

- values are true and false (case insensitive)
- 0, " " and "0" are false; others are true

5.4.2. Arithmetic Operators and Expressions

Usual operators

- If the result of integer division is not an integer, a double is returned
- Any integer operation that results in overflow produces a double
- The modulus operator coerces its operands to integer, if necessary
- When a double is rounded to an integer, the rounding is always towards zero

Arithmetic functions

- floor, ceil, round, abs, min, max, rand, etc.

String Operations and Functions

- The only operator is period, for concatenation
- Indexing - $str{3} is the fourth character

Functions:

- strlen, strcmp, strpos, substr, strtolower, strtoupper as in C
- chop – remove whitespace from the right end
- trim – remove whitespace from both ends
- ltrim – remove whitespace from the left end

Scalar Type Conversions

String to numeric

- If the string contains an e or an E, it is converted to double; otherwise to int
- If the string does not begin with a sign or a digit, zero is used

Explicit conversions – casts

- e.g., (int)$total or intval($total) or settype($total, "integer")

The type of a variable can be determined with ***gettype*** or ***is_type***

- gettype($total) - it may return "unknown"
- is_integer($total) – a predicate function

5.5. Output

Output from a PHP script is HTML that is sent to the browser . HTML is sent to the browser through standard output

There are three ways to produce output:

- echo
- print
- printf

echo and print take a string, but will coerce other values to strings

An Example:

```
<html>
<head><title> Trivial php example </title>
</head>
<body>
<?php
print "Welcome to my Web site!";
?>
</body>
</html>
```

5.6. Control Statements

Control Expressions

- Relational operators - same as JavaScript, (including === and !==)
- Boolean operators - same as Perl (two sets, && , || etc.)

Selection statements

- if, if-else, elseif --> same as C
- switch - as in C
- The switch expression type must be integer, double, or string
- while - just like C
- do-while - just like C
- for - just like C
- foreach - discussed later
- break - in any for, foreach, while, do-while, or switch
- continue - in any loop

5.6.1. IF...ELSE IF...ELSE

The if statement execute a single statement or a group of statements if a certain condition is met. It can not do anything if the condition is false. For this purpose elseif and else is used. elseif is a combination of if and else. It extends an if statement to execute a single statement or a group of statements if a certain condition is met. It can not do anything if the condition is false.

Syntax

```
if (condition)
   execute statement(s) if condition is true;
elseif (another condition)
  execute statement(s) if condition is true;
else
  execute statement(s) if all condition(s) are false;
```

Example :

```
<?php
if ($x > $y)
{
echo "x is bigger than y";
}
elseif ($x == $y)
{
echo "x is equal to y";
}
else
{
echo "x is smaller than y";
}
?>
```

5.6.2. The While Loop

The While statement executes a block of code if and as long as a specified condition evaluates to true. If the condition becomes false, the statements within the loop stop executing and control passes to the statement following the loop. The While loop syntax is as follows:

```
while (condition)
{
 code to be executed;
}
```

The block of code associated with the While statement is always enclosed within the { opening and } closing brace symbols to tell PHP clearly which lines of code should be looped through.

While loops are most often used to increment a list where there is no known limit to the number of iterations of the loop.

For example:

```
while (there are still rows to read from a database)
{
  read in a row;
  move to the next row;
}
```

Let's have a look at the examples. The first example defines a loop that starts with i=0. The loop will continue to run as long as the variable i is less than, or equal to 10. i will increase by 1 each time the loop runs:

```
<?php
$i=0;
while ($i <= 10) { // Output values from 0 to 10
  echo "The number is ".$i."<br />";
  $i++;
}
?>
```

5.6.3. The Do...While Loop

The Do...While statements are similar to While statements, except that the condition is tested at the end of each iteration, rather than at the beginning. This means that the Do...While loop is guaranteed to run at least once. The Do...While loop syntax is as follows:

```
do
{
  code to be exected;
}
while (condition);
```

The example below will increment the value of i at least once, and it will continue incrementing the variable i as long as it has a value of less than or equal to 10:

```
<?php
$i = 0;
do {
  echo "The number is ".$i."<br/>";
  $i++;
}
while ($i <= 10);
?>
```

5.6.4. The For Loop

The For statement loop is used when you know how many times you want to execute a statement or a list of statements. For this reason, the For loop is known as a definite loop. The syntax of For loops is a bit more complex, though for loops are often more convenient than While loops. The For loop syntax is as follows:

```
for (initialization; condition; increment)
{
  code to be executed;
}
```

The For statement takes three expressions inside its parentheses, separated by semi-colons. When the For loop executes, the following occurs:

1. The initializing expression is executed. This expression usually initializes one or more loop counters, but the syntax allows an expression of any degree of complexity.
2. The condition expression is evaluated. If the value of condition is true, the loop statements execute. If the value of condition is false, the For loop terminates.
3. The update expression increment executes.
4. The statements execute, and control returns to step 2.

Have a look at the very simple example that prints out numbers from 0 to 10:

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

Next example generates a multiplication table 2 through 9. Outer loop is responsible for generating a list of dividends, and inner loop will be responsible for generating lists of dividers for each individual number:

```
<?php
echo "<h1>Multiplication table</h1>";
echo "<table border=2 width=50%";

for ($i = 1; $i <= 9; $i++ ) {   //this is the outer loop
 echo "<tr>";
 echo "<td>".$i."</td>";

  for ( $j = 2; $j <= 9; $j++ ) { // inner loop
      echo "<td>".$i * $j."</td>";
   }

  echo "</tr>";
}

echo "</table>";
?>
```

5.6.5. The Foreach Loop

The Foreach loop is a variation of the For loop and allows you to iterate over elements in an array. There are two different versions of the Foreach loop. The Foreach loop syntaxes are as follows:

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

```
foreach (array as key => value)
{
  code to be executed;
}
```

The example below demonstrates the Foreach loop that will print the values of the given array:

```
<?php
$email = array('john.smith@example.com', 'alex@example.com');
foreach ($email as $value) {
  echo "Processing ".$value."<br />";
}
?>
```

PHP executes the body of the loop once for each element of $email in turn, with $value set to the current element. Elements are processed by their internal order. Looping continues until the Foreach loop reaches the last element or upper bound of the given array.

An alternative form of Foreach loop gives you access to the current key:

```
<?php
$person = array('name' => 'Andrew', 'age' => 21, 'address' => '77, Lincoln st.');
foreach ($person as $key => $value) {
  echo $key." is ".$value."<br />";
```

```
}
?>
```

In this case, the key for each element is placed in $key and the corresponding value is placed in $value.

The Foreach construct does not operate on the array itself, but rather on a copy of it. During each loop, the value of the variable $value can be manipulated but the original value of the array remains the same.

5.6.6. Break and Continue Statements

Sometimes you may want to let the loops start without any condition, and allow the statements inside the brackets to decide when to exit the loop. There are two special statements that can be used inside loops: Break and Continue.

The Break statement terminates the current While or For loop and continues executing the code that follows after the loop (if any). Optionally, you can put a number after the Break keyword indicating how many levels of loop structures to break out of. In this way, a statement buried deep in nested loops can break out of the outermost loop.

Examples below show how to use the Break statement:

```
<?php
echo "<p><b>Example of using the Break statement:</b></p>";
for ($i=0; $i<=10; $i++) {
  if ($i==3){break;}
  echo "The number is ".$i;
  echo "<br />";
}
echo "<p><b>One more example of using the Break statement:</b><p>";
$i = 0;
$j = 0;
while ($i < 10) {
 while ($j < 10) {
  if ($j == 5) {break 2;} // breaks out of two while loops
```

```
   $j++;
 }
 $i++;
}
echo "The first number is ".$i."<br />";
echo "The second number is ".$j."<br />";
?>
```

The Continue statement terminates execution of the block of statements in a While or For loop and continues execution of the loop with the next iteration:

```
<?php
echo "<p><b>Example of using the Continue statement:</b><p>";
for ($i=0; $i<=10; $i++) {
  if (i==3){continue;}
  echo "The number is ".$i;
  echo "<br />";
}
?>
```

5.7. Arrays

PHP arrays are not like the arrays of any other programming language . A PHP array is a generalization of the arrays of other languages . A PHP array is really a mapping of keys to values, where the keys can be numbers (to get a traditional array) or strings (to get a hash).

5.7.1. Creating Arrays

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

e.g.,

```
$list = array(0 => "apples",1 => "oranges", 2 => "grapes")
```

This is a “regular” array of strings.

- If a key is omitted and there have been integer keys, the default key will be the largest current key + 1
- If a key is omitted and there have been no integer keys, 0 is the default key
- If a key appears that has already appeared, the new value will overwrite the old one

Arrays can have mixed kinds of elements

```
$list = array("make" => "Cessna", "model" => "C210", "year" => 1960, 3 => "sold");
$list = array(1, 3, 5, 7, 9);
$list = array(5, 3 => 7, 5 => 10, "month" => "May");
$colors = array('red' , 'blue' , 'green' , 'yellow');
```

5.7.2. Accessing array elements – use brackets

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

- If an element with the specified key does not exist, it is created
- If the array does not exist, the array is created
- The keys or values can be extracted from an array

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

5.7.3. Deleting an array or its elements

An array can be deleted with unset

```
unset($list);
unset($list[4]); # No index 4 element now
```

Array Functions

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

Sequential access to array elements

- current and next

```
$colors = array("Blue", "red", "green", "yellow");
$color = current($colors);
print("$color <br />");
while ($color = next($colors))
print ("$color <br />");
```

This does not always work – for example, if the value in the array happens to be FALSE

- ***array_push($list, $element) and array_pop($list)*** Used to implement stacks in arrays

```
foreach (array_name as scalar_name) { ... }
```

```
foreach ($colors as $color) {
print "Is $color your favorite?<br /> ";
}
```

Output:

Is red your favorite color?

Is blue your favorite color?

Is green your favorite color?

Is yellow your favorite color?

foreach can iterate through both keys and values:

```
foreach ($colors as $key => $color) { … }
```

Inside the compound statement, both $key and $color are defined

```
$ages = array("Bob" => 42, "Mary" => 43);
foreach ($ages as $name => $age)
print("$name is $age years old <br />");
```

5.7.4. Array Sorting

To sort the values of an array, leaving the keys in their present order - intended for traditional arrays

e.g., sort($list);

- The sort function does not return anything
- Works for both strings and numbers, even mixed strings and numbers

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

In PHP 4, the sort function can take a second parameter, which specifies a particular kind of sort

```
sort($list, SORT_NUMERIC);
```

5.7.4.1. asort

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

5.7.4.2. rsort

To sort the values of an array into reverse order

5.7.4.3. ksort

To sort the elements of an array by the keys, maintaining the key/value relationships

```
$list("Fred" => 17, "Mary" => 21, "Bob" => 49, "Jill" => 28);
ksort($list);
// $list is now ("Bob" => 49, // "Fred" => 17, "Jill" => 28, "Mary" => 21)
```

5.7.4.4. krsort

To sort the elements of an array by the keys into reverse order.

5.8. MySQL

MySQL is an open-source Database Management System, widely used in websites around the world. The queries are written in Structured Query Language. PHP combined with MySQL are cross-platform (you can develop in Windows and serve on a Unix platform).

Datatypes in MySQL

Type	Size	Description
CHAR[Length]	Length bytes	A fixed-length field from 0 to 255 characters long.
VARCHAR(Length)	String length + 1 bytes	A fixed-length field from 0 to 255 characters long.
TINYTEXT	String length + 1 bytes	A string with a maximum length of 255 characters.
TEXT	String length + 2 bytes	A string with a maximum length of 65,535 characters.
MEDIUMTEXT	String length + 3 bytes	A string with a maximum length of 16,777,215 characters.
LONGTEXT	String length + 4 bytes	A string with a maximum length of 4,294,967,295 characters.
TINYINT[Length]	1 byte	Range of -128 to 127 or 0 to 255 unsigned.
SMALLINT[Length]	2 bytes	Range of -32,768 to 32,767 or 0 to 65535 unsigned.
MEDIUMINT[Length]	3 bytes	Range of -8,388,608 to 8,388,607 or 0 to 16,777,215 unsigned.
INT[Length]	4 bytes	Range of -2,147,483,648 to 2,147,483,647 or 0 to 4,294,967,295 unsigned.
BIGINT[Length]	8 bytes	Range of -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 or 0 to 18,446,744,073,709,551,615 unsigned.
FLOAT	4 bytes	A small number with a floating decimal point.
DOUBLE[Length, Decimals]	8 bytes	A large number with a floating decimal point.
DECIMAL[Length, Decimals]	Length + 1 or Length + 2 bytes	A DOUBLE stored as a string, allowing for a fixed decimal point.
DATE	3 bytes	In the format of YYYY-MM-DD.
DATETIME	8 bytes	In the format of YYYY-MM-DD HH:MM:SS.
TIMESTAMP	4 bytes	In the format of YYYYMMDDHHMMSS; acceptable range ends inthe year 2037.
TIME	3 bytes	In the format of HH:MM:SS
ENUM	1 or 2 bytes	Short for enumeration, which means that each column can haveone of several possible values.
SET	1, 2, 3, 4, or 8 bytes	Like ENUM except that each column can have more than one ofseveral possible values.

5.8.1. Structured Query Language

SQL is used to communicate with a database. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database. The standard SQL commands such as "Select", "Insert", "Update", "Delete", "Create", and "Drop" can be used to accomplish almost everything that one needs to do with a database.

5.8.1.1. Database Operations

List all databases

```
SHOW DATABASES;
```

Use a database

```
USE <db_name>;
```

Delete a database

```
DROP DATABASE <db_name>;
```

Creating a database

```
CREATE DATABASE <db_name>;
```

5.8.1.2. Table Operations

List all tables

```
SHOW TABLES;
```

Delete a table

```
DROP TABLE <table_name>;
```

Creating a table

```
CREATE TABLE table_name
(
        column_name1 data_type(size),
        column_name2 data_type(size),
        column_name3 data_type(size),
        ....
);
```

Insert data in table

```
INSERT INTO table_name
        VALUES (value1,value2,value3,...);
```

OR

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

Select data from table

```
SELECT column_name,column_name  FROM table_name;
```

OR

```
SELECT  *  FROM table_name;
```

Eg:

```
SELECT name, roll, age
FROM students;
```

Where clause for Selection

```
SELECT column_name,column_name
FROM table_name
WHERE column_name operator value;
```

Eg:

```
SELECT name, roll, age
FROM students
WHERE age > 20;
```

Like keyword in Where clause

```
SELECT column_name,column_name
FROM table_name
WHERE column_name LIKE pattern;
```

Eg:

```
SELECT name, roll, age
FROM students
WHERE name LIKE '%as%';
```

Update data in table

```
UPDATE table_name
SET column1=value1,column2=value2,...
WHERE some_column=some_value;
```

Eg:

```
UPDATE Customers
SET ContactName='Alfred Schmidt', City='Hamburg'
WHERE CustomerID=7;
```

NOTE: The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

Delete data from table

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

Eg:

```
DELETE FROM Customers
WHERE CustomerName='Alfreds Futterkiste' AND ContactName='Maria Anders';
```

NOTE: The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

To delete everything from table

```
DELETE FROM table_name;
OR
DELETE * FROM table_name;
```

5.8.2. PHP Connect to MySQL

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

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

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

5.8.2.1. Connect to Database

To connect to a database, we need a connection object to be made with valid credentials. Once connected to database, we can execute queries for various SQL operations.

Eg:

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

5.8.2.2. Creating a Table

To create a table, we need a connection object to be made with valid credentials. Once connected to database, we can execute the create table statement.

Eg:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";
try {
   $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
   // set the PDO error mode to exception
   $conn -> setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
   // sql to create table
   $sql = "CREATE TABLE MyGuests (
   id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
   firstname VARCHAR(30) NOT NULL,
   lastname VARCHAR(30) NOT NULL,
   email VARCHAR(50),
   reg_date TIMESTAMP
   )";
   // use exec() because no results are returned
   $conn -> exec($sql);
   echo "Table MyGuests created successfully";
   }
catch(PDOException $e)
   {
   echo $sql . "<br>" . $e -> getMessage();
   }
$conn = null;
?>
```

5.8.2.3. Inserting data into table

To insert the data, we need to connect to the database, then execute the insert statement.

Eg:

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

Inserting Multiple values with transaction statements

We can use transaction to insert multiple values at once. If anyone fails, we can rollback the whole transaction.

Eg:

```
 <?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";
try {
   $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
   // set the PDO error mode to exception
   $conn -> setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
   // begin the transaction
   $conn -> beginTransaction();
   // our SQL statements
   $conn -> exec("INSERT INTO MyGuests (firstname, lastname, email)
   VALUES ('John', 'Doe', 'john@example.com')");
   $conn -> exec("INSERT INTO MyGuests (firstname, lastname, email)
   VALUES ('Mary', 'Moe', 'mary@example.com')");
   $conn -> exec("INSERT INTO MyGuests (firstname, lastname, email)
   VALUES ('Julie', 'Dooley', 'julie@example.com')");
   // commit the transaction
   $conn -> commit();
   echo "New records created successfully";
   }
catch(PDOException $e)
   {
   // roll back the transaction if something failed
   $conn - > rollback();
   echo "Error: " . $e -> getMessage();
   }$conn = null;
?>
```

5.8.2.4. Deleting data from table

To delete the data, we need to connect to the database and then execute the delete query.

Eg:

```
 <?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";
try {
   $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
   // set the PDO error mode to exception
   $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
   // sql to delete a record
   $sql = "DELETE FROM MyGuests WHERE id=3";
   // use exec() because no results are returned
   $conn -> exec($sql);
   echo "Record deleted successfully";
   }
catch(PDOException $e)
   {
   echo $sql . "<br>" . $e -> getMessage();
   }
$conn = null;
?>
```

5.8.2.5. Selecting data from tables and showing it

Prepared Statements to get the data, avoid using concatenation.

Eg:

```
$stmt = $pdo->prepare('SELECT * FROM users WHERE email = ? AND status=?');
$stmt->execute([$email, $status]);
$user = $stmt->fetch();
```

OR

```
$stmt = $pdo->prepare('SELECT * FROM users WHERE email = :email AND
status=:status');
$stmt->execute(['email' => $email, 'status' => $status]);
$user = $stmt->fetch();
```

Fetch and iterate, use php foreach and echo

```
$stmt = $pdo->query('SELECT name FROM users');
foreach ($stmt as $row)
{
   echo $row['name'] . "\n";
}
```

Example:

PHP & MYSQL fetch data into associative mode, show in HTML table

```
 <?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";
try {
   $pdo = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
 $sql = "SELECT lastname,  firstname,  jobtitle FROM employees ORDER BY lastname";
   $q = $pdo -> query($sql);
   $q->setFetchMode(PDO::FETCH_ASSOC);
} catch (PDOException $e) {
```

```
  die("Could not connect to the database $dbname :" . $e ->  getMessage());
}
?>
<!DOCTYPE html>
<html>
  <head>
    <title>PHP MySQL Query Data Demo</title>
  </head>
  <body>
        <h1>Employees</h1>
      <table border=”1”>
        <thead>
          <tr>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Job Title</th>
          </tr>
        </thead>
        <tbody>
          <?php while ($row = $q->fetch()): ?>
            <tr>
              <td><?php echo $row['lastname']; ?></td>
              <td><?php echo $row['firstname']; ?></td>
              <td><?php echo $row['jobtitle']; ?></td>
            </tr>
          <?php endwhile; ?>
        </tbody>
      </table>
  </body>
</html>
```

5.9. PHP Form Handling

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

5.9.1. GET vs. POST

Both GET and POST create an array (e.g. array(key => value, key2 => value2, key3 => value3, ...)). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.

Both GET and POST are treated as $_GET and $_POST. These are superglobals, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

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

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

5.9.1.1. When to use GET?

Information sent from a form with the GET method is visible to everyone (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.

GET may be used for sending non-sensitive data.

Note: GET should NEVER be used for sending passwords or other sensitive information!

5.9.1.2. When to use POST?

Information sent from a form with the POST method is invisible to others (all names/values are embedded within the body of the HTTP request) and has no limits on the amount of information to send.

Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.

However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

Eg:

HTML FORM

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

PHP PROCESSING

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

5.10. PHP Cookies

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

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

Syntax:

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

Eg:

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

5.11. PHP Sessions

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

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

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

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

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

5.11.1. Start a PHP Session

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

5.11.2. Demonstrating Session across pages

To demonstrate Session across pages, we need to create session in one page and access it in another page.

Eg:

first.php

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
// Set session variables
```

```
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>
</body>
</html>
```

second.php

```
<?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>
```

5.11.3. Modifying Sessions

To change a session variable, just overwrite it.

Eg:

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
// to change a session variable, just overwrite it
$_SESSION["favcolor"] = "yellow";
print_r($_SESSION);
?>
</body>
</html>
```

5.11.4. Destroy a PHP Session

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

Eg:

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