How Web server works?

A web server is a system that delivers web content to users over the internet. When a user requests a web page by entering a URL in a browser, the request is sent to the web server. The server processes the request, retrieves the requested content (like an HTML page or image), and sends it back to the user's browser, which then displays the content.

Difference between client-side scripting and server-side scripting :

Client-side scripting	**Server-side scripting**
Source code is visible to the user.	Source code is not visible to the user because its output of server-sideside is an HTML page.
Its main function is to provide the requested output to the end user.	Its primary function is to manipulate and provide access to the respective database as per the request.
It usually depends on the browser and its version.	In this any server-side technology can be used and it does not depend on the client.
It runs on the user's computer.	It runs on the webserver.
There are many advantages linked with this like faster. response times, a more interactive application.	The primary advantage is its ability to highly customize, response requirements, access rights based on user.
It does not provide security for data.	It provides more security for data.
It is a technique used in web development in which scripts run on the client's browser.	It is a technique that uses scripts on the webserver to produce a response that is customized for each client's request.
HTML, CSS, and javascript are used.	PHP, Python, Java, Ruby are used.

Client-side scripting	Server-side scripting
No need of interaction with the server.	It is all about interacting with the servers.
It reduces load on processing unit of the server.	It surge the processing load on the server.

Difference between Static and Dynamic Web Pages:

SL.NO	Static Web Page	Dynamic Web Page
1.	In static web pages, Pages will remain same until someone changes it manually.	In dynamic web pages, Content of pages are different for different visitors.
2.	Static Web Pages are simple in terms of complexity.	Dynamic web pages are complicated.
3.	In static web pages, Information are change rarely.	In dynamic web page, Information are change frequently.
4.	Static Web Page takes less time for loading than dynamic web page.	Dynamic web page takes more time for loading.
5.	In Static Web Pages, database is not used.	In dynamic web pages, database is used.
6.	Static web pages are written in languages such as: HTML, JavaScript, CSS, etc.	Dynamic web pages are written in languages such as: CGI, AJAX, ASP, ASP.NET, etc.
7.	Static web pages does not contain any application program .	Dynamic web pages contains application program for different services.
8.	Static web pages require less work and cost in designing them.	Dynamic web pages require comparatively more work and cost in designing them.

Write a code for PHP CRUD Operation?

```
<?php
// Database connection
$conn = new mysqli("localhost", "root", "", "crud_example");
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Create operation
if (isset($_POST['create'])) {
    $name = $_POST['name'];
    $email = $_POST['email'];
    $conn->query("INSERT INTO users (name, email) VALUES ('$name', '$email')");
}

// Read Operation
$result = $conn->query("SELECT * FROM users");
while ($row = $result->fetch_assoc()) {
    echo "ID: {$row['id']} - Name: {$row['name']} - Email: {$row['email']}<br>";
}

// Update Operation
if (isset($_POST['update'])) {
    $id = $_POST['id'];
    $name = $_POST['name'];
    $email = $_POST['email'];
    $conn->query("UPDATE users SET name='$name', email='$email' WHERE id=$id");
}

// Delete
```

```
if (isset($_POST['delete'])) {
    $id = $_POST['id'];
    $conn->query("DELETE FROM users WHERE id=$id");
}

$conn->close();
?>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>PHP CRUD Example</title>
</head>
<body>
    <h2>Create User</h2>
    <form method="POST">
        Name: <input type="text" name="name" required>
        Email: <input type="email" name="email" required>
        <button type="submit" name="create">Create</button>
    </form>

    <h2>Update User</h2>
    <form method="POST">
        ID: <input type="number" name="id" required>
        Name: <input type="text" name="name" required>
        Email: <input type="email" name="email" required>
        <button type="submit" name="update">Update</button>
    </form>

    <h2>Delete User</h2>
```

```
    <form method="POST">
        ID: <input type="number" name="id" required>
        <button type="submit" name="delete">Delete</button>
    </form>
</body>
</html>
```

Design a web page and write a program in php to connect with MYSQL database and describe the parameter used in the function.

1. Web Page Design (HTML)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>PHP MySQL Connection Example</title>
</head>
<body>

<h2>Test Database Connection</h2>
<form action="connect.php" method="POST">
    <button type="submit">Connect to Database</button>
</form>

</body>
</html>
```

?**2. PHP Script to Connect to MySQL Database**

```
<?php
// Database connection parameters
$servername = "localhost";
$username = "root";
$password = "";
```

```
$dbname = "crud_example";   // The name of the database you want to connect to


// Creating connection
$conn = new mysqli($servername, $username, $password, $dbname);


// Checking connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
} else {
   echo "Connected successfully to the database '$dbname'";
}


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

3. Explanation of Parameters

Database Connection Parameters:

1. **$servername = "localhost"**:
    - This specifies the server where the MySQL database is hosted. If you are running MySQL on your local machine, this will be localhost.
2. **$username = "root"**:
    - The username to connect to the MySQL database. In local development environments like XAMPP or WAMP, the default username is usually root.
3. **$password = ""**:
    - The password associated with the MySQL user. The default password for local servers is often left empty.
4. **$dbname = "crud_example"**:
    - The name of the MySQL database you want to connect to. This should be an existing database on your MySQL server.

Database Connection Function:

1. **$conn = new mysqli($servername, $username, $password, $dbname);**:

- This line creates a new instance of the MySQLi class, attempting to connect to the database using the provided server name, username, password, and database name.

2. **if ($conn->connect_error) { die("Connection failed: " . $conn->connect_error); }**:
   - This conditional statement checks if the connection failed. If it did, the script stops execution and displays an error message.
3. **echo "Connected successfully to the database '$dbname'";**:
   - If the connection is successful, this message will be displayed.
4. **$conn->close();**:
   - After completing the connection check, this line closes the connection to the MySQL database.

What are XML syntax rule? Explain the different templates used in xml with an example.

XML Syntax Rules

XML (Extensible Markup Language) is used to store and transport data. It is both human-readable and machine-readable, and its syntax rules ensure that the data is well-formed and easily parsed. Here are the fundamental syntax rules for XML:

1. **An XML document must have one root element that contains all other elements.**
2. **XML tags must be written with consistent case. <note> and <Note> would be considered different tags.**
3. **Every opening tag must have a corresponding closing tag.**

   <message>Hello, World!</message>

4. **XML elements must be nested correctly. For instance, you cannot close an inner element before its outer element.**

   <note>

   <to>Tove</to>

   <from>Jani</from>

   </note>

5. **In XML, attribute values must always be enclosed in quotes, either single (') or double (").**

   <note date="2024-08-17" author='Jani'>

   <to>Tove</to>

   </note>

6. **XML preserves whitespace, including spaces, tabs, and newlines. This is important when whitespace matters in the data.**

```
<message>
   This is a message with
   significant whitespace.
</message>
```

7. Comments can be added to an XML document and are ignored by the parser.

```
<!-- This is a comment -->
<note>
   <to>Tove</to>
</note>
```

Templates in XML

XML itself doesn't have "templates" like some other templating languages (e.g., HTML with PHP, or templating engines in web frameworks). However, XML is often used in conjunction with XSLT (Extensible Stylesheet Language Transformations), which can be considered a form of templating to transform XML data into different formats, such as HTML, plain text, or another XML format.

1. XSLT Templates

XSLT is used to transform XML documents into other formats, using templates to match specific XML elements and apply styles or generate new elements.

Syntax:

```
<xsl:template match="element">
   <!-- Transformation rules -->
</xsl:template>
```

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

   <!-- Template for the root element -->
   <xsl:template match="/note">
      <html>
         <body>
            <h2><xsl:value-of select="heading"/></h2>
            <p><xsl:value-of select="body"/></p>
```

```
    </body>
  </html>
```

2. XML Schema (XSD) Templates

XSD (XML Schema Definition) defines the structure and rules for an XML document. An XSD file serves as a template that specifies what elements, attributes, and data types are allowed in an XML document.

Syntax:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="elementName" type="xs:string"/>
</xs:schema>
```

Example:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="heading" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

What is an Exception?

An exception is an event that disrupts the normal flow of a program's execution. Exceptions are used to signal that an error or an unexpected situation has occurred, allowing the program to handle these issues in a controlled manner rather than crashing or producing incorrect results.

Exception Handling in PHP

PHP provides a mechanism for handling exceptions using a set of keywords and constructs.

```
<?php
class CustomException extends Exception {
   // Custom exception class
}

function divide($numerator, $denominator) {
   if ($denominator == 0) {
      throw new CustomException("Division by zero is not allowed.");
   }
   return $numerator / $denominator;
}

try {
   $result = divide(10, 0);
   echo "Result: " . $result;
} catch (CustomException $e) {
   echo "Caught custom exception: " . $e->getMessage();
} catch (Exception $e) {
   echo "Caught general exception: " . $e->getMessage();
} finally {
   echo "This will always execute.";
}
?>
```

Creating and Manipulating Strings in PHP

Strings in PHP are sequences of characters used to represent text. PHP provides robust functions and operators to create, manipulate, and format strings efficiently. Key operations include string creation, concatenation, searching, replacing, and formatting.

1. String Creation

- **Single Quotes (')**: Creates a string literally. Variables inside single quotes are not parsed.
- **Double Quotes (")**: Allows variable interpolation and supports escape sequences like newline (\n).

2. String Concatenation

- **Dot Operator (.)**: Joins two or more strings.

3. Common String Functions

- **strlen()**: Returns the length of a string.
- **strtoupper()**: Converts a string to uppercase.
- **strtolower()**: Converts a string to lowercase.
- **substr()**: Extracts a substring from a string.
- **strpos()**: Finds the position of the first occurrence of a substring.
- **str_replace()**: Replaces all occurrences of a search string with a replacement string.
- **trim()**: Removes whitespace or other characters from the beginning and end of a string.
- **explode()**: Splits a string by a delimiter and returns an array.
- **implode()**: Joins array elements into a single string.

4. String Formatting

- **sprintf()**: Formats a string and returns it.
- **printf()**: Outputs a formatted string directly.

5. String Conversion

- **Type-Casting**: Converts other data types to strings.
- **strval()**: Converts a value to a string.

Is PHP Object oriented programming? How to create and add properties to a class?

Yes, PHP supports Object-Oriented Programming (OOP). OOP in PHP allows you to design and structure your code using objects and classes, making it easier to manage and maintain complex applications.

Object-Oriented Programming (OOP) is a programming paradigm that uses objects and classes to organize and manage code. In PHP, OOP enables you to model real-world entities, encapsulate data and behavior, and promote code reuse.

Creating a Class and Adding Properties

1. **Defining a Class:**
    - A class is a blueprint for creating objects. It can contain properties (variables) and methods (functions).
2. **Adding Properties:**
    - Properties are variables defined within a class. They represent the state or attributes of an object.

```
<?php
// Define a class named 'Person'
class Person {
    // Properties (attributes) of the class
    public $firstName;
    public $lastName;
    public $age;

    // Method to set the properties
    public function setDetails($firstName, $lastName, $age) {
        $this->firstName = $firstName;
        $this->lastName = $lastName;
```

```
        $this->age = $age;
    }

    // Method to display the properties
    public function displayDetails() {
        echo "Name: " . $this->firstName . " " . $this->lastName . "<br>";
        echo "Age: " . $this->age . "<br>";
    }
}

// Create an object of the class
$person1 = new Person();

// Set properties using the setDetails method
$person1->setDetails("John", "Doe", 30);

// Display properties using the displayDetails method
$person1->displayDetails();
?>
```

What is regular expression? Explain with example.

A **regular expression (regex)** is a sequence of characters that defines a search pattern. It's used for finding or matching text within strings, such as checking if a string contains a certain pattern or replacing parts of a string.

Example in Code:

```
const email = "example@gmail.com";
const regex = /^[a-zA-Z0-9._%+-]+@gmail\.com$/;
```

```
if (regex.test(email)) {
   console.log("Valid Gmail address!");
} else {
   console.log("Not a Gmail address.");
}
```

Explain different PHP arrays used in PHP with example.

In PHP, arrays are an essential data structure that allow you to store multiple values in a single variable. PHP arrays can hold different types of values, and they come in three main types: Indexed Arrays, Associative Arrays, and Multidimensional Arrays. Each of these types serves a different purpose.

1. Indexed Arrays

Indexed arrays are arrays where the keys (indexes) are automatically assigned numeric values, starting from 0.

Example:

```
$cars = array("Toyota", "Honda", "Ford");
echo $cars[0]; // Output: Toyota
echo $cars[1]; // Output: Honda
echo $cars[2]; // Output: Ford
```

2. Associative Arrays

Associative arrays use named keys that you assign to them manually. This allows you to associate values with a particular key, making it more descriptive.

Example:

```
$age = array("John" => 25, "Jane" => 30, "Doe" => 22);
echo $age["John"]; // Output: 25
echo $age["Jane"]; // Output: 30
echo $age["Doe"]; // Output: 22
```

3. Multidimensional Arrays

Multidimensional arrays are arrays that contain one or more arrays. They allow you to store data in a tabular format.

Example:

```
$students = array(
    array("John", 25, "A"),
    array("Jane", 30, "B"),
    array("Doe", 22, "A+")
);
echo $students[0][0]; // Output: John
echo $students[1][1]; // Output: 30
echo $students[2][2]; // Output: A+
```

What is inheritance? How can you implement inheritance in PHP? Explain with suitable example.

Inheritance is a fundamental concept in object-oriented programming (OOP) that allows a class (called a child or subclass) to inherit properties and methods from another class (called a parent or superclass). This promotes code reusability and establishes a natural hierarchy between classes.

Implementing Inheritance in PHP

In PHP, inheritance is implemented using the extends keyword. A child class can extend a parent class, inheriting its properties and methods.

Example:

Let's consider a scenario where we have a general Animal class, and then we create a specific Dog class that inherits from it.

```
class Animal {
    public $name;
```

```
    public $type;
    public function __construct($name, $type) {
        $this->name = $name;
        $this->type = $type;
    }
    public function makeSound() {
        return "Some generic animal sound";
    }
}
```

Step 2: Child Class

```
class Dog extends Animal {
    public function makeSound() {
        return "Bark!";
    }
    public function fetch() {
        return "$this->name is fetching!";
    }
}
```

Step 3: Using the Classes

```
$myDog = new Dog("Buddy", "Dog");

echo $myDog->name;         // Output: Buddy
echo $myDog->type;         // Output: Dog
echo $myDog->makeSound();  // Output: Bark!
echo $myDog->fetch();      // Output: Buddy is fetching!
```

What are the various access modifier used in php?

In PHP, access modifiers are keywords used to set the visibility of class properties and methods. They control how and where the properties and methods can be accessed, ensuring encapsulation and security in object-oriented programming. PHP supports three primary access modifiers.

1. Public

- **Visibility**: Public properties and methods can be accessed from anywhere—inside the class, outside the class, and by child classes.
- **Use**: When you want the data or behavior to be accessible from any part of the code.

```
class MyClass {
    public $publicProperty = "I am public!";
    public function publicMethod() {
        return "This is a public method!";
    }
}
$obj = new MyClass();
echo $obj->publicProperty; // Output: I am public!
echo $obj->publicMethod(); // Output: This is a public method!
```

2. Protected

- **Visibility**: Protected properties and methods can be accessed within the class itself and by classes derived from that class (child classes). They cannot be accessed from outside the class.
- **Use**: When you want to restrict access to the data or behavior to the class itself and its subclasses but not to the outside world.

```
class ParentClass {
    protected $protectedProperty = "I am protected!";
    protected function protectedMethod() {
        return "This is a protected method!";
    }
}
class ChildClass extends ParentClass {
    public function accessProtected() {
        return $this->protectedProperty . " and " . $this->protectedMethod();
    }
}
$obj = new ChildClass();
echo $obj->accessProtected(); // Output: I am protected! and This is a protected method!
```

3. Private

- **Visibility**: Private properties and methods can only be accessed within the class in which they are declared. They cannot be accessed from child classes or from outside the class.
- **Use**: When you want to hide the data or behavior completely from other classes and restrict its access strictly to the class itself.

```
class MyClass {
    private $privateProperty = "I am private!";

    private function privateMethod() {
        return "This is a private method!";
    }
```

```
    public function accessPrivate() {
        return $this->privateProperty . " and " . $this->privateMethod();
    }
}

$obj = new MyClass();

echo $obj->accessPrivate(); // Output: I am private! and This is a private method!
```

Why we need responsive websites? Discuss various MVC Framework of PHP.

Responsive websites are crucial in today's digital environment because they provide an optimal viewing experience across a wide range of devices, from desktop computers to smartphones and tablets. Here's why responsive design is important:

1. **Mobile Usage**: With the rise in mobile device usage, it's essential for websites to work well on smaller screens. A responsive website adapts to the screen size, ensuring that users have a good experience whether they're on a phone, tablet, or desktop.
2. **Improved User Experience**: A responsive website ensures that users don't have to zoom in, scroll sideways, or struggle to read content. It adjusts elements like images, text, and navigation for easy use on any device, leading to better engagement and satisfaction.
3. **SEO Benefits**: Search engines like Google prioritize mobile-friendly websites in their search results. A responsive design helps improve your website's visibility and ranking in search engine results pages (SERPs).
4. **Cost-Efficiency**: Instead of creating and maintaining separate websites for desktop and mobile, responsive design allows you to manage a single website that works across all devices. This reduces development and maintenance costs.
5. **Future-Proofing**: A responsive website is better equipped to handle new devices with varying screen sizes and resolutions, making your site more adaptable to future technological changes.

MVC Frameworks in PHP

The **Model-View-Controller (MVC)** architecture is a design pattern that separates an application into three interconnected components, promoting organized and maintainable code. Here's a breakdown:

1. **Model**: Handles the data and business logic of the application. It interacts with the database and sends data to the controller.
2. **View**: Represents the user interface and displays data to the user. It takes care of how the data is presented, using the information provided by the controller.
3. **Controller**: Acts as an intermediary between the Model and View. It processes user input, interacts with the model, and decides which view to display.

Several PHP frameworks use the MVC architecture. Here are some of the popular ones:

1. Laravel

- **Description**: Laravel is one of the most popular PHP frameworks known for its elegant syntax, robust features, and extensive ecosystem.
- **Features**: Includes built-in authentication, routing, sessions, caching, and a powerful ORM (Eloquent) for database management.
- **Use Cases**: Suitable for both small projects and large-scale enterprise applications.

2. Symfony

- **Description**: Symfony is a high-performance PHP framework used for building complex web applications. It provides a set of reusable PHP components.
- **Features**: Includes a robust routing system, templating engine (Twig), and strong integration with other libraries.
- **Use Cases**: Ideal for large-scale enterprise-level applications and projects requiring high scalability.

3. CodeIgniter

- **Description**: CodeIgniter is a lightweight and simple-to-use PHP framework known for its speed and small footprint.
- **Features**: Provides essential tools for routing, form validation, and database management, with minimal configuration.
- **Use Cases**: Best suited for small to medium-sized projects where simplicity and speed are critical.

4. Yii

- **Description**: Yii is a high-performance PHP framework designed for developing modern web applications. It's known for being fast and efficient.
- **Features**: Includes a powerful caching system, active record for database interaction, and built-in security features.
- **Use Cases**: Ideal for projects that require high performance, such as e-commerce platforms, forums, and content management systems.

5. Zend Framework / Laminas

- **Description**: Zend Framework, now known as Laminas, is an enterprise-grade PHP framework focused on building robust, scalable applications.
- **Features**: Offers extensive components for form validation, input filtering, and an advanced MVC implementation.
- **Use Cases**: Suitable for large enterprise applications and projects requiring high levels of customization.

6. CakePHP

- **Description**: CakePHP is a rapid development framework that provides a structured and consistent way to build web applications.
- **Features**: Comes with a built-in ORM, form validation, authentication, and security features.
- **Use Cases**: Ideal for developers looking to build applications quickly with minimal configuration.

Define login and authentication. Write a Program in PHP to display prime Number between 1 - 100.

Login and Authentication

Login and authentication are key processes in securing access to applications and systems.

- **Login:** The act of a user providing credentials (usually a username and password) to gain access to a system. The login process typically involves submitting these credentials through a form.
- Authentication: The process of verifying that the provided credentials are correct. If the credentials match what is stored in the system (e.g., in a database), the user is granted access. Authentication ensures that only authorized users can access protected resources.

PHP Program to Display Prime Numbers Between 1 and 100

```
<?php
echo "Prime numbers between 1 and 100 are: ";
for ($num = 2; $num <= 100; $num++) {
   $isPrime = true;
   for ($i = 2; $i < $num; $i++) {
      if ($num % $i == 0) {
         $isPrime = false;
         break;
      }
   }
   if ($isPrime) {
      echo $num . " ";
   }
}
```

?>

AJAX

AJAX (Asynchronous JavaScript and XML) enables web pages to update content dynamically without reloading the entire page by sending and receiving data in the background. This technique uses the XMLHttpRequest object to communicate with the server asynchronously, improving the user experience by making interactions faster and smoother. For example, AJAX allows for live search suggestions and form submissions without refreshing the page, enhancing the overall efficiency and responsiveness of web applications.

OWL

OWL (Web Ontology Language) is a formal language used to create and manage ontologies on the web, providing a rich framework for defining complex relationships and data structures. It allows for detailed modeling of concepts and their interrelationships, enabling automated reasoning and inference to derive new insights. OWL is particularly useful for organizing and integrating information across different systems, such as in medical or scientific domains, where precise and interoperable data representation is crucial.

Cookies

Cookies are small pieces of data stored on the client's browser by a web server, used to remember information between visits. They can store user preferences, login details, or tracking information and have a set expiration time. Cookies are sent with every HTTP request to the server, which allows the server to maintain state or personalize content based on the stored data.

Sessions

sessions are a server-side storage mechanism that retains user-specific data during a single visit. Each session is identified by a unique session ID, which is typically stored in a cookie or URL. Sessions are used to keep track of user activities, such as login states or shopping cart contents, and are generally more secure as the sensitive data is kept on the server rather than the client. While cookies can persist beyond a single session, sessions typically expire when the user closes the browser or after a period of inactivity.

Here are four common MySQL functions with examples:

1. CONCAT()

Description: The CONCAT() function is used to concatenate (combine) two or more strings into one string.

Syntax:

CONCAT(string1, string2, ...)

Example:

SELECT CONCAT('Hello', ' ', 'World') AS greeting;

2. NOW()

Description: The NOW() function returns the current date and time.

Syntax:

NOW()

Example:

SELECT NOW() AS current_datetime;

3. ROUND()

Description: The ROUND() function rounds a number to a specified number of decimal places.

Syntax:

ROUND(number, decimals)

Example:

SELECT ROUND(123.4567, 2) AS rounded_number;

4. DATEDIFF()

Description: The DATEDIFF() function calculates the difference in days between two dates.

Syntax:

DATEDIFF(date1, date2)

Example:

SELECT DATEDIFF('2024-09-10', '2024-09-02') AS days_difference;

2019 Q.No 2 (a)

CREATE DATABASE patient_db;

```
CREATE TABLE patients (
    id INT(11) AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    address TEXT NOT NULL,
    contact VARCHAR(15) NOT NULL,
    gender ENUM('Male', 'Female') NOT NULL
);
```

Step 2: Create the PHP Script

Here's a basic PHP script to handle the CRUD operations.

1. connect.php: This file connects to the database.

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

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
```

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

3. **index.php: The main file with the form and CRUD operations.**

```
<?php include 'connect.php'; ?>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Patient Form</title>
</head>
<body>
    <h2>Patient Form</h2>

    <form action="" method="POST">
        <label for="patient_id">Patient ID:</label>
        <input type="text" name="id" id="patient_id"><br>

        <label for="name">Name:</label>
        <input type="text" name="name" id="name"><br>

        <label for="address">Address:</label>
        <textarea name="address" id="address"></textarea><br>
```

```
    <label for="contact">Contact:</label>
    <input type="text" name="contact" id="contact"><br>

    <label for="gender">Gender:</label>
    <input type="radio" name="gender" value="Male"> Male
    <input type="radio" name="gender" value="Female"> Female<br>

    <input type="submit" name="save" value="Save">
    <input type="submit" name="show" value="Show">
    <input type="submit" name="update" value="Update">
    <input type="submit" name="delete" value="Delete">
</form>

<?php
if (isset($_POST['save'])) {
    $name = $_POST['name'];
    $address = $_POST['address'];
    $contact = $_POST['contact'];
    $gender = $_POST['gender'];

    $sql = "INSERT INTO patients (name, address, contact, gender) VALUES
('$name', '$address', '$contact', '$gender')";
    if ($conn->query($sql) === TRUE) {
        echo "New record created successfully";
    } else {
        echo "Error: " . $sql . "<br>" . $conn->error;
```

```
    }
  }

  if (isset($_POST['show'])) {
    $sql = "SELECT * FROM patients";
    $result = $conn->query($sql);

    if ($result->num_rows > 0) {
      while($row = $result->fetch_assoc()) {
        echo "ID: " . $row["id"]. " - Name: " . $row["name"]. " - Address: " . $row["address"]. " - Contact: " . $row["contact"]. " - Gender: " . $row["gender"]. "<br>";
      }
    } else {
      echo "0 results";
    }
  }

  if (isset($_POST['update'])) {
    $id = $_POST['id'];
    $name = $_POST['name'];
    $address = $_POST['address'];
    $contact = $_POST['contact'];
    $gender = $_POST['gender'];

    $sql = "UPDATE patients SET name='$name', address='$address', contact='$contact', gender='$gender' WHERE id=$id";
```

```
        if ($conn->query($sql) === TRUE) {
            echo "Record updated successfully";
        } else {
            echo "Error updating record: " . $conn->error;
        }
    }

    if (isset($_POST['delete'])) {
        $id = $_POST['id'];

        $sql = "DELETE FROM patients WHERE id=$id";
        if ($conn->query($sql) === TRUE) {
            echo "Record deleted successfully";
        } else {
            echo "Error deleting record: " . $conn->error;
        }
    }

    $conn->close();
    ?>
</body>
</html>
```