Unit – III

❖ Objects :

Object-oriented programming (OOP) opens the door to cleaner designs, easier maintenance, and greater code reuse. The proven value of OOP is such that few today would dare to introduce a language that wasn't object-oriented. PHP supports many useful features of OOP, and this unit shows us how to use them.

OOP acknowledges the fundamental connection between data and the code that works on that data, and it lets we design and implement programs around that connection. For example, a bulletin-board system usually keeps track of many users. In a procedural programming language, each user would be a data structure, and there would pro4rbably be a set of functions that work with users data structures (create the new users, get their information, etc.). In an object-oriented programming language, each user would be an object - a data structure with attached code. The data and the code are still there, but they're treated as an inseparable unit.

In this hypothetical bulletin-board design, objects can represent not just users, but also messages and threads. A user object has a username and password for that user, and code to identify all the messages by that author. A message object knows which thread it belongs to and has code to post a new message, reply to an existing message, and display messages. A thread object is a collection of message objects, and it has code to display a thread index. This is only one way of dividing the necessary functionality into objects, though. For instance, in an alternate design, the code to post a new message life in the user object, not the message object. Designing object-oriented systems is a complex topic, and many books have been written on it. The good news is that however we design our system, we can implement it in PHP.

The object, as union of code and data, is the modular unit for application development and code reuse. This unit shows us how to define, create, and use objects in PHP. It covers basic OOP concepts as well as advanced topics such as introspection and serialization.

I] Terminology :

Every object-oriented language seems to have a different set of terms for the same old concepts. This section describes the terms that PHP uses, but be warned that in other languages these terms may have other meanings.

Let's return to the example of the users of a bulletin board. We need to keep track of the same information for each user, and the same functions can be called on each user's data structure. When we design the program, we decide the fields for each user and come up with the functions. In OOP terms, we are designing the user class. A class is a template for building objects.

An object is an instance (or occurrence) of a class. In this case, it's an actual user data structure with attached code. Objects and classes are a bit like values and data types. There is only one integer

data type, but there are many possible integers. Similarly, our program defines only one user class but can create many different (or identical) users from it.
The data associated with an object are called its properties. The functions associated with an object are called its methods. When we define a class, we define the names of its properties and give the code for its methods.

Debugging and maintenance of programs is much easier if we use encapsulation. This is the idea that a class provides certain methods (the interface) to the code that uses its objects, so the outside code does not directly access the data structures of those objects. Debugging is thus easier because we know where to look for bugs—the only code that changes an object's data structures is within the class—and maintenance is easier because we can swap out implementations of a class without changing the code that uses the class, as long as we maintain the same interface.

Any nontrivial object-oriented design probably involves inheritance. This is a way of defining a new class by saying that it's like an existing class, but with certain new or changed properties and methods. The old class is called the **superclass** (or parent or base class), and the new class is called the **subclass** (or derived class). Inheritance is a form of code reuse—the base-class code is reused instead of being copied and pasted into the new class. Any improvements or modifications to the base class are automatically passed on to the derived class.

II] Creating an Object :

In PHP, we can create objects by defining classes and then instantiating them using the **new** keyword. Here's a step-by-step explanation with an example :

Step 1: Define a Class

A class is a blueprint for creating objects. It defines the properties (variables) and methods (functions) that the objects of the class will have. Here's an example class named **Person** :

```
class Person {
    // Properties (variables)
    public $name;
    public $age;
    // Constructor method
    public function __construct($name, $age) {
        $this->name = $name;
        $this->age = $age;
    }
    // Method
    public function greet() {
        echo "Hello, my name is {$this->name} and I am {$this->age} years old.";
    }
}
```

Step 2 : Create Objects

Once the class is defined, we can create objects by instantiating the class using the **new** keyword. Pass any required parameters to the class constructor if it has one :

```
// Create objects
$person1 = new Person("John", 30);
$person2 = new Person("Alice", 25);
```

In this example, **$person1** and **$person2** are two instances of the **Person** class.

Step 3 : Access Object Properties and Methods

We can access the properties and methods of an object using the arrow (`->`) operator:

```
// Access properties
echo $person1->name;        // Outputs : John
echo $person2->age;         // Outputs : 25


// Call methods
$person1->greet();     // Outputs : Hello, my name is John and I am 30 years old.
$person2->greet();     // Outputs : Hello, my name is Alice and I am 25 years old.
```

In this example, we are accessing the name and age properties, as well as calling the greet method on each object.

Example:

```
    class Person {
    public $name;
    public $age;

    public function __construct($name, $age) {
        $this->name = $name;
        $this->age = $age;
    }

    public function greet() {
        echo "Hello, my name is {$this->name} and I am {$this->age} years old.";
    }
}
// Create objects
$person1 = new Person("John", 30);
$person2 = new Person("Alice", 25);
```

```
// Access properties
echo $person1->name;        // Outputs : John
echo $person2->age;         // Outputs : 25

// Call methods
$person1->greet();    // Outputs : Hello, my name is John and I am 30 years old.
$person2->greet();    // Outputs : Hello, my name is Alice and I am 25 years old.
```

This example demonstrates the basic process of creating objects in PHP. We define a class, instantiate objects from that class and then access the properties and methods of those objects.

III] Accessing Properties and Methods :

In PHP, objects are instances of classes, and you can access their properties and methods using the arrow (**'->'**) operator. Here's a basic overview :

Accessing Properties :

Assuming we have a class **'Person'** with a property **'$name'** :

```
class Person {
    public $name;

    public function __construct($name) {
        $this->name = $name;
    }
}
// Create an instance of the Person class
$person = new Person("John");

// Access the property
echo $person->name;                    // Outputs : John
```

Accessing Methods :

Assuming you have a class **Calculator** with a method '**add'** :

```
class Calculator {
    public function add($a, $b) {
        return $a + $b;
    }}
// Create an instance of the Calculator class
$calculator = new Calculator();
```

```
// Call the method
$result = $calculator->add(5, 3);

echo $result;                    // Outputs: 8
```

Accessing Methods with Object Properties :

Methods can also use object properties:

```
class Circle {
    public $radius;

    public function __construct($radius) {
        $this->radius = $radius;
    }
    public function calculateArea() {
        return pi() * pow($this->radius, 2);
    }
}
// Create an instance of the Circle class
$circle = new Circle(5);

// Call the method that uses the object property
$area = $circle->calculateArea();

echo $area;                      // Outputs : 78.54
```

In the example above, the **calculateArea** method accesses the **$radius** property of the object using **$this->radius**.

Remember, properties and methods can have different visibility levels (public, private, protected), affecting where they can be accessed from. The examples above assume public visibility for simplicity.

IV] Declaring Class :

Declaring a class in programming involves defining a blueprint or a template for creating objects. In the context of object-oriented programming (OOP), a class serves as a blueprint that encapsulates data (in the form of properties or attributes) and the behaviours (in the form of methods or functions) that operate on that data.

Here's a breakdown of the key components involved in declaring a class :

1. **Class Keyword :** In most object-oriented programming languages, including PHP, we use the **class** keyword to declare a class.

```
class MyClass {
    // Class definition goes here
}
```

2. **Properties (Attributes) :** These are variables that hold data within the class. Properties define the characteristics or attributes of the objects created from the class.

```
class MyClass {
    public $property1;          // Public property
    private $property2;         // Private property
}
```

3. **Methods (Functions):** These are functions defined within the class that operate on the class's data. Methods represent the behaviors associated with the objects created from the class.

```
class MyClass {
    public function myMethod() {
        // Method implementation goes here
    }
}
```

4. **Constructor Method :** This is a special method that is automatically called when an object is created from the class. It is used to initialize the object's properties or perform other setup tasks.

```
class MyClass {
    public function __construct() {
        // Constructor implementation goes here
    }
}
```

5. **Access Modifiers :** In many OOP languages, including PHP, we can use access modifiers like **public**, **private**, and **protected** to control the visibility of properties and methods.

```
class MyClass {
    public $publicProperty;             // Accessible from outside the class
    private $privateProperty;           // Accessible only within the class
    protected $protectedProperty;       // Accessible within the class and its subclasses
}
```

Declaring a class allows we to create instances or objects based on that class, each with its own set of properties and the ability to perform actions defined by the methods of the class. Classes provide a way to structure and organize code in a modular and reusable manner, a fundamental principle of object-oriented programming.

V] Introspection :

In PHP, introspection refers to the ability to examine and manipulate the structure and behaviour of code at runtime. PHP provides several functions and techniques for introspection, allowing us to inspect classes, objects, functions, and other elements of our code dynamically. Here are some common introspection techniques in PHP :

1. **get_defined_vars():** This function returns an array of all defined variables in the current scope. It's useful for inspecting the variables available in a certain context.

```
print_r(get_defined_vars());
```

Example :

```
function exampleFunction() {
    $var1 = 'Hello';
    $var2 = 42;

    print_r(get_defined_vars());
}
exampleFunction();
```

Output :

```
Array
(
    [var1] => Hello
    [var2] => 42
)
```

2. **get_class() and get_object_vars() :** These functions are used to inspect objects. **get_class()** returns the class name of an object, and **get_object_vars()** returns an associative array of object properties.

```
class Person{
    public $age;
    private $name;
}
$obj = new Person();
echo get_class($obj);                    // Output : Person
print_r(get_object_vars($obj));          // Output : Array ( [age] =>
                                                      [name] => )
```

3. **get_class_methods() :** This function returns an array of method names for a given class.

```
class MyClass {
    public function getdata() {}
    private function displaydata() {}
```

```
}
$obj = new MyClass();
print_r(get_class_methods($obj));
```

Output :

```
Array
(
   [0] => getdata
   [1] => displaydata
)
```

4. **method_exists() :** This function checks if a class or object has a particular method.

```
class MyClass {
   public function getdata() {}
}
$obj = new MyClass();
if (method_exists($obj, 'getdata')) {
   echo "Method exists!";
}
```

Output :

```
Method exists !
```

5. **class_exists() :** This function checks if a class has been defined.

```
class MyClass {
   public function getdata() {}
}
if (class_exists('MyClass')) {
   echo "Class exists !";
}
```

Output :

```
Class exists !
```

6. **Reflection :** The Reflection API provides a more powerful and flexible way to inspect classes, methods, and properties. It allows you to retrieve detailed information about classes and their members.

```
class MyClass {
   public function getdata() {}
}
$reflectionClass = new ReflectionClass('MyClass');
echo $reflectionClass->getName().”\n”;

$methods = $reflectionClass->getMethods();
foreach ($methods as $method) {
   echo $method->getName();
```

```
}
```

Output :

MyClass
getdata

These are just a few examples of introspection in PHP. Depending on our specific use case, we may find other functions or techniques more suitable for our needs. Introspection is a powerful feature that can be used for debugging, dynamic code generation, and other advanced programming tasks.

VI] Serialization :

Serialization in PHP refers to the process of converting complex data structures, such as arrays or objects, into a format that can be easily stored or transmitted and later reconstructed. The serialized data is typically a string representation of the original data, and it can be stored in files, databases, or sent over networks. PHP provides two main functions for serialization : **serialize** and **unserialize**.

1. serialize() Function :

The **serialize** function in PHP is used to convert a variable (typically an array or an object) into a storable string representation.

```
$data = array("name" => "John", "age" => 30, "city" => "New York");
$serializedData = serialize($data);
echo “Serialized Data :”.$serializedData;
// $serializedData is a string that can be stored or transmitted
```

Output :

```
Serialized Data: a:3:{s:4:"name";s:4:"John";s:3:"age";i:30;s:4:"city";s:8:"New York";}
```

The serialized data starts with **a:3:{,** indicating an array with three elements. Each element is represented with a **key** and a **value**, where s stands for string and i stands for integer. The **s:4**, **s:3**, and **s:4** indicate the lengths of the strings "**name**," "**age**," and "**city**," respectively.

2. unserialize() Function :

The **unserialize** function is used to recreate the original variable from a serialized string.

```
$originalData = unserialize($serializedData);
// $originalData is an array with the original data
```

Example :

```
$serializedData = 'a:3:{s:4:"name";s:8:"John Doe";s:3:"age";i:30;s:4:"city";s:8:"New York";}';
$unserializedData = unserialize($serializedData);
```

```
echo "Unserialized Data:\n";
print_r($unserializedData);
```

Output :

```
Unserialized Data:
Array
(
        [name] => John
        [age] => 30
        [city] => New York
)
```

Use Cases :

1. **Data Storage :** Serialized data can be stored in files or databases, allowing us to save the state of complex data structures.
2. **Session Handling :** In web development, we can use serialization to store and retrieve complex data in session variables.
3. **Interprocess Communication :** When passing data between different processes or servers, serialization allows us to convert data into a format that can be easily transmitted and reconstructed on the receiving end.

It's important to note that not all data types can be serialized. Resources (like database connections or file handles) and objects that contain them cannot be serialized. Additionally, when using **serialize** and **unserialize**, ensure that the classes of the serialized objects are defined before attempting to unserialize them.

❖ Web Techniques :

PHP was designed as a web-scripting language and although it is possible to use it in purely command-line and GUI scripts, the Web accounts for the vast majority of PHP uses. A dynamic website may have forms, sessions, and sometimes redirection, and this unit explains how to implement those things in PHP. We will learn how PHP provides access to form parameters and uploaded files, how to send cookies and redirect the browser, how to use PHP sessions, and more.

I] HTTP Basics :

The Web runs on HTTP, or HyperText Transfer Protocol. This protocol governs how web browsers request files from web servers and how the servers send the files back. To understand the various techniques, we'll show in this unit, but we need to have a basic understanding of HTTP.

1. **HTTP Requests :**
   - When a web browser wants to retrieve a web page, it sends an HTTP request message to a web server.
   - The request message includes header information and, optionally, a body.

- The first line of an HTTP request includes the HTTP method (e.g., GET), the address of the requested document ("/index.html"), and the version of the HTTP protocol (HTTP/1.1).
  Example :

  ```
  GET /index.html HTTP/1.1
  ```

- Following the initial line, the request may contain optional header information (e.g., User-Agent and Accept headers) and a blank line to mark the end of the header section.
- The request may also include additional data, depending on the HTTP method used (e.g., POST).

2. HTTP Responses :

- The web server processes the request and sends back an HTTP response message.
- The first line of an HTTP response includes the protocol version, a status code, and a description of the status.
  Example :

  ```
  HTTP/1.1 200 OK
  ```

- In this example, the status code "200" indicates a successful request.
- Following the status line, the response contains headers providing additional information (e.g., Date, Server, Content-Type, Content-Length).
- After the headers, there is a blank line, and then the response may include the requested data if the request was successful.

3. Common HTTP Methods :

- The two most common HTTP methods are GET and POST.
- GET is used for retrieving information from the server, such as documents, images, or query results.
- POST is used for sending information to the server, often used for submitting forms or posting sensitive data like credit card numbers.
- The choice between GET and POST can depend on whether information is being retrieved or submitted, and this is often determined by user actions (e.g., clicking a link uses GET, submitting a form can use either GET or POST).

This overview provides a foundational understanding of how HTTP facilitates communication between web browsers and servers, emphasizing the importance of methods (GET and POST) and the structure of request and response messages.

II] Variables :

Server configuration and request information — including form parameters and cookies — are accessible in three different ways from our PHP scripts, as described in this section. Collectively, this information is referred to as **EGPCS** in PHP, which stands for Environment, GET, POST, Cookies, and Server. These represent different sources of information that PHP scripts can access.

PHP creates six global arrays to store this information.

The Global Arrays are :

1. $_COOKIE :

- Contains any cookie values passed as part of the request.
- Keys of the array are the names of the cookies.

```
$cookieValue = $_COOKIE['cookie_name'];
```

2. $_GET :

- Contains any parameters that are part of a GET request.
- Keys of the array are the names of the form parameters.

```
$getParam = $_GET['parameter_name'];
```

3. $_POST :

- Contains any parameters that are part of a POST request.
- Keys of the array are the names of the form parameters.

```
$postParam = $_POST['parameter_name'];
```

4. $_FILES :

- Contains information about any uploaded files.
- Used when dealing with file uploads in HTML forms.

```
$uploadedFile = $_FILES['file_input'];
```

5. $_SERVER :

- Contains useful information about the web server.
- Provides details such as server name, request method, script filename, etc.

```
$serverName = $_SERVER['SERVER_NAME'];
```

6. $_ENV :

- Contains the values of any environment variables.
- Keys of the array are the names of the environment variables.

```
$envVar = $_ENV['environment_variable'];
```

7. $_REQUEST :

- Automatically created by PHP.
- **Use :** Collect data from both GET and POST requests.

```
$requestParam = $_REQUEST['parameter_name'];
```

These variables are global, meaning they can be accessed from anywhere in your PHP script, including within function definitions. They provide a convenient way to access various types of information related to the current request and environment.

III] Server Information :

The **$_SERVER** variable in PHP provides valuable information about the web server and the current request. Here's an explanation of some key entries in the **$_SERVER** array :

1. **PHP_SELF :**
   - The name of the current script, relative to the document root.
   - Useful for creating self-referencing scripts.

```
$currentScript = $_SERVER['PHP_SELF'];
```

2. **SERVER_SOFTWARE :**
   - A string that identifies the server, including the server software and version.

```
$serverSoftware = $_SERVER['SERVER_SOFTWARE'];
```

3. **SERVER_NAME :**
   - The hostname, DNS alias, or IP address for self-referencing URLs.

```
$serverName = $_SERVER['SERVER_NAME'];
```

4. **GATEWAY_INTERFACE :**
   - The version of the CGI standard being followed.

```
$cgiVersion = $_SERVER['GATEWAY_INTERFACE'];
```

5. **SERVER_PROTOCOL :**
   - The name and revision of the request protocol.

```
$protocol = $_SERVER['SERVER_PROTOCOL'];
```

6. **SERVER_PORT :**
   - The server port number to which the request was sent.

```
$serverPort = $_SERVER['SERVER_PORT'];
```

7. **REQUEST_METHOD:**
   - The method the client used to fetch the document (e.g., "GET" or "POST").

```
$requestMethod = $_SERVER['REQUEST_METHOD'];
```

8. **PATH_INFO :**
   - Extra path elements given by the client.

```
$pathInfo = $_SERVER['PATH_INFO'];
```

9. PATH_TRANSLATED :

- The value of PATH_INFO, translated by the server into a filename.

```
$pathTranslated = $_SERVER['PATH_TRANSLATED'];
```

10. SCRIPT_NAME :

- The URL path to the current page, useful for self-referencing scripts.

```
$scriptName = $_SERVER['SCRIPT_NAME'];
```

11. QUERY_STRING :

- Everything after the "?" in the URL.

```
$queryString = $_SERVER['QUERY_STRING'];
```

12. REMOTE_HOST:

- The hostname of the machine that requested this page.

```
$remoteHost = $_SERVER['REMOTE_HOST'];
```

13. REMOTE_ADDR :

- The IP address of the machine that requested this page.

```
$remoteAddr = $_SERVER['REMOTE_ADDR'];
```

14. AUTH_TYPE :

- If the page is password-protected, this is the authentication method used.

```
$authType = $_SERVER['AUTH_TYPE'];
```

15. REMOTE_USER :

- If the page is password-protected, this is the username with which the client authenticated.

```
$remoteUser = $_SERVER['REMOTE_USER'];
```

16. REMOTE_IDENT :

- If the server is configured to use identd (RFC 931) identification checks, this is the username fetched from the host that made the web request.

```
$remoteIdent = $_SERVER['REMOTE_IDENT'];
```

17. CONTENT_TYPE :

- The content type of the information attached to queries such as PUT and POST.

```
$contentType = $_SERVER['CONTENT_TYPE'];
```

18. CONTENT_LENGTH:

- The length of the information attached to queries such as PUT and POST.

```
$contentLength = $_SERVER['CONTENT_LENGTH'];
```

Additionally, the Apache server creates entries in the **$_SERVER** array for each HTTP header in the request. Header names are converted to uppercase, hyphens (-) are turned into underscores (*), and the string "HTTP*" is prepended.

1. HTTP_USER_AGENT :

- The string the browser used to identify itself.

```
$userAgent = $_SERVER['HTTP_USER_AGENT'];
```

2. HTTP_REFERER :

- The page the browser said it came from to get to the current page.

```
$referer = $_SERVER['HTTP_REFERER'];
```

These **$_SERVER** entries are useful for understanding the context of the current request and can be used in various ways, such as logging, security checks, or dynamically generating content based on server and client information.

IV] Processing Forms :

When we submit a form in HTML, the data is sent to the server for processing. Depending on the form's method attribute (either "GET" or "POST"), the form data is made available in the **$_GET** or **$_POST** super global arrays in PHP.

Here's a brief explanation of how we can access form data submitted through web elements using **$_GET** and **$_POST** :

Using '$_GET' :

```
<!-- HTML form with method="get" -->
<form action="process_form.php" method="get">
    <label for="username">Username:</label>
    <input type="text" name="username" id="username">
    <input type="submit" value="Submit">
</form>
```

In the PHP script (**process_form.php**), we can access the submitted data using **$_GET** :

```
<?php
if (isset($_GET['username'])) {
    $username = $_GET['username'];
    echo "Submitted username via GET: " . htmlspecialchars($username);
} else {
    echo "No username submitted.";
}
?>
```

Using '$_POST' :

```
<!-- HTML form with method="post" -->
<form action="process_form.php" method="post">
    <label for="username">Username:</label>
    <input type="text" name="username" id="username">
    <input type="submit" value="Submit">
</form>
```

In the PHP script (process_form.php), access the submitted data using $_POST :

```
<?php
if (isset($_POST['username'])) {
    $username = $_POST['username'];
    echo "Submitted username via POST: " . htmlspecialchars($username);
} else {
    echo "No username submitted.";
}
?>
```

In both cases, when the form is submitted, the data is sent to the specified PHP script (**process_form.php** in these examples), and we can access the form fields through the respective super global (**$_GET** or **$_POST**). Always remember to validate and sanitize user input to enhance security and prevent issues such as SQL injection or Cross-Site Scripting (XSS) attacks.

V] Setting Response Headers :

HTTP headers are part of the communication between a web server and a client (typically a web browser) and provide information about the response.

Here's a detailed explanation of the provided information:

1. **HTTP Response Headers :**
    - HTTP responses contain headers that convey metadata about the response, such as content type, length, server information, etc.
    - PHP and Apache usually handle the headers automatically, determining content type and other details based on the response content.

2. **Manual Header Setting with header() function :**
    - If you need to customize headers, like specifying a content type, setting expiration times, redirecting the browser, or generating specific HTTP errors, you use the **'header()'** function in PHP.
    - Example: **<?php header("Content-Type: text/plain"); ?>** sets the content type to plain text.

3. **Order of Header Setting :**
    - Headers must be set before any content is generated and sent to the client.
    - This means that all calls to **header()** (or **setcookie()**, if you're dealing with cookies) should happen at the top of your PHP file, even before the <html> tag.

4. **Headers and HTML Output :**
    - Trying to set headers after the document has started (i.e., after any HTML output) will result in a warning.
    - The warning message is: "Cannot add header information - headers already sent."

5. **Output Buffering as a Solution :**
    - To avoid the warning about headers already being sent, you can use output buffering.
    - Output buffering is a mechanism where the output is stored in a buffer rather than being sent to the client immediately.
    - Functions like ob_start(), ob_end_flush(), and related functions are used to control output buffering.

6. **Example Usage :**

```
<?php
ob_start();  // Start output buffering
header("Content-Type: text/plain");  // Set content type header

// ... Your PHP code and content generation ...

ob_end_flush();  // Flush the output buffer and send the content to the client
?>
```

In summary, when working with headers in PHP, it's crucial to set them before any output is sent to the client, and output buffering can be used to achieve this when dealing with dynamic content generation.

VI] Cookies :

Cookies are small pieces of data that websites send to a user's web browser while the user is browsing. These data files are stored on the user's device and are designed to hold a modest amount of specific information about the user and the website. Cookies serve various purposes, including improving the user experience, remembering user preferences, and providing anonymized tracking data to website owners.

Here are some key points about cookies :

- **Storage on the User's Device :** Cookies are stored as text files on the user's device (computer, smartphone, or tablet) within the browser's directory.
- **Information Exchange :** When a user visits a website, the server sends a cookie to the browser. The browser then stores the cookie for future use. The next time the user visits the same website, the browser sends the stored cookie back to the server, providing information about the user's previous activity.

Types of Cookies :

- **Session Cookies:** These are temporary cookies that are erased when the user closes the web browser. They are used to store temporary information (e.g., shopping cart contents) during a browsing session.

- **Persistent Cookies:** These cookies remain on the user's device for a specified period or until manually deleted. They are used for purposes like remembering login information or user preferences.

Purpose of Cookies :

- **Authentication :** Cookies can be used to identify a user and maintain their logged-in status.
- **Personalization :** Cookies help websites remember user preferences, such as language settings and personalized content.
- **Tracking :** Website owners use cookies for analytics and tracking user behavior, which helps in improving website performance and user experience.
- **Targeted Advertising :** Cookies may be used to track users across websites, allowing advertisers to deliver more targeted and relevant ads.
- **Security Considerations :** While cookies are generally harmless, concerns arise when they are misused or if they store sensitive information. Secure practices involve encrypting sensitive data and using secure connections (HTTPS) for transmitting cookies.

It's important to note that there are privacy considerations with cookies, and regulations such as the General Data Protection Regulation (GDPR) and others may impose restrictions on how cookies are used and require explicit user consent for certain types of tracking cookies.
The provided information explains the basics of working with cookies in web development using PHP. Let me summarize the key points :

Creating Cookies :

To send a cookie to the browser, we use the **setcookie()** function in PHP. The syntax is as follows:

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

- **name :** A unique name for the cookie.
- **value :** The data payload of the cookie.
- **expire :** Expiration date for the cookie.
- **path :** The URL path for which the cookie is valid.
- **domain :** The domain for which the cookie is valid.

- **secure :** Specifies whether the cookie should only be transmitted over HTTPS.

Accessing Cookies :

When a browser sends a cookie back to the server, we can access it through the $_COOKIE array in PHP. The key is the cookie name, and the value is the cookie's payload.

```
$cookieValue = $_COOKIE['cookieName'];
```

Example Usage :

Here's an example that keeps track of the number of times a page has been accessed by a client :

```
$pageAccesses = isset($_COOKIE['accesses']) ? $_COOKIE['accesses'] : 0;
setcookie('accesses', ++$pageAccesses);
```

Note :

- Cookies are sent as headers in the response, so '**setcookie()'** must be called before any part of the document body is sent.
- Cookie names must be unique, and they must not contain whitespace or semicolons.
- The **'expire'** parameter determines when the cookie will expire. If not specified, the cookie will be a session cookie.

VII] Session :

In web development, a session is a mechanism that allows you to preserve data across subsequent accesses to a website within a defined period. Sessions are crucial for maintaining user-specific information, such as login credentials, shopping cart contents, and other user preferences, throughout their interaction with a web application.

In PHP, sessions are often implemented using the **$_SESSION** super global variable. Here's a basic overview of how sessions work in PHP:

Basic Session Usage :

1. **Starting a Session :**
   To use sessions in PHP, we need to start a session. This is typically done at the beginning of each page where you want to use session variables.

   ```
   <?php
      session_start();
   ?>
   ```

2. **Setting Session Variables :**
   We can store data in session variables, making it accessible across different pages during a user's visit.

```php
<?php
// Setting a session variable
    $_SESSION['username'] = 'john_doe';
?>
```

3. **Accessing Session Variables :**
   Retrieve session data on subsequent pages.

```php
<?php
// Accessing a session variable
    $username = $_SESSION['username'];
    echo "Welcome, $username!";
?>
```

4. **Destroying a Session:**
   To end a session, use **session_destroy().**

```php
<?php
// Ending a session
    session_destroy();
?>
```

Session Security :

1. **Session Timeout :**
   Set a session timeout to automatically end a session after a period of inactivity.

```php
<?php
// Set session timeout (e.g., 30 minutes)
    ini_set('session.gc_maxlifetime', 1800);
    session_set_cookie_params(1800);
    session_start();
?>
```

2. **Regenerate Session ID:**
   Regenerate the session ID periodically to enhance security.

```php
<?php
// Regenerate session ID
    session_regenerate_id(true);
?>
```

Handling Login Sessions :

1. **Login Example :**
   Use sessions to manage user authentication.

```php
<?php
// After successful login
```

```
session_start();
$_SESSION['user_id'] = $user_id;
header('Location: dashboard.php'); exit(); ?>
```

2. Check Authentication on Protected Pages :

On protected pages, check if the user is authenticated using sessions.

```
<?php
session_start();
if (!isset($_SESSION['user_id'])) {
    header('Location: login.php');
    exit();
}
?>
```

Sessions play a crucial role in web development, especially when managing user authentication, preserving user-specific data, and maintaining state across multiple pages. Always handle sessions securely to protect user data and ensure the reliability of your web application.

VIII] Files :

Handling files is a common task in web development, and it involves tasks such as uploading files, downloading files, reading from or writing to files, and managing file systems. Here are some common techniques and considerations related to files in web development:

1. **File Uploads :**
   - Allow users to upload files through HTML forms.
   - Use the **<input type="file">** element in HTML forms.
   - PHP provides the **'$_FILES'** super global to handle file uploads.
   - Validate file types, sizes, and perform server-side security checks.

Example PHP code for file upload :

```
<?php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["file"]["name"]);
move_uploaded_file($_FILES["file"]["tmp_name"], $target_file);
?>
```

2. **File Downloads :**
   - Set appropriate headers for file downloads, including content type and content disposition.
   - Use server-side scripting (e.g., PHP) to control access to files and enforce security.

Example PHP code for file download :

```
<?php
$file = "path/to/file.txt";
header("Content-type: application/octet-stream");
header("Content-Disposition: attachment; filename=" . basename($file));
readfile($file);  ?>
```

3. **File I/O Operations :**
   - Use server-side languages like PHP, Python, or Node.js for file input/output operations.
   - Read from and write to files using functions provided by the programming language.
   - Implement error handling and security measures when dealing with file operations.

Example PHP code for reading from a file :

```
<?php
$file = fopen("example.txt", "r");
while (!feof($file)) {
    echo fgets($file) . "<br>";
}
fclose($file);
?>
```

4. **File System Management :**
   - Be cautious about file paths and permissions to prevent security vulnerabilities.
   - Use server-side scripting to interact with the file system (create, delete, move files and directories).
   - Implement access controls to restrict user actions on the file system.

Example PHP code for file system management:

```
<?php
$file_path = "path/to/file.txt";
if (file_exists($file_path)) {
    unlink($file_path); // Delete the file
}
?>
```

5. **Client-Side File Handling :**
   - Use JavaScript for client-side file operations, like reading files using the FileReader API.
   - Be aware of browser security restrictions when dealing with client-side file handling.

Example JavaScript code for reading a file on the client side:

```
const fileInput = document.getElementById('fileInput');
fileInput.addEventListener('change', (event) => {
    const file = event.target.files[0];
    const reader = new FileReader();
    reader.onload = function(e) {
        console.log(e.target.result); // File content
    };
    reader.readAsText(file);
});
```

Remember to implement proper error handling, input validation, and security measures when working with files to prevent vulnerabilities and ensure the robustness of our web application.

IX] Maintaining State :

It introduces various techniques for session tracking and highlights that PHP has a built-in session-tracking system that uses cookies or URLs to manage state. Let's break down the information :

1. **Statelessness of HTTP :**
    - HTTP is a stateless protocol, meaning that each request from a client to a server is independent, and the server does not retain information about previous requests.
    - This statelessness poses a challenge for applications that require continuity of user actions, such as maintaining a shopping cart.

2. **Session Tracking Techniques :**

    - **Hidden Form Fields :**
        - Uses hidden form fields to pass information between requests.
        - Values are accessible in the **$_GET** and **$_POST** arrays.
        - Can be used to pass information like the contents of a shopping cart.

    - **URL Rewriting :**
        - Modifies URLs dynamically to include extra information (parameters).
        - For example: **http://www.example.com/catalog.php?userid=123.**
        - Allows tracking users through all dynamically generated documents.

    - **Cookies :**
        - A server sends a small piece of information (cookie) to the client, which is then sent back with every subsequent request.
        - Useful for retaining information across multiple visits by the same browser.
        - Note: Some users may disable cookies in their browsers, so it's essential to have a fallback mechanism.

    - **PHP Session Tracking :**
        - PHP provides a built-in session-tracking system.
        - Allows the creation of persistent variables accessible across different pages and visits by the same user.
        - Behind the scenes, PHP's session tracking uses cookies or URLs to manage state details automatically.

3. **Challenges with Cookies :**
    - Cookies are a commonly used technique for session tracking, but they have limitations.
    - Users can disable cookies, so applications using cookies for state maintenance need alternative mechanisms as a fallback.

4. **PHP's Session-Tracking System :**
    - PHP's session tracking is recommended as the best way to maintain state.
    - It allows the creation of persistent variables that can be accessed across different pages and visits.

- Handles the details of state management, using cookies or URLs behind the scenes.

Here's a basic example of using PHP sessions :

```
<?php
session_start();        // Start the session

// Set session variables
$_SESSION['user_id'] = 123;
$_SESSION['username'] = 'example_user';

// Access session variables on other pages
$user_id = $_SESSION['user_id'];
$username = $_SESSION['username'];
?>
```

In this example, **session_start()** initializes or resumes a session, and **$_SESSION** is used to store and retrieve session variables. The session ID is typically managed using cookies or URL rewriting by PHP's session mechanism.

X] SSL :

SSL (Secure Sockets Layer) it emphasizes that PHP itself doesn't control the encryption related to SSL but mentions a way to check whether a PHP page was generated over an SSL connection using the **$_SERVER** array.

Let's break down the key points :

1. **SSL and PHP :**
   - SSL provides a secure channel for transmitting HTTP requests and responses.
   - PHP doesn't directly control SSL encryption; it relies on the web server's SSL configuration.

2. **Detecting SSL in PHP :**

   - The **$_SERVER['HTTPS']** variable can be used to check if a PHP page was generated in response to an SSL-encrypted connection.
   - If **$_SERVER['HTTPS']** is set to 'on', it indicates that the connection is secure.

Example PHP code to enforce a secure connection :

```
if ($_SERVER['HTTPS'] !== 'on') {
   die("Must be a secure connection.");
}
```

This code snippet checks if the current page was accessed over an SSL connection and terminates the script if not, indicating that it must be a secure connection.

1. **Ensuring Form Security :**
   - The information warns against a common mistake where a form is sent over a secure connection (https://), but the form's action attribute points to an insecure URL (http://).
   - If the form action is not set to an HTTPS URL, form data may be transmitted over an insecure connection, making it vulnerable to interception by packet sniffers.

Example of a potential issue :

```
<form action="http://www.example.com/process_form.php" method="post">
   <!-- Form fields -->
   <input type="submit" value="Submit">
</form>
```

To address this, ensure that the form's action attribute uses the secure (https://) URL :

```
<form action="https://www.example.com/process_form.php" method="post">
   <!-- Form fields -->
   <input type="submit" value="Submit">
</form>
```

This way, form data will be transmitted securely over HTTPS.

These practices are important for maintaining the security of web applications, especially when handling sensitive information. Checking and enforcing secure connections help protect user data from potential security threats associated with unencrypted communication.

❖ Databases :

PHP has support for over 20 databases, including the most popular commercial and open-source varieties. Relational database systems such as MySQL, PostgreSQL, and Oracle is the backbone of most modern dynamic websites. In these are stored shopping-cart information, purchase histories, product reviews, user information, credit card numbers, and sometimes even web pages themselves.

This unit covers how to access databases from PHP. We focus on the built-in PHP Data Objects (or PDO) system, which lets we use the same functions to access any database, rather than on the myriad database-specific extensions. In this unit, we'll learn how to fetch data from the database, store data in the database, and handle errors. We finish with a sample application that shows how to put various database techniques into action.

I] Relational Database :

1. **Relational Database Management System (RDBMS) :**
    - Manages structured data in tables.
    - Tables have columns with names and types.
    - Databases consist of tables, and RDBMS has a user system for access control.

2. **PHP and SQL Interaction :**
    - PHP communicates with RDBMS like MySQL and Oracle using SQL.
    - SQL has two main parts: Data Manipulation Language (DML) for retrieving and modifying data, and Data Definition Language (DDL) for creating and modifying database structures.

3. **Data Manipulation Language (DML) :**
    - Consists of SELECT, INSERT, UPDATE, and DELETE statements.
    - Examples include inserting a new row, deleting records, updating values and querying data.

4. **Data Definition Language (DDL) :**
    - Used to create and modify database structures.
    - Syntax is not as standardized as DML, but it's supported by the specific RDBMS.

5. **Sample SQL Statements :**
    - Examples of INSERT, DELETE, UPDATE, and SELECT statements.
    - Illustration of specifying columns for INSERT and short-form for table names in SELECT.

6. **Querying Data from Multiple Tables :**
    - Examples of joining tables using SELECT to fetch information from multiple tables.

II] Procedure to connect PHP with Database :

Connecting PHP with a database typically involves a few steps. Below is a step-by-step procedure to connect PHP with a MySQL database :

1. Install a Database Server :

Ensure that we have a database server installed. For **MySQL**, we can use tools like **XAMP, WAMP, OR install MySQL separately.**

2. Create a Database :

Use a database management tool (**e.g., phpMyAdmin**) or command line to create a database.

3. Create a Database User :

Create a user and grant necessary privileges to access the database.

4. Install PHP and a Web Server :

Install PHP on your system and set up a web server (e.g., XAMP , WAMP , LAMP).

5. Install MySQL Extension for PHP (if not already installed) :

Use the following command to install the MySQL extension for PHP :

```
sudo apt-get install php-mysql
```

For other operating systems, use the relevant package manager or download the extension manually.

6. Create a PHP File for Database Connection :

Create a PHP file (**e.g., connect.php**) to handle the database connection.

```
<?php
// Database configuration
$host = "localhost";
$username = "your_username";
$password = "your_password";
$database = "your_database";

// Create connection
$conn = new mysqli($host, $username, $password, $database);

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

Replace "your_username", "your_password", and "your_database" with your actual database credentials.

7. Use the Connection in Other PHP Files :

Include the **connect.php** file in other PHP files where we need database connectivity.

```
<?php
include "connect.php";
// Your PHP code using the database connection goes here
?>
```

8. Perform Database Operations :

We can now perform various database operations using PHP, such as querying, inserting, updating, and deleting data.

9. Close the Database Connection (Optional) :

It's a good practice to close the database connection when we're done with it.

```
<?php
// Close the connection
$conn->close();
?>
```

If we are working with a different database system, the connection details and PHP extensions might vary. Adjust the code accordingly based on your specific database setup.

III] Advanced Database Techniques that are used in PHP :

PHP is often used in conjunction with databases to create dynamic and interactive web applications. There are several advanced database techniques and concepts that can be applied when working with PHP and databases. Here are some of them:

1. Object-Relational Mapping (ORM) :

- ORM is a technique that allows you to interact with a database using PHP objects instead of raw SQL queries. Popular PHP ORM libraries include Doctrine and Eloquent (used in Laravel).

2. Database Abstraction Layers :

- Use of database abstraction layers allows developers to write code that is independent of the underlying database system. Libraries like PDO (PHP Data Objects) provide a uniform method of access to multiple database systems.

3. Prepared Statements :

- Prepared statements help prevent SQL injection attacks by separating SQL code from user input. PHP supports prepared statements through PDO and MySQLi, and they should be used whenever user input is involved in database queries.

4. Transactions :

- Transactions ensure the integrity of the database by grouping a set of SQL queries into a single unit of work. If any part of the transaction fails, the entire transaction is rolled back. Transactions are essential for maintaining data consistency.

5. Caching :

- Caching database query results can significantly improve performance by reducing the need to execute the same query repeatedly. Tools like Memcached or Redis can be used for caching.

6. Database Indexing :

- Proper indexing of database tables can dramatically improve query performance. Understanding the types of queries your application will run and adding appropriate indexes is a crucial optimization technique.

7. Database Sharding :

- Sharding involves splitting a large database into smaller, more manageable parts called shards. Each shard is a separate database with its own set of data. This technique is useful for scaling horizontally and distributing data across multiple servers.

8. Database Connection Pooling :

- Connection pooling involves reusing existing database connections rather than opening a new connection for each request. This can improve performance by reducing the overhead associated with opening and closing connections.

9. Data Warehousing :

- Data warehousing involves the extraction, transformation, and loading (ETL) of data from different sources into a centralized repository. Data can be transformed and aggregated for reporting and analytics purposes.

10. Database Version Control :

- Version control for databases ensures that changes to the database schema are tracked over time. Tools like Flyway or Liquibase can be used to manage database schema migrations and versioning.

11. Full-Text Search :

- Full-text search capabilities can be implemented using databases that support full-text indexing. This is useful for applications that require advanced search functionality.

12. Database Partitioning :

- Database partitioning involves dividing large database tables into smaller, more manageable pieces called partitions. Each partition can be stored on different physical disks, improving query performance.

When working with databases in PHP, it's essential to consider these advanced techniques based on the specific requirements and scale of your application. The choice of technique often depends on factors such as performance, scalability, and the complexity of the application.

IV] MySQL Database Basics :

MySQL is a popular open-source relational database management system (RDBMS) that is widely used in web development. Here are some fundamental concepts and basics of working with MySQL databases :

1. **Database :**

- A database is a collection of related data. In MySQL, you can create multiple databases to organize and separate different sets of data.

```
CREATE DATABASE mydatabase;
```

2. **Table :**

- A table is a structured way to organize data in a database. It consists of rows and columns. Each column has a specific data type, and each row represents a record.

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50),
    email VARCHAR(255)
);
```

3. **Column :**

- Columns define the attributes or fields of a table. Each column has a data type that specifies the kind of data it can store.

4. **Primary Key :**

- A primary key is a unique identifier for a record in a table. It ensures that each record can be uniquely identified.

```
id INT AUTO_INCREMENT PRIMARY KEY
```

5. **Data Types :**

- MySQL supports various data types, including INT (integer), VARCHAR (variable-length string), TEXT (long text), DATE (date), and more.

6. **INSERT Statement :**

- The INSERT statement is used to add new records (rows) to a table.

```
INSERT INTO users (username, email) VALUES ('john_doe', 'john@example.com');
```

7. **SELECT Statement :**

- The SELECT statement retrieves data from one or more tables. It can be used to query specific columns, apply filters, and sort the results.

```
SELECT * FROM users WHERE username = 'john_doe';
```

8. **UPDATE Statement :**

- The UPDATE statement modifies existing records in a table.

```
UPDATE users SET email = 'john_new@example.com' WHERE username = 'john_doe';
```

9. DELETE Statement :

- The DELETE statement removes records from a table based on a specified condition.

```
DELETE FROM users WHERE username = 'john_doe';
```

10. Index :

- Indexes are used to optimize query performance. They provide a quick lookup mechanism, allowing the database engine to locate data more efficiently.

```
CREATE INDEX idx_username ON users (username);
```

11. Foreign Key :

- A foreign key is a field that refers to the primary key in another table. It establishes a link between the two tables.

```
CREATE TABLE orders (
    order_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    order_date DATE,
    FOREIGN KEY (user_id) REFERENCES users(id)
);
```

1. Normalization :

- Normalization is the process of organizing data to reduce redundancy and improve data integrity. It involves breaking down large tables into smaller, related tables.

2. Transaction :

- A transaction is a sequence of one or more SQL statements that are executed as a single unit of work. Transactions ensure the consistency and integrity of the database.

```
START TRANSACTION;
-- SQL statements
COMMIT;
```

These are some of the fundamental concepts and SQL statements used in MySQL. Understanding these basics is crucial for effectively working with databases and building robust and efficient web applications.

V] PHP Data Objects (PDO) :

The PHP Data Objects (PDO) extension defines a lightweight, consistent interface for accessing databases in PHP. Each database driver that implements the PDO interface can expose database-specific features as regular extension functions. Note that we cannot perform any

database functions using the PDO extension by itself; we must use a database-specific PDO driver to access a database server.

PDO has (among others) these unique features :

- PDO is a native C extension.
- PDO takes advantage of the latest PHP 5 internals.
- PDO uses buffered reading of data from the result set.
- PDO gives common DB features as a base.
- PDO is still able to access DB-specific functions.
- PDO can use transaction-based techniques.
- PDO can interact with LOBS (Large Objects) in the database.
- PDO can use prepared and executable SQL statements with bound parameters.
- PDO can implement scrollable cursors.
- PDO has access to SQLSTATE error codes and has very flexible error handling.

Since there are a number of features here, we will only touch on a few of them to show you just how beneficial PDO can be.

First, a little about PDO. It has drivers for almost all database engines in existence, and those drivers that PDO does not supply should be accessible through PDO's generic ODBC connection. PDO is modular in that it has to have at least two extensions enabled to be active: the PDO extension itself and the PDO extension specific to the database to which you will be interfacing. See the online documentation to set up the connections for the database of your choice here. As an example, for establishing PDO on a Windows server for MySQL interaction, simply enter the following two lines into your php.ini file and restart your server:

```
extension=php_pdo.dll
extension=php_pdo_mysql.dll
```

The PDO library is also an object-oriented extension (you will see this in the code examples that follow).

Making a connection

The first thing that is required for PDO is that we make a connection to the database in question and hold that connection in a connection handle variable, as in the following code :

```
$db = new PDO ($dsn, $username, $password);
```

The $dsn stands for the data source name, and the other two parameters are self-explanatory. Specifically, for a MySQL connection, you would write the following code:

```
$db = new PDO("mysql:host=localhost;dbname=library", "petermac", "abc123");
```

Of course, we should maintain the username and password parameters as variable-based for code reuse and flexibility reasons.

Interaction with the database

So, once we have the connection to our database engine and the database that we want to interact with, we can use that connection to send SQL commands to the server. A simple UPDATE statement would look like this:

```
$db->query("UPDATE books SET authorid=4 WHERE pub_year=1982");
```

This code simply updates the books table and releases the query. This is how you would usually send non-resulting simple SQL commands (**UPDATE, DELETE, INSERT**) to the database through PDO unless you are using prepared statements, a more complex approach that is discussed in the next section.

PDO and prepared statements

PDO also allows for what are known as prepared statements. This is done with PDO calls in stages or steps. Consider the following code :

```
$statement = $db->prepare( "SELECT * FROM books");
$statement->execute();

// gets rows one at a time
while ($row = $statement->fetch())  {
print_r($row);
// or do something more meaningful with each returned row
}
$statement = null;
```

In this code, we “prepare” the SQL code and then “execute” it. Next, we cycle through the result with the while code and, finally, we release the result object by assigning null to it. This may not look all that powerful in this simple example, but there are other features that can be used with prepared statements. Now, consider this code :

```
$statement = $db->prepare("INSERT INTO books (authorid, title, ISBN, pub_year)"
 . "VALUES (:authorid, :title, :ISBN, :pub_year)");

$statement->execute(array(
'authorid' => 4,
'title' => "Foundation",
'ISBN' => "0-553-80371-9",
'pub_year' => 1951)
);
```

Here, we prepare the SQL statement with four named placeholders: authorid, title, ISBN, and pub_year. These happen to be the same names as the columns in the database. This is done only for clarity; the placeholder names can be anything that is meaningful to us. In the execute call, we replace these placeholders with the actual data that we want to use in this particular query. One of the advantages of prepared statements is that we can execute the same SQL command and pass in

different values through the array each time. We can also do this type of statement preparation with positional placeholders (not actually naming them), signified by a ?, which is the positional item to be replaced. Look at the following variation of the previous code:

```
$statement = $db->prepare("INSERT INTO books (authorid, title, ISBN, pub_year)"
. "VALUES (?,?,?,?)");
$statement->execute(array(4, "Foundation", "0-553-80371-9", 1951));
```

This accomplishes the same thing but with less code, as the value area of the SQL statement does not name the elements to be replaced, and therefore the array in the execute statement only needs to send in the raw data and no names. We just have to be sure about the position of the data that you are sending into the prepared statement.

❖ Program in PHP to access Server Information :

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Server Information</title>
</head>
<body>
<h1>Server Information</h1>
<?php
// Function to get server information
function getServerInfo($info) {
    $result = 'Not available';
    if (isset($_SERVER[$info])) {
        $result = $_SERVER[$info];
    }
    return $result;
}
?>
<table border="1">
    <tr>
        <td>Server Software</td>
        <td><?php echo getServerInfo('SERVER_SOFTWARE'); ?></td>
    </tr>
    <tr>
        <td>Server Name</td>
        <td><?php echo getServerInfo('SERVER_NAME'); ?></td>
    </tr>
    <tr>
        <td>Server IP Address</td>
```

```
            <td><?php echo getServerInfo('SERVER_ADDR'); ?></td>
        </tr>
        <tr>
            <td>Server Port</td>
            <td><?php echo getServerInfo('SERVER_PORT'); ?></td>
        </tr>
        <tr>
            <td>Document Root</td>
            <td><?php echo getServerInfo('DOCUMENT_ROOT'); ?></td>
        </tr>
        <tr>
            <td>PHP Version</td>
            <td><?php echo phpversion(); ?></td>
        </tr>
    </table>
    </body>
    </html>
```

Save this code in a file with a ".php" extension, for example, "server_info.php". When we access this file through a web browser, it will display a table containing various server information.
