EJEMPLO DE CRUD (Create-Read-Update-Delete)

Conexión.php

```php
<?php
// Se definen las variables que contienen la información necesaria para conectarse a la base de datos.
$servername = "localhost"; // El nombre del servidor donde está alojada la base de datos.
$username = "magux";       // El nombre de usuario para acceder a la base de datos.
$password = "gAsDev=7A1B??"; // La contraseña correspondiente al usuario para acceder a la base de datos.
$dbname = "db_crud1";      // El nombre de la base de datos a la que se quiere acceder.

// Se crea una nueva conexión a la base de datos utilizando la clase mysqli.
// Se pasan las variables definidas anteriormente como parámetros al constructor de mysqli.
$conn = new mysqli($servername, $username, $password, $dbname);

// Se verifica si hay algún error en la conexión.
if ($conn->connect_error) {
   // Si hay un error de conexión, se muestra un mensaje y se detiene la ejecución del script.
   die("Error de conexión: " . $conn->connect_error);
}
// Si la conexión es exitosa, el script continúa ejecutándose.
// No se muestra ningún mensaje de éxito, pero se puede asumir que a partir de este punto
// la variable $conn se puede usar para interactuar con la base de datos.
?>
```

Explicación del código:

1. **Definición de las variables de conexión:**

```
$servername = "localhost";
$username = "magux";
$password = "gAsDev=7A1B??";
$dbname = "db_crud1";
```

donde:

$servername: Define el nombre del servidor de la base de datos. En este caso, "localhost" indica que la base de datos está alojada en el mismo servidor donde se está ejecutando el script.

- $username: Define el nombre de usuario utilizado para conectarse a la base de datos.
- $password: Define la contraseña correspondiente al usuario para la conexión a la base de datos.
- $dbname: Define el nombre de la base de datos a la que se quiere acceder.

- **Creación de la conexión:**

```
$conn = new mysqli($servername, $username, $password, $dbname);
```

donde:

Aquí se crea un nuevo objeto mysqli utilizando las variables definidas anteriormente. Este objeto ($conn) se utilizará para interactuar con la base de datos.

- **Verificación de la conexión:**

```
if ($conn->connect_error) {
   die("Error de conexión: " . $conn->connect_error);
}
```

donde:

Esta sección del código verifica si hubo algún error al intentar conectar con la base de datos.

- **connect_error** es una propiedad del objeto mysqli que contiene cualquier error ocurrido durante el intento de conexión.
- Si connect_error no está vacío, significa que hubo un error de conexión. En este caso, se utiliza **die()** para mostrar un mensaje de error y detener la ejecución del script.

Flujo del código:

1. **Definición de variables**: El script define las variables necesarias para la conexión a la base de datos.
2. **Creación de la conexión**: Se crea una conexión a la base de datos utilizando la clase mysqli.
3. **Verificación de errores**: Se verifica si la conexión fue exitosa. Si hubo un error, se muestra un mensaje y se detiene la ejecución. Si la conexión es exitosa, el script continúa ejecutándose normalmente.

Este es un script típico para establecer una conexión a una base de datos MySQL utilizando PHP. Es esencial para cualquier operación que requiera interacción con una base de datos, como consultas, inserciones, actualizaciones, y eliminaciones de registros.

index.php

```
<?php
        // Incluir el archivo de conexión a la base de datos
        include('conexion.php');

        // Consulta SQL para obtener todos los usuarios
        $sql = "SELECT * FROM usuarios";
        $result = $conn->query($sql);
?>

<!DOCTYPE html>
<html>
<head>
  <title>Listado de Usuarios</title>
</head>
<body>
  <!-- Título de la sección de la página -->
  <h1>Listado de Usuarios</h1>
```

```
    <!-- Tabla para mostrar la lista de usuarios -->
    <table border="1">
      <tr>
        <th>ID</th>
        <th>Nombre</th>
        <th>Email</th>
        <th>Acciones</th>
      </tr>
      <!-- Código PHP para rellenar la tabla con los datos de la base de datos -->
      <?php
      // Comprobar si hay usuarios en el resultado de la consulta
      if ($result->num_rows > 0) {
        // Recorrer cada fila del resultado y mostrarla en la tabla
        while($row = $result->fetch_assoc()) {
          echo "<tr>";
          echo "<td>" . $row["id"] . "</td>";
          echo "<td>" . $row["nombre"] . "</td>";
          echo "<td>" . $row["email"] . "</td>";
          // Enlaces para editar y eliminar cada usuario
              echo "<td><a href='editar_usuario.php?id=" . $row["id"] . "'>Editar</a> | <a href='eliminar_usuario.php?id=" . $row["id"] . "'>Eliminar</a></td>";
          echo "</tr>";
        }
      } else {
        // Mostrar un mensaje si no hay usuarios registrados
        echo "<tr><td colspan='4'>No hay usuarios registrados.</td></tr>";
      }
      ?>

    </table>
    <br>
    <!-- Enlace para agregar un nuevo usuario -->
      <a style="color: yellow; display: block; text-align: right;" href="formulario.php">Agregar Usuario</a>
   </body>
   </html>
  </body>
</html>
```

Explicación del código:

Bloque PHP para obtener usuarios de la base de datos:

```
<?php
// Incluir el archivo de conexión a la base de datos
include('conexion.php');

// Consulta SQL para obtener todos los usuarios
$sql = "SELECT * FROM usuarios";
$result = $conn->query($sql);
?>
```

include('conexion.php');: Incluye el archivo conexion.php que establece la conexión a la base de datos.

- $sql = "SELECT * FROM usuarios";: Define una consulta SQL para seleccionar todos los registros de la tabla usuarios.
- $result = $conn->query($sql);: Ejecuta la consulta SQL y guarda el resultado en la variable $result.

Estructura HTML para mostrar la lista de usuarios:

```
<!DOCTYPE html>
<html>
	<head>
		<title>Listado de Usuarios</title>
	</head>
	<body>
		<h1>Listado de Usuarios</h1>
		<table border="1">
		  <tr>
		    <th>ID</th>
		    <th>Nombre</th>
		    <th>Email</th>
		    <th>Acciones</th>
		  </tr>
```

Bloque PHP para rellenar la tabla con los datos de la base de datos:

```
<?php
if ($result->num_rows > 0) {
  while($row = $result->fetch_assoc()) {
    echo "<tr>";
    echo "<td>" . $row["id"] . "</td>";
    echo "<td>" . $row["nombre"] . "</td>";
    echo "<td>" . $row["email"] . "</td>";
          echo "<td><a href='editar_usuario.php?id=" . $row["id"] . "'>Editar</a> | <a href='eliminar_usuario.php?id=" . $row["id"] . "'>Eliminar</a></td>";
    echo "</tr>";
  }
} else {
  echo "<tr><td colspan='4'>No hay usuarios registrados.</td></tr>";
}
?>
```

if ($result->num_rows > 0) { ... }: Comprueba si hay registros en el resultado de la consulta.

- while($row = $result->fetch_assoc()) { ... }: Itera sobre cada fila del resultado de la consulta.
- echo "<tr>"; ... echo "</tr>";: Crea filas en la tabla para cada registro.
- <td>: Define una celda en la tabla.
- Los enlaces de edición y eliminación (editar_usuario.php y eliminar_usuario.php) permiten realizar acciones sobre cada usuario.

Enlace para agregar un nuevo usuario:

```
<br>
<a style="color: yellow; display: block; text-align: right;" href="formulario.php">Agregar Usuario</a>
```

Este código combina HTML y PHP para mostrar una lista de usuarios obtenida de una base de datos MySQL. La lista incluye enlaces para editar y eliminar usuarios, así como un enlace para agregar nuevos usuarios.

agregar_usuario.php

```php
<?php
// Incluir el archivo de conexión a la base de datos
include('conexion.php');

// Obtener los datos enviados desde el formulario mediante el método POST
$nombre = $_POST['nombre'];        // Nombre del usuario
$email = $_POST['email'];          // Email del usuario
$contraseña = $_POST['contraseña'];  // Contraseña del usuario

// Construir la consulta SQL para insertar un nuevo usuario en la base de datos
$sql = "INSERT INTO usuarios (nombre, email, contraseña) VALUES ('$nombre', '$email', '$contraseña')";

// Ejecutar la consulta y comprobar si fue exitosa
if ($conn->query($sql) === TRUE) {
   // Si la inserción fue exitosa, redirigir al usuario a la página de inicio
   header("Location: index.php");
   exit();  // Terminar la ejecución del script
} else {
   // Si hubo un error en la inserción, mostrar un mensaje de error
   echo "Error al agregar el usuario: " . $conn->error;
}

// Cerrar la conexión a la base de datos
$conn->close();
?>
```

Explicación del código:

1. Incluir el archivo de conexión a la base de datos:

```
include('conexion.php');
```

include('conexion.php');: Este comando incluye el archivo conexion.php, que establece la conexión con la base de datos MySQL. Este archivo debe contener el código necesario para crear la conexión a la base de datos y asignarla a la variable $conn.

- **Obtener los datos enviados desde el formulario:**

```
$nombre = $_POST['nombre'];
$email = $_POST['email'];
$contraseña = $_POST['contraseña'];
```

$_POST['nombre']: Captura el valor del campo nombre enviado por el formulario HTML.

- $_POST['email']: Captura el valor del campo email enviado por el formulario HTML.
- $_POST['contraseña']: Captura el valor del campo contraseña enviado por el formulario HTML.

Construir la consulta SQL:

```
$sql = "INSERT INTO usuarios (nombre, email, contraseña)
        VALUES ('$nombre', '$email', '$contraseña')";
```

Esta línea construye una consulta SQL para insertar un nuevo registro en la tabla usuarios con los valores capturados del formulario.

Ejecutar la consulta SQL y manejar el resultado:

```
if ($conn->query($sql) === TRUE) {
   header("Location: index.php");
   exit();
} else {
   echo "Error al agregar el usuario: " . $conn->error;
}
```

if ($conn->query($sql) === TRUE): Ejecuta la consulta SQL y comprueba si fue exitosa.

- header("Location: index.php"); exit();: Si la inserción fue exitosa, redirige al usuario a la página index.php y termina la ejecución del script.
- else { echo "Error al agregar el usuario: " . $conn->error; }: Si hubo un error en la inserción, muestra un mensaje de error junto con la descripción del error proporcionada por la base de datos.
- **Cerrar la conexión a la base de datos:**

```
$conn→close();
```

conn->close();: Cierra la conexión a la base de datos, liberando los recursos asociados con la conexión.

Comentarios adicionales:

- **Seguridad:** Este código es vulnerable a ataques de inyección SQL. Para evitar esto, se recomienda utilizar sentencias preparadas (prepared statements) con parámetros enlazados.
- **Validación:** Es buena práctica validar y sanitizar los datos recibidos del usuario antes de procesarlos. Esto incluye verificar que los campos no estén vacíos y que el correo electrónico tenga un formato válido.
- **Contraseñas:** Las contraseñas nunca deben almacenarse en texto plano. Utiliza funciones de hash como password_hash() para almacenar contraseñas de forma segura.

Ejemplo mejorado utilizando sentencias preparadas:

Aquí tienes una versión mejorada del script utilizando sentencias preparadas para evitar inyecciones SQL:

```
<?php
include('conexion.php');

$nombre = $_POST['nombre'];
$email = $_POST['email'];
$contraseña = password_hash($_POST['contraseña'], PASSWORD_DEFAULT); // Hashear la contraseña

// Preparar la consulta SQL para insertar un nuevo usuario
$sql = $conn->prepare("INSERT INTO usuarios (nombre, email, contraseña) VALUES (?, ?, ?)");
$sql->bind_param("sss", $nombre, $email, $contraseña);

// Ejecutar la consulta y comprobar si fue exitosa
if ($sql->execute() === TRUE) {
    header("Location: index.php");
    exit();
} else {
    echo "Error al agregar el usuario: " . $conn->error;
}

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

En esta versión, se utiliza password_hash para hashear la contraseña y mysqli::prepare para preparar la consulta SQL, lo que mejora la seguridad del script.

eliminar_usuario.php

```php
<?php
// Incluir el archivo de conexión a la base de datos
include('conexion.php');

// Obtener el ID del usuario a eliminar desde la URL mediante el método GET
$id = $_GET['id'];

// Construir la consulta SQL para eliminar un usuario de la base de datos
$sql = "DELETE FROM usuarios WHERE id=$id";

// Ejecutar la consulta y comprobar si fue exitosa
if ($conn->query($sql) === TRUE) {
    // Si la eliminación fue exitosa, redirigir al usuario a la página de inicio
    header("Location: index.php");
    exit();  // Terminar la ejecución del script
} else {
    // Si hubo un error en la eliminación, mostrar un mensaje de error
    echo "Error al eliminar el usuario: " . $conn->error;
}

// Cerrar la conexión a la base de datos
$conn->close();
?>
```

Explicación del código:

1. Incluir el archivo de conexión a la base de datos:

```
include('conexion.php');
```

include('conexion.php');: Este comando incluye el archivo conexion.php, que establece la conexión con la base de datos MySQL. Este archivo debe contener el código necesario para crear la conexión a la base de datos y asignarla a la variable $conn.

Obtener el ID del usuario a eliminar:

```
$id = $_GET['id'];
```

$_GET['id']: Captura el valor del parámetro id enviado en la URL mediante el método GET. Este valor corresponde al ID del usuario que se desea eliminar.

Construir la consulta SQL

```
$sql = "DELETE FROM usuarios WHERE id=$id";
```

Esta línea construye una consulta SQL para eliminar el registro de la tabla usuarios que tenga el ID especificado.

Ejecutar la consulta SQL y manejar el resultado:

```
if ($conn->query($sql) === TRUE) {
    header("Location: index.php");
    exit();
} else {
    echo "Error al eliminar el usuario: " . $conn->error;
}
```

if ($conn->query($sql) === TRUE): Ejecuta la consulta SQL y comprueba si fue exitosa.

- header("Location: index.php"); exit();: Si la eliminación fue exitosa, redirige al usuario a la página index.php y termina la ejecución del script.
- else { echo "Error al eliminar el usuario: " . $conn->error; }: Si hubo un error en la eliminación, muestra un mensaje de error junto con la descripción del error proporcionada por la base de datos.

Comentarios adicionales:

- **Seguridad:** Este código es vulnerable a ataques de inyección SQL. Para evitar esto, se recomienda utilizar sentencias preparadas (prepared statements) con parámetros enlazados.
- **Validación:** Es buena práctica validar y sanitizar los datos recibidos del usuario antes de procesarlos. Esto incluye verificar que el ID recibido es un número entero válido.

Ejemplo mejorado utilizando sentencias preparadas:

Aquí tienes una versión mejorada del script utilizando sentencias preparadas para evitar inyecciones SQL:

```
<?php
include('conexion.php');

// Obtener el ID del usuario a eliminar desde la URL mediante el método GET
$id = $_GET['id'];

// Preparar la consulta SQL para eliminar un usuario
$sql = $conn->prepare("DELETE FROM usuarios WHERE id = ?");
$sql->bind_param("i", $id);

// Ejecutar la consulta y comprobar si fue exitosa
if ($sql->execute() === TRUE) {
    header("Location: index.php");
    exit();
} else {
    echo "Error al eliminar el usuario: " . $conn->error;
}
// Cerrar la conexión a la base de datos
$conn->close();
?>
```

En esta versión, se utiliza mysqli::prepare para preparar la consulta SQL, lo que mejora la seguridad del script. La variable $id se enlaza a la consulta preparada utilizando bind_param, lo que ayuda a prevenir inyecciones SQL.

editar_usuario.php

```
<?php
// Incluir el archivo de conexión a la base de datos
include('conexion.php');

// Obtener el ID del usuario a editar desde la URL mediante el método GET
$id = $_GET['id'];

// Construir la consulta SQL para obtener la información del usuario seleccionado
$sql = "SELECT * FROM usuarios WHERE id=$id";

// Ejecutar la consulta y obtener el resultado
$result = $conn->query($sql);

// Obtener los datos del usuario en un array asociativo
$row = $result->fetch_assoc();
?>
<!DOCTYPE html>
<html>
<head>
<title>Editar Usuario</title>
</head>
<body>
<h1>Editar Usuario</h1>
<!-- Formulario para editar la información del usuario -->
<form action="guardar_edicion.php" method="POST">
    <!-- Campo oculto para enviar el ID del usuario -->
    <input type="hidden" name="id" value="<?php echo $row['id']; ?>">

    <!-- Campo para editar el nombre del usuario -->
    Nombre: <input type="text" name="nombre" value="<?php echo $row['nombre']; ?>"><br>

    <!-- Campo para editar el email del usuario -->
    Email: <input type="email" name="email" value="<?php echo $row['email']; ?>"><br>
```

```
    <!-- Campo para editar la contraseña del usuario -->
        Contraseña: <input type="password" name="contraseña" value="<?php echo $row['contraseña']; ?>"><br>

    <!-- Botón para enviar el formulario -->
    <input type="submit" value="Guardar Cambios">
</form>
<br>
<!-- Enlace para volver al listado de usuarios -->
<a href="index.php">Volver al Listado de Usuarios</a>
</body>
</html>
```

Explicación del código:

1. Incluir el archivo de conexión a la base de datos:

```
include('conexion.php');
```

include('conexion.php');: Este comando incluye el archivo conexion.php, que establece la conexión con la base de datos MySQL. Este archivo debe contener el código necesario para crear la conexión a la base de datos y asignarla a la variable $conn.

- **Obtener el ID del usuario a editar:**

```
$id = $_GET['id'];
```

$_GET['id']: Captura el valor del parámetro id enviado en la URL mediante el método GET. Este valor corresponde al ID del usuario que se desea editar.

- **Construir la consulta SQL:**

```
$sql = "SELECT * FROM usuarios WHERE id=$id";
```

Esta línea construye una consulta SQL para seleccionar el registro de la tabla usuarios que tiene el ID especificado.

- **Ejecutar la consulta SQL y obtener el resultado:**

```
$result = $conn->query($sql);
$row = $result→fetch_assoc();
```

result = $conn->query($sql);: Ejecuta la consulta SQL y guarda el resultado en la variable $result.

$row = $result->fetch_assoc();: Obtiene los datos del usuario como un array asociativo y los guarda en la variable $row.

- **Estructura HTML para mostrar el formulario de edición:**

```
<!DOCTYPE html>
<html>
<head>
<title>Editar Usuario</title>
</head>
<body>
<h1>Editar Usuario</h1>
<form action="guardar_edicion.php" method="POST">
   <input type="hidden" name="id" value="<?php echo $row['id']; ?>">
 Nombre: <input type="text" name="nombre" value="<?php echo $row['nombre']; ?>"><br>
 Email: <input type="email" name="email" value="<?php echo $row['email']; ?>"><br>
Contraseña: <input type="password" name="contraseña" value="<?php echo $row['contraseña']; ?>"><br>
   <input type="submit" value="Guardar Cambios">
</form>
<br>
<a href="index.php">Volver al Listado de Usuarios</a>
</body>
</html>
```

<!DOCTYPE html>: Define el documento como HTML5.

1. **<html>:** Marca el inicio del documento HTML.
2. **<head>:** Contiene metadatos sobre el documento, como el título.
3. **<title>**Editar Usuario**</title>:** Define el título de la página que aparece en la pestaña del navegador.
4. **<body>:** Contiene el contenido visible de la página.
5. **<h1>**Editar Usuario**</h1>**: Título principal de la sección de la página.
6. **<form action="guardar_edicion.php" method="POST">**: Define un formulario que envía los datos a **guardar_edicion.php** mediante el método **POST**.
7. **<input type="hidden" name="id" value="<?php echo $row['id']; ?>">**: Campo oculto que contiene el ID del usuario.
8. **<input type="text" name="nombre" value="<?php echo $row['nombre']; ?>">**
: Campo de texto para editar el nombre del usuario.
9. **<input type="email" name="email" value="<?php echo $row['email']; ?>">**
: Campo de correo electrónico para editar el email del usuario.
10. **<input type="password" name="contraseña" value="<?php echo $row['contraseña']; ?>">**
: Campo de contraseña para editar la contraseña del usuario.
11. **<input type="submit" value="Guardar Cambios">:** Botón para enviar el formulario.
12. Volver al Listado de Usuarios: Enlace para volver al listado de usuarios.

Comentarios adicionales:

- **Seguridad:** Este código es vulnerable a ataques de inyección SQL. Para evitar esto, se recomienda utilizar sentencias preparadas (prepared statements) con parámetros enlazados.
- **Validación:** Es buena práctica validar y sanitizar los datos recibidos del usuario antes de procesarlos. Esto incluye verificar que el ID recibido es un número entero válido.
- **Contraseñas:** Las contraseñas nunca deben almacenarse en texto plano. Utiliza funciones de hash como password_hash() para almacenar contraseñas de forma segura.

Ejemplo mejorado utilizando sentencias preparadas:

Aquí tienes una versión mejorada del script utilizando sentencias preparadas para evitar inyecciones SQL:

```
<?php
        include('conexion.php');
        // Obtener el ID del usuario a editar desde la URL mediante el método GET
        $id = $_GET['id'];
        // Preparar la consulta SQL para obtener la información del usuario
        $sql = $conn->prepare("SELECT * FROM usuarios WHERE id = ?");
        $sql->bind_param("i", $id);
        $sql->execute();
        $result = $sql->get_result();
        $row = $result->fetch_assoc();
?>
<!DOCTYPE html>
<html>
<head>
<title>Editar Usuario</title>
</head>
<body>
<h1>Editar Usuario</h1>
<form action="guardar_edicion.php" method="POST">
   <input type="hidden" name="id" value="<?php echo $row['id']; ?>">
   Nombre: <input type="text" name="nombre" value="<?php echo htmlspecialchars($row['nombre']); ?
>"><br>
   Email: <input type="email" name="email" value="<?php echo htmlspecialchars($row['email']); ?>"><br>
   Contraseña: <input type="password" name="contraseña" value="<?php echo
htmlspecialchars($row['contraseña']); ?>"><br>
   <input type="submit" value="Guardar Cambios">
</form>
<br>
<a href="index.php">Volver al Listado de Usuarios</a>
</body>
</html>
```

En esta versión, se utiliza mysqli::prepare para preparar la consulta SQL, lo que mejora la seguridad del script. La variable $id se enlaza a la consulta preparada utilizando bind_param, lo que ayuda a prevenir inyecciones SQL. Además, se utiliza htmlspecialchars() para escapar los datos mostrados en el formulario, protegiendo contra inyecciones de HTML.

formulario.php

```
<!DOCTYPE html>
<html>
<head>
  <title>Agregar Usuario</title>
</head>
<body>
  <h1>Agregar Usuario</h1>
  <!-- Formulario para agregar un nuevo usuario -->
  <form action="agregar_usuario.php" method="POST">
    <!-- Campo para ingresar el nombre del usuario -->
    Nombre: <input type="text" name="nombre"><br>

    <!-- Campo para ingresar el correo electrónico del usuario -->
    Email: <input type="email" name="email"><br>

    <!-- Campo para ingresar la contraseña del usuario -->
    Contraseña: <input type="password" name="contraseña"><br>

    <!-- Botón para enviar el formulario -->
    <input type="submit" value="Agregar Usuario">
  </form>
  <br>
  <!-- Enlace para volver al listado de usuarios -->
  <a href="index.php">Volver al Listado de Usuarios</a>
</body>
</html>
```

Explicación del código:

- **Formulario para agregar un nuevo usuario:**

```
<form action="agregar_usuario.php" method="POST">
  Nombre: <input type="text" name="nombre"><br>
  Email: <input type="email" name="email"><br>
  Contraseña: <input type="password" name="contraseña"><br>
  <input type="submit" value="Agregar Usuario">
</form>
```

donde:

<form action="agregar_usuario.php" method="POST">: Define un formulario que envía los datos a agregar_usuario.php mediante el método POST.

- Nombre: <input type="text" name="nombre">
: Campo de texto para ingresar el nombre del usuario.
- Email: <input type="email" name="email">
: Campo de correo electrónico para ingresar el email del usuario.
- Contraseña: <input type="password" name="contraseña">
: Campo de contraseña para ingresar la contraseña del usuario.
- <input type="submit" value="Agregar Usuario">: Botón para enviar el formulario.

Enlace para volver al listado de usuarios:

```
<a href="index.php">Volver al Listado de Usuarios</a>
```

Volver al Listado de Usuarios: Enlace para volver al listado de usuarios. Permite al usuario regresar a la página principal donde se muestra la lista de usuarios.

Comentarios adicionales:

- **Validación:** Es buena práctica validar los datos recibidos del usuario antes de procesarlos. Esto incluye verificar que los campos no estén vacíos y que el correo electrónico tenga un formato válido.
- **Seguridad:** Las contraseñas nunca deben almacenarse en texto plano. Utiliza funciones de hash como password_hash() para almacenar contraseñas de forma segura.
- **Sanitización:** Sanitiza los datos del formulario para evitar inyecciones de HTML y ataques XSS.

Ejemplo mejorado con validación y sanitización:

Aquí tienes una versión mejorada del formulario con validación y sanitización básica:

```
<!DOCTYPE html>
<html>
<head>
  <title>Agregar Usuario</title>
</head>
<body>
  <h1>Agregar Usuario</h1>
  <form action="agregar_usuario.php" method="POST" onsubmit="return validarFormulario()">
    Nombre: <input type="text" name="nombre" required><br>
    Email: <input type="email" name="email" required><br>
```

```
    Contraseña: <input type="password" name="contraseña" required><br>
    <input type="submit" value="Agregar Usuario">
  </form>
  <br>
  <a href="index.php">Volver al Listado de Usuarios</a>
  <script>
    function validarFormulario() {
      var nombre = document.forms[0]["nombre"].value;
      var email = document.forms[0]["email"].value;
      var contraseña = document.forms[0]["contraseña"].value;

      if (nombre == "" || email == "" || contraseña == "") {
        alert("Todos los campos son obligatorios.");
        return false;
      }
      return true;
    }
  </script>
</body>
</html>
```

En esta versión, se utiliza JavaScript para validar que todos los campos del formulario estén completos antes de enviar los datos. Además, se añade el atributo required a los campos del formulario para que el navegador también realice una validación básica.

guardar_edicion.php

```php
<?php
// Incluir el archivo de conexión a la base de datos
include('conexion.php');

// Obtener los datos enviados desde el formulario mediante el método POST
$id = $_POST['id'];            // ID del usuario a actualizar
$nombre = $_POST['nombre'];    // Nombre del usuario
$email = $_POST['email'];      // Email del usuario
$contraseña = $_POST['contraseña']; // Contraseña del usuario

// Construir la consulta SQL para actualizar la información del usuario
$sql = "UPDATE usuarios SET nombre='$nombre', email='$email', contraseña='$contraseña'
WHERE id=$id";

// Ejecutar la consulta y comprobar si fue exitosa
if ($conn->query($sql) === TRUE) {
   // Si la actualización fue exitosa, redirigir al usuario a la página de inicio
   header("Location: index.php");
   exit();  // Terminar la ejecución del script
} else {
   // Si hubo un error en la actualización, mostrar un mensaje de error
   echo "Error al actualizar el usuario: " . $conn->error;
}

// Cerrar la conexión a la base de datos
$conn->close();
?>
```

Explicación del código:

1. Incluir el archivo de conexión a la base de datos:

```php
include('conexion.php');
```

include('conexion.php');: Este comando incluye el archivo conexion.php, que establece la conexión con la base de datos MySQL. Este archivo debe contener el código necesario para crear la conexión a la base de datos y asignarla a la variable $conn.

- **Obtener los datos enviados desde el formulario:**

```php
$id = $_POST['id'];
$nombre = $_POST['nombre'];
$email = $_POST['email'];
$contraseña = $_POST['contraseña'];
```

$_POST['id']: Captura el valor del campo id enviado por el formulario HTML. Este valor corresponde al ID del usuario que se desea actualizar.

- $_POST['nombre']: Captura el valor del campo nombre enviado por el formulario HTML.
- $_POST['email']: Captura el valor del campo email enviado por el formulario HTML.
- $_POST['contraseña']: Captura el valor del campo contraseña enviado por el formulario HTML.

Construir la consulta SQL:

```
$sql = "UPDATE usuarios SET nombre='$nombre', email='$email', contraseña='$contraseña' WHERE id=$id";
```

Esta línea construye una consulta SQL para actualizar el registro de la tabla usuarios que tiene el ID especificado. Se actualizan los campos nombre, email y contraseña con los valores proporcionados por el formulario.

Ejecutar la consulta SQL y manejar el resultado:

```
if ($conn->query($sql) === TRUE) {
    header("Location: index.php");
    exit();
} else {
    echo "Error al actualizar el usuario: " . $conn->error;
}
```

if ($conn->query($sql) === TRUE): Ejecuta la consulta SQL y comprueba si fue exitosa.

- header("Location: index.php"); exit();: Si la actualización fue exitosa, redirige al usuario a la página index.php y termina la ejecución del script.
- else { echo "Error al actualizar el usuario: " . $conn->error; }: Si hubo un error en la actualización, muestra un mensaje de error junto con la descripción del error proporcionada por la base de datos.

Cerrar la conexión a la base de datos:

```
$conn→close();
```

conn->close();: Cierra la conexión a la base de datos, liberando los recursos asociados con la conexión.

Comentarios adicionales:

- **Seguridad:** Este código es vulnerable a ataques de inyección SQL. Para evitar esto, se recomienda utilizar sentencias preparadas (prepared statements) con parámetros enlazados.
- **Validación:** Es buena práctica validar y sanitizar los datos recibidos del usuario antes de procesarlos. Esto incluye verificar que los campos no estén vacíos y que el correo electrónico tenga un formato válido.
- **Contraseñas:** Las contraseñas nunca deben almacenarse en texto plano. Utiliza funciones de hash como password_hash() para almacenar contraseñas de forma segura.

Ejemplo mejorado utilizando sentencias preparadas:

Aquí tienes una versión mejorada del script utilizando sentencias preparadas para evitar inyecciones SQL:

```
<?php
include('conexion.php');

// Obtener los datos enviados desde el formulario mediante el método POST
$id = $_POST['id'];
$nombre = $_POST['nombre'];
$email = $_POST['email'];
$contraseña = password_hash($_POST['contraseña'], PASSWORD_DEFAULT); // Hashear la contraseña

// Preparar la consulta SQL para actualizar la información del usuario
$sql = $conn->prepare("UPDATE usuarios SET nombre = ?, email = ?, contraseña = ? WHERE id = ?");
$sql->bind_param("sssi", $nombre, $email, $contraseña, $id);

// Ejecutar la consulta y comprobar si fue exitosa
if ($sql->execute() === TRUE) {
    header("Location: index.php");
    exit();
} else {
    echo "Error al actualizar el usuario: " . $conn->error;
}

// Cerrar la conexión a la base de datos
$conn->close();
?>
```

En esta versión, se utiliza password_hash para hashear la contraseña y mysqli::prepare para preparar la consulta SQL, lo que mejora la seguridad del script. La variable $id se enlaza a la consulta preparada utilizando bind_param, lo que ayuda a prevenir inyecciones SQL.

BASE DE DATOS

db_crud1.sql

```
-- Estructura de tabla para la tabla `usuarios`
CREATE TABLE `usuarios` (
  `id` int(11) NOT NULL,
  `nombre` varchar(50) NOT NULL,
  `email` varchar(100) NOT NULL,
  `contraseña` varchar(255) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
-- Volcado de datos para la tabla `usuarios`
INSERT INTO `usuarios` (`id`, `nombre`, `email`, `contraseña`) VALUES
(2, 'María López', 'maria.lopez@example.com', 'maria2024'),
(3, 'Carlos García', 'carlos.garcia@example.com', 'carlit0s'),
(4, 'Ana Torres', 'ana.torres@example.com', 'ana_321');
-- Estructura de tabla para la tabla `vehiculos`
CREATE TABLE `vehiculos` (
  `id` int(11) NOT NULL,
  `marca` varchar(255) NOT NULL,
  `modelo` varchar(255) NOT NULL,
  `cilindrada` int(11) NOT NULL,
  `fechaFabricacion` date NOT NULL,
  `subclase` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
-- Volcado de datos para la tabla `vehiculos`
INSERT INTO `vehiculos` (`id`, `marca`, `modelo`, `cilindrada`, `fechaFabricacion`, `subclase`)
VALUES
(1, 'Marca1', 'Modelo1', 150, '2022-01-01', 'Moto'),
(2, 'Marca2', 'Modelo2', 2000, '2021-05-15', 'Coche'),
(3, 'Marca3', 'Modelo3', 4000, '2020-11-30', 'Camion');
-- Índices para tablas volcadas
-- Indices de la tabla `usuarios`
ALTER TABLE `usuarios`
  ADD PRIMARY KEY (`id`);
-- Indices de la tabla `vehiculos`

ALTER TABLE `vehiculos`
  ADD PRIMARY KEY (`id`);
-- AUTO_INCREMENT de las tablas volcadas
-- AUTO_INCREMENT de la tabla `usuarios`
ALTER TABLE `usuarios`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=6;
-- AUTO_INCREMENT de la tabla `vehiculos`
ALTER TABLE `vehiculos`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=4;
```