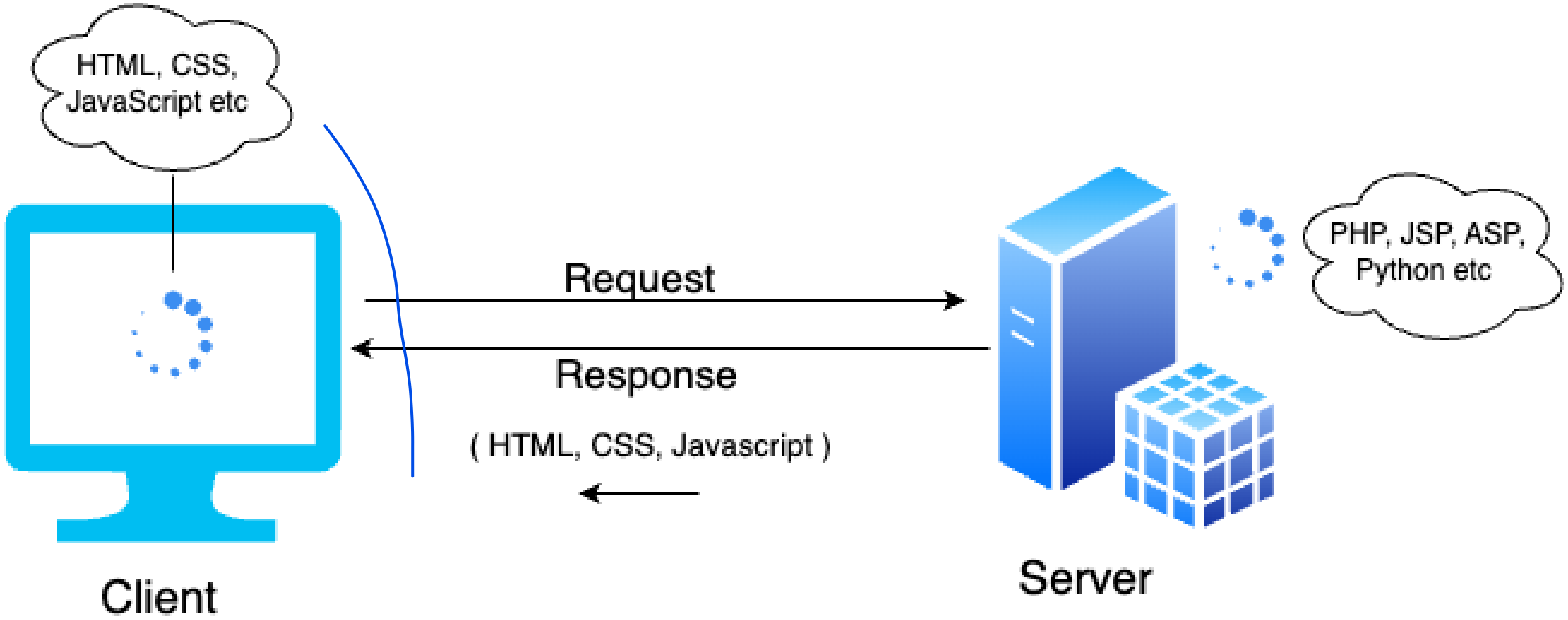
Full Stack Web Development

by Dr. Piyush Bagla

PHP

- PHP stands for **P**HP **H**ypertext **P**re-processor.
- It was originally created by Danish-Canadian programmer **Rasmus Lerdorf** in 1993 and released in 1995.
- PHP is a server-side scripting language.
- PHP is used to create dynamic and interactive websites.
- Open source: PHP is free to download and use.
- Platform independent: PHP code can run on any platform.
- Faster: PHP scripts are usually faster than other scripting languages.
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP supports a wide range of databases (MySQL, Oracle, MongoDB etc).

Client-Side vs Server-Side Scripting



What Can PHP Do?

- PHP can generate dynamic page content
- PHP can create, open, read, write, delete, and close files on the server
- PHP can collect form data
- PHP can send and receive cookies
- PHP can add, delete, modify data in your database
- PHP can be used to control user-access
- PHP can encrypt data

PHP with HTML

- Write HTML inside PHP.
- Write PHP inside HTML tags
- Add some style with PHP
- Use JS with PHP

PHP echo and print Statements

- echo does not have a return value. It simply outputs the specified content to the screen and does not return anything.
- print has a return value of 1, which means it can be used within expressions or assigned to variables.
- print can only accept a single parameter, attempting to use multiple parameters will result in a parse error.
  - **echo "Hello", "world";** // correct
  - print "Hello", "World"; // Error
- echo is generally considered to be marginally faster than print

Display Errors in PHP

- Why do we need to display errors?
- Check the **php.ini** file location.
- Change **display error** property.
- Test errors.

PHP supports the following data types:

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- Array
- Object
- NULL
- Resource

PHP string

In **PHP**, double quotes (") and single quotes (') handle strings differently, especially when it comes to **variable parsing** and **special characters** like newline (\n), tab (\t), etc.

✅ **Key Differences:**

1. Variable Parsing

- **Double quotes**: Variables **inside** the string are parsed.
- **Single quotes**: Variables are **not parsed**.

2. Special Characters

- **Double quotes**: Recognizes escape sequences like \n, \t, \\, etc.
- **Single quotes**: Treats most escape sequences **literally**, except for \\ and \'.

PHP string

✅ **Single Quotes in PHP:**

Single-quoted strings treat everything **literally**, **except**:

- \\ → interpreted as a single backslash (\)
- \' → interpreted as a single apostrophe (‘)

- ❌ **All other escape sequences are not processed:**

PHP Array

1. **Indexed Array** (like a list)

- $colors = array("red", "green", "blue");

// or

- $colors = ["red", "green", "blue"];

echo $colors[0]; // Output: red

PHP Array

2. **Associative Array** (like a dictionary / key-value pairs)

```
$person = array(
    "name" => "Piyush",
    "age" => 34,
    "city" => "Dehradun"
);
echo $person["name"]; // Output: Piyush
```

PHP Array

3. **Multidimensional Array** (array of arrays)

```
$students = [
   ["Swastik", 7, "A"],
   ["Ravi", 8, "B"],
   ["Neha", 7, "A"]
];
echo $students[0][0]; // Output: Swastik
```

Most Commonly Used PHP Array Functions

Function	Purpose	Example
`count()`	Count elements in an array	`count($arr)`
`array_push()`	Add element(s) to the end	`array_push($arr, "value")`
`array_pop()`	Remove the last element	`array_pop($arr)`
`in_array()`	Check if a value exists	`in_array("apple", $arr)`
`array_merge()`	Merge two or more arrays	`array_merge($arr1, $arr2)`
`array_keys()`	Get all keys from an array	`array_keys($assocArray)`
`array_values()`	Get all values from an array	`array_values($assocArray)`
`sort()`	Sort an indexed array (ascending order)	`sort($arr)`
`asort()`	Sort associative array by values (ascending)	`asort($assocArray)`

PHP Object

```php
class Car
{
	public $color;
	public $model;
	public function __construct($color, $model)
		{
			$this->color = $color;
			$this->model = $model;
		}
	 public function message()
		{
			 return "My car is a " . $this->color . " " . $this->model . "!";
		}
 }
$myCar = new Car("red", "Volvo");
var_dump($myCar);
```

Decision, Looping and Function

- Decision
  - if-else
  - switch
- Loops
  - for
  - while
  - do-while
  - foreach
- Functions
  - In PHP, a function can be defined before or after it is called.

Form Processing

Form processing is the **handling of user input** submitted via an HTML <form> using PHP. It usually involves:

1. Displaying a form
2. Submitting it via GET or POST
3. Processing data in a PHP script

Form Processing

HTML form

```
<form action="process.php" method="post">
   Name: <input type="text" name="username"><br>
   Email: <input type="email" name="email"><br>
   <input type="submit" value="Submit">
</form>
```

Form Processing

process.php

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = $_POST['username'];
    $email = $_POST['email'];

    echo "Welcome, $name!<br>";
    echo "Your email is: $email";
}
?>
```

Form Processing

Important: Validating and Sanitizing Input

✅ Sanitize input:

```
$name = htmlspecialchars(trim($_POST['username']));
$email = filter_var($_POST['email'], FILTER_SANITIZE_EMAIL);
```

Form Processing

Important: Validating and Sanitizing Input

✅ Validate email:

```
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
   echo "Invalid email!";
}
```

filter_var() with FILTER_SANITIZE_EMAIL removes illegal characters (like spaces, quotes, etc.) from the email input.

Form Processing

✅ Validate email

```
$email = "  someone @gmail.com  ";
$sanitized = filter_var($email, FILTER_SANITIZE_EMAIL); // becomes someone@gmail.com

if (filter_var($sanitized, FILTER_VALIDATE_EMAIL)) {
    echo "Valid email after sanitization!";
} else {
    echo "Still invalid.";
}
```

Form Processing

✅ Validate email

```
$email = "<script>alert('hack')</script>@gmail.com";
$sanitized = filter_var($email, FILTER_SANITIZE_EMAIL);
echo $sanitized;
```

Output after sanitization:

alerthack@gmail.com

Form Processing

Super global Variables

_GET

_POST

_REQUEST

_COOKIE

_ SESSION and more

COOKIES in PHP

- What are cookies?
- How are Cookies Sent?
- Syntax and parameters.
- Set data in cookies.
- Get data from cookies.
- Modify a cookie.
- Delete a cookie.

Cookies

What are cookies?

Cookies are text files that contain small amounts of data, which are sent from a server to a user's web browser and then returned to the server each time the browser requests a page from the server. They can store various types of information, such as user preferences, login status, and tracking identifiers.

Cookies

How are Cookies Sent?

1. **Setting Cookies**: When a server wants to store a cookie on the user's device, it sends an HTTP header with the ***Set-Cookie*** directive. This header includes the cookie name, value, expiration date, path, domain, and security attributes.

2. **Sending Cookies Back**: When the user makes subsequent requests to the same server, the browser automatically includes the relevant cookies in the HTTP request headers. This allows the server to recognize the user and retrieve any stored information.

Cookies

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

name – Name of the cookie (compulsory)

Is the value parameter in setcookie() optional?

```
setcookie("user");          // Sets cookie with empty value ("")
setcookie("user", "");      // Same as above, more explicit
setcookie("user", "John");  // Sets value to "John"
```

Cookies

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

path

If the **cookie stores the user's location** and you set it with the path /, then:

- ✅ **Any page** on your website — like Home, About Us, Contact Us — can **read that location cookie**.

Cookies

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

Httponly

It protects your cookie (especially things like session IDs or login tokens) from:

- Cookie not accessible via JavaScript
- Cross-Site Scripting (XSS) attacks
- Malicious scripts that try to steal cookies using JavaScript

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

```
$name = "user";
$value = "John";
$expire = time() + (60 * 60 * 24 * 30); // 30 days from now
$path = "/"; // available in the entire domain
$domain = ”.example.com"; // available for the domain and all subdomains
$secure = true; // available only over HTTPS
$httponly = true; // accessible only through HTTP (not JavaScript)

// Set the cookie
setcookie($name, $value, $expire, $path, $domain, $secure, $httponly);
```

Cookies
Default values

Parameter	Type	Default Value	Description
`$name`	`string`	**(Required)**	The name of the cookie.
`$value`	`string`	`""` (empty string)	The value to store in the cookie.
`$expires_or_options`	`int` / `array`	`0` (session cookie)	Expiry time as Unix timestamp (0 = until browser closes).
`$path`	`string`	`""` (current script's directory)	Path where the cookie is available.
`$domain`	`string`	`""` (current host)	Domain that can access the cookie.
`$secure`	`bool`	`false`	If `true`, cookie is sent only over HTTPS.
`$httponly`	`bool`	`false`	If `true`, cookie can't be accessed via JavaScript.

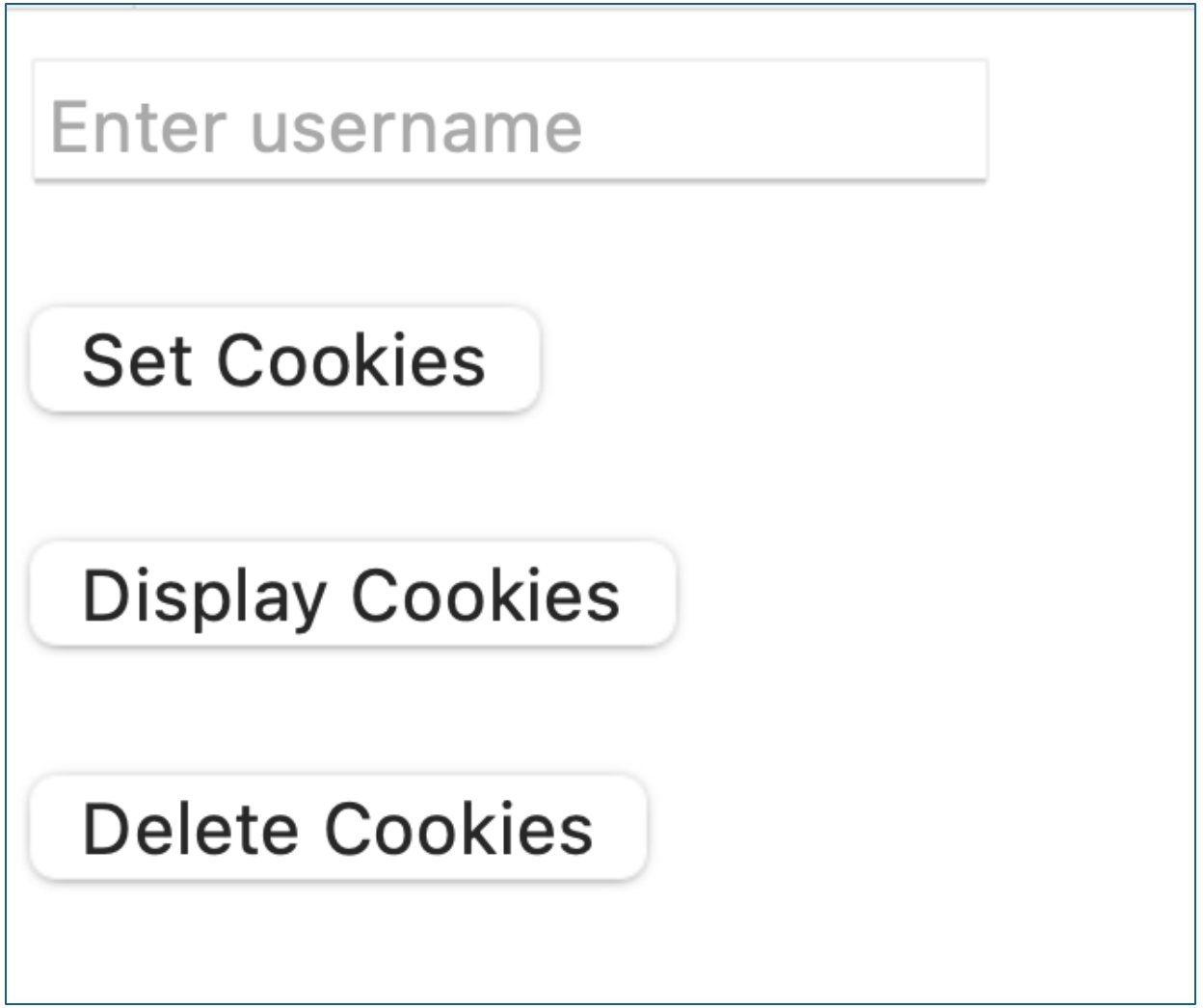
setcookie(*name, value, expire, path, domain, secure, httponly*);

- If the path parameter is set to "/account", meaning the cookie will only be available within the "/account" directory and its subdirectories.

  For example, it will be accessible at "example.com/account", "example.com/account/subdirectory", etc., but not at "example.com" or other directories.

- The domain parameter is set to "example.com", meaning the cookie will only be available for the domain "example.com" and not for any subdomains or other domains.

Homework


Enter username
Set Cookies
Display Cookies
Delete Cookies

Sessions

Client

1.login

3. cookie
session ID

4.request cookie
session ID

6.resource

Server

2.create
session

session ID

5.verify

Summary of the Flow in Terms of Session and Cookie:

1.User visits the website.

2.Server creates a session and generates a unique session ID.

3.The session ID is stored in a cookie and sent to the user's browser.

4.On subsequent requests, the browser sends the session ID back in the cookie.

5.The server retrieves the session data using the session ID and processes the request.

6.Session and cookie data may be updated based on user interactions (e.g., login, preferences).

7.The session can expire, and the cookie may be removed or expire naturally.

Cookies are essential for session management as they allow the server to recognize users across different requests, ensuring a smooth user experience (e.g., staying logged in, preserving preferences).

Many types of data that can be stored in sessions can also be stored in cookies.

Both cookies and sessions have their own advantages and trade-offs. Cookies are useful for storing small amounts of non-sensitive data on the client side, while sessions offer better security and flexibility for storing user-specific data on the server side. The choice between cookies and sessions depends on your application's specific requirements and the type of data you want to store.

Cookies vs Session

Aspect	Cookies	Sessions
Storage Location	Stored on the client side (in the web browser).	Stored on the server side.
Data Access	Read and written by the client and server.	Managed primarily by the server; accessed via session ID.
Size Limit	Limited to around 4 KB per cookie; limit on number of cookies per domain.	No strict size limits; only limited by server resources.
Security	Vulnerable to XSS attacks; can be tampered with.	More secure as data is stored on the server.
Persistence	Can be persistent or session-based; set with an expiration date.	Typically expires after a set time or when the session ends.
Use Case	Used for storing small, non-sensitive data, such as user preferences, shopping cart, or tracking data.	Used for storing sensitive data like user authentication, session data, or complex data structures.
Cross-Domain Sharing	Cookies can be shared across subdomains, depending on their settings.	Sessions are typically specific to a single domain.
Accessibility	Can be accessed via client-side scripts (JavaScript), increasing risk of exposure.	Not directly accessible to client-side scripts, providing better security.
Session ID	May hold session ID for server-side session tracking.	Identified by a session ID, typically stored in a cookie.

Session

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

1. Start a session
2. Set session variables
3. Get the session variable values
4. Modify the session variables
5. Destroy session

   Remove all session variables first; then, only the session will be destroyed

login.php

```php
<?php
session_start();

$valid_username = "admin";
$valid_password = "password123";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
$username = $_POST['username'];
$password = $_POST['password'];

if ($u === $valid_username && $p === $valid_password) {
$_SESSION['username'] = $u;
header("Location: dashboard.php");
exit();
} else {
$error = "Invalid username or password!";
}
}
?>
```

continue

```
<!DOCTYPE html>
<html>
<head><title>Login</title></head>
<body>
<h2>Login Page</h2>
<form method="POST” action=“login.php”>
Username: <input type="text" name="username" required><br><br>
Password: <input type="password" name="password" required><br><br>
<input type="submit" value="Login">
</form>
</body>
</html>
```

dashboard.php

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

<!DOCTYPE html>
<html>
<head><title>Dashboard</title></head>
<body>
<h2>Welcome, <?php echo $_SESSION['username']; ?>!</h2>
<p>This is your dashboard.</p>
<a href="logout.php">Logout</a>
</body>
</html>
```

logout.php

```php
<?php
session_start();
session_unset(); // Remove all session variables
session_destroy(); // Destroy the session
header("Location: login.php");
exit();
```

Database & PHP

Connect PHP with MySql

- How many ways do we have to connect?
- Make Variables for connection.
- Connect with MySQLi or PDO class.
  - 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.
- Get the Table List from the database.

MySQLi or PDO?

Both MySQLi and PDO have their advantages:

- PDO will work on 12 different database systems, whereas MySQLi will only work with MySQL databases.

  - So, if you have to switch your project to use another database, PDO makes the process easy. You only have to change the connection string and a few queries. With MySQLi, you will need to rewrite the entire code - queries included.

- Both are object-oriented, but MySQLi also offers a procedural API.

- Both support Prepared Statements.
  - Prepared Statements protect from SQL injection, and are very important for web application security.

Database Connectivity using MySQLi

$servername = "localhost";
- This line specifies the server name or IP address where the MySQL database server is hosted.

$username = "username";
- This line specifies the username needed to authenticate with the MySQL server.

$password = "password";
- This line specifies the password for the specified username.

// Create connection
$conn = new mysqli($servername, $username, $password);
- This line creates a new instance of the MySQLi class, which establishes a connection to the MySQL database server using the server name, username, and password specified earlier.

// Check connection
if ($conn->connect_error) {
die("Connection failed: " . $conn->connect_error); }
- This block checks whether the connection to the MySQL server was successful. If there is a connection error (`connect_error` property is set), the script terminates (`die`) and outputs an error message indicating the reason for the connection failure.

echo "Connected successfully";
$conn->close(); // close the connection

Database Connectivity using PDO

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

try {
$conn = new PDO("mysql:host=$servername;dbname=gehu", $username, $password);
/*The first argument specifies the DSN (Data Source Name), which includes the database type (mysql), the hostname ($servername), and the database name (gehu). The second and third arguments specify the MySQL username and password, respectively.*/

// set the PDO error mode to exception
$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
•/* Setting PDO error mode: This line sets the PDO error mode to PDO::ERRMODE_EXCEPTION for the connection $conn.
    • This means that if an error occurs during the database connection or query execution, PDO will throw an exception (PDOException) instead of returning an error code.
    • This makes it easier to handle database errors using exception handling.
 */
echo "Connected successfully";
} catch(PDOException $e) {
echo "Connection failed: " . $e->getMessage();
?/Error message: The script prints an error message "Connection failed: " followed by the exception message ($e->getMessage()).
}
$conn = NULL;
?>
```

SQL Injection

1) Connection

```
$conn = new mysqli(Server Name, User Name, Password, Database Name)
```

2) Run SQL Query

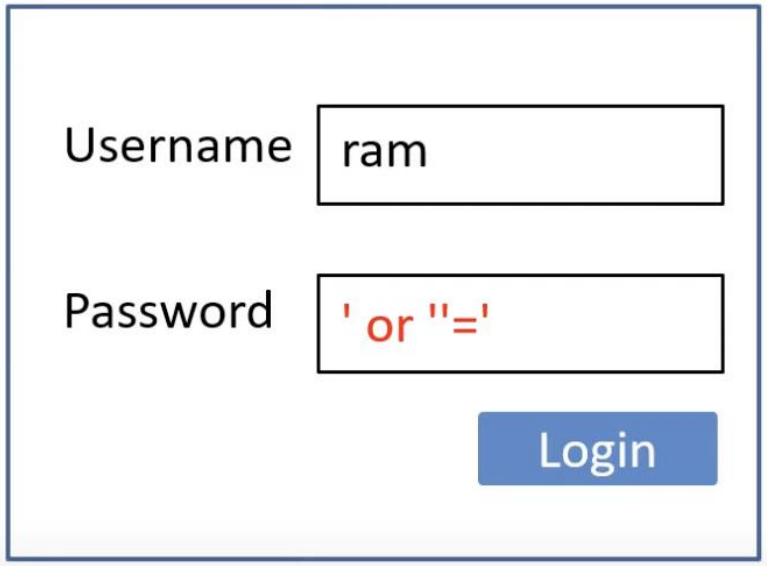
```
$conn->query(SQL Query)
```

SQL Injection

3) Close Connection

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

SQL Injection




```
$username = $_POST["username"];
$password = $_POST["password"];
```

```
$sql = "SELECT * FROM users WHERE user = '$username' AND pass = '$password'";
```

```
SELECT * FROM users WHERE user = 'ram' AND pass = ' ' or ' '=' '
```

Protection from SQL Injection (Method 1)

real_escape_string()

```
$username = $conn->real_escape_string ($_POST["username"]);
$password = $conn->real_escape_string ($_POST["password"]);
```

- **Original Username:** admin' OR '1'='1
- **Escaped Username:** admin\' OR \'1\'=\'1

Character	Escaped As
'	\'
"	\"
\	\\
\0	\\0
\n	\\n
\r	\\r
\x1a	\\Z

Protection from SQL Injection (Method 2)

Prepare Statement ()

1) Connection

```
$conn = new mysqli(Server Name, User Name, Password, Database Name)
```

2) Run SQL Query

```
$sql = $conn->prepare("SELECT * FROM users WHERE user = ? AND pass = ?");
$sql->bind_param("ss", $username, $password);
$sql->execute();
```

3) Close Connection

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

Protection from SQL Injection (Method 2)

Data types for bind_param parameters

- i - Integer
- d - Double
- s - String
- b - Blob

PHP MySQL Prepared Statements

Prepared statements are very useful against SQL injections.

Prepared Statements and Bound Parameters

A prepared statement is a feature used to execute the same (or similar) SQL statements repeatedly with high efficiency.

Prepared statements basically work like this:

1.Prepare: An SQL statement template is created and sent to the database. Certain values are left unspecified, called parameters (labeled "?"). Example: INSERT INTO MyGuests VALUES(?, ?, ?)

2.The database parses, compiles, and performs query optimization on the SQL statement template, and stores the result without executing it

3.Execute: At a later time, the application binds the values to the parameters, and the database executes the statement. The application may execute the statement as many times as it wants with different values

PHP MySQL Prepared Statements

Compared to executing SQL statements directly, prepared statements have three main advantages:

- Prepared statements reduce parsing time as the preparation of the query is done only once (although the statement is executed multiple times)
- Bound parameters minimize bandwidth to the server as you need to send only the parameters each time, and not the whole query.
- Prepared statements are one of the most effective ways to prevent SQL injection attacks, which occur when an attacker attempts to manipulate the query by inserting malicious SQL code through user input fields.

Prepared Statements

```
// prepare and bind
$stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email) VALUES
(?, ?, ?)");
$stmt->bind_param("sss", $firstname, $lastname, $email);

// set parameters and execute
$firstname = ”Rahul";
$lastname = ”Sharma";
$email = ”rahul@example.com";
$stmt->execute();

$firstname = ”Danish";
$lastname = ”Khan";
$email = ”khan@example.com";
$stmt->execute();

echo "New records created successfully";

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