Práctica 10 de PHP. Programación de bases de datos 1 para el producto Developer sin Panza Transmite Confianza

TEORIA

Casos de uso fundamentales

1. **Configuración y Conexión a la base de datos:**
    - **Definir los parámetros de conexión:** host, usuario, contraseña y nombre de la base de datos.
    - **Establecer la conexión** con la función `mysqli_connect()`.
    - **Comprobar si la conexión es exitosa:** Si falla, imprimir el error usando `mysqli_connect_error()` y detener la ejecución del script (por ejemplo, con `die()`).
2. **Operaciones CRUD (Create, Read, Update, Delete):**
    - **INSERT:** Insertar registros en la base de datos.
    - **SELECT:** Recuperar datos de la base de datos.
    - **UPDATE:** Actualizar registros existentes.
    - **DELETE:** Eliminar registros de la base de datos.
3. **Gestión de resultados:**
    - **Ejecutar consultas:** Utilizando `mysqli_query()`.
    - **Obtener resultados:** Mediante funciones como `mysqli_fetch_assoc()` o `mysqli_fetch_array()` para recorrer los datos.
4. **Cierre de conexión:**
    - Cerrar la conexión con `mysqli_close()` una vez que se han realizado todas las operaciones.
5. **Uso de consultas preparadas:**
    - Para prevenir inyecciones SQL y trabajar de manera segura con los parámetros de las consultas.

Configuración y conexión

Ejemplo de Configuración y conexión

```php
<?php
// Configuración: Define los parámetros de conexión
$host = "localhost:puerto";
$usuario = "tu_usuario";
$contrasena = "tu_contrasena";
$nombreBD = "nombre_de_la_base";

// Establece la conexión con la base de datos
$conexion = mysqli_connect($host, $usuario, $contrasena, $nombreBD);

// Comprueba si la conexión fue exitosa
if (!$conexion) {
    // Si falla, imprime el error y detiene la ejecución
    die("Error de conexión: " . mysqli_connect_error());
}

echo "Conexión exitosa a la base de datos."; //Comenta esta línea
                                             // en el código final

// Finalmente, cierra la conexión
mysqli_close($conexion);     // Comenta esta línea
                             // en el código final
?>
```

Este bloque, como buena práctica, es interesante ponerlo en un archivo PHP aparte (por ejemplo conexiondb.php, e incluirlo con `require_once` en las páginas PHP que hagan operaciones con la BBDD.

Explicación del ejemplo

- **Configuración:**
  Se definen variables para el host, usuario, contraseña y nombre de la base de datos.
- **Conexión:**
  `mysqli_connect()` intenta establecer la conexión utilizando esos parámetros.
- **Comprobación de la conexión:**
  Se verifica si la variable `$conexion` es falsa. En caso afirmativo, se utiliza `mysqli_connect_error()` para mostrar el mensaje de error y se detiene la ejecución con `die()`.
- **Cierre de conexión:**
  Se cierra la conexión usando `mysqli_close()` una vez que se han realizado las operaciones necesarias.

Operaciones CRUD (Create, Read, Update, Delete)

Crear registros

Hay dos casos distintos de creación de registros en una base de datos que tienen procedimientos ligeramente diferentes.:

1. **Caso 1: Tabla con clave normal (no autoincrementada):**
   Debes proporcionar el valor de la clave al insertar el registro. Se utiliza `mysqli_query` que devuelve un valor booleano indicando el éxito de la operación.
2. **Caso 2: Tabla con clave autoincrementada:**
   No es necesario suministrar el valor de la clave, y se puede obtener el ID insertado mediante `mysqli_insert_id()`.

```
<?php
// Configuración: Define los parámetros de conexión
$host = "localhost";
$usuario = "tu_usuario";
$contrasena = "tu_contrasena";
$nombreBD = "nombre_de_la_base";

// Establece la conexión con la base de datos
$conexion = mysqli_connect($host, $usuario, $contrasena, $nombreBD);

// Comprueba si la conexión fue exitosa
if (!$conexion) {
    die("Error de conexión: " . mysqli_connect_error());
}

echo "Conexión exitosa a la base de datos.<br><br>";

//
=====================================================================
// Caso 1: Inserción en una tabla con campo clave normal
// Ejemplo: Tabla 'clientes' con campos:
//   - cliente_id (clave primaria normal, NO autoincrementada)
//   - nombre
//   - email

$sql1 = "INSERT INTO clientes (cliente_id, nombre, email)
         VALUES (123, 'Juan Perez', 'juan@example.com')";

$result1 = mysqli_query($conexion, $sql1);

if (!$result1) {
    echo "Error al insertar en 'clientes': " .
mysqli_error($conexion) . "<br>";
```

```
} else {
    echo "Registro insertado en 'clientes' exitosamente.<br>";
}

echo "<br>";

//
=====================================================================
// Caso 2: Inserción en una tabla con campo clave autoincrementado
// Ejemplo: Tabla 'productos' con campos:
//   - id (clave primaria autoincrementada)
//   - nombre
//   - precio

$sql2 = "INSERT INTO productos (nombre, precio)
         VALUES ('Laptop', 1500)";

$result2 = mysqli_query($conexion, $sql2);

if (!$result2) {
    echo "Error al insertar en 'productos': " .
          mysqli_error($conexion) . "<br>";
} else {
    // Obtener el ID insertado en la tabla autoincrementada
    $idInsertado = mysqli_insert_id($conexion);
    echo "Registro insertado en 'productos' exitosamente.
          ID insertado: " . $idInsertado . "<br>";

// Puedo necesitar el ID insertado para operaciones posteriores

}

// Cierra la conexión
mysqli_close($conexion);
?>
```

Explicación

- **Conexión a la base de datos:**
  Se definen los parámetros de conexión y se establece la conexión usando `mysqli_connect()`.
  Si la conexión falla, se muestra el error mediante `mysqli_connect_error()` y se detiene la ejecución con `die()`.
- **Caso 1 (Clave normal):**
  Se inserta un registro en la tabla `clientes` incluyendo el valor de la clave (`cliente_id`), ya que no es autoincrementable.

- **Caso 2 (Clave autoincrementada):**
  Se inserta un registro en la tabla `productos` sin especificar el campo `id`.
  Después, se utiliza `mysqli_insert_id()` para obtener el valor generado automáticamente por la base de datos, por si fuera necesario utilizarlo para realizar otras operaciones que necesiten la clave del registro.
- **Cierre de conexión:**
  Finalmente, se cierra la conexión con `mysqli_close()`.

Un ejemplo clásico es el de una aplicación de gestión de usuarios y sus pedidos. Imagina que tienes una tabla `usuarios` con una columna `id` autoincrementada, y otra tabla `pedidos` que tiene una columna `usuario_id` como clave foránea.

Caso de uso:

1. **Registro de usuario:**
   - Cuando se inserta un nuevo usuario en la tabla `usuarios`, el campo `id` se autogenera de forma única, garantizando que cada usuario tenga un identificador distinto sin tener que preocuparse por asignarlo manualmente.
2. **Creación de pedidos:**
   - Después de insertar el usuario, se obtiene el `id` autoincrementado (usando, por ejemplo, `mysqli_insert_id()` en un entorno procedimental).
   - Ese `id` se utiliza para insertar registros en la tabla `pedidos`, de modo que cada pedido se relacione correctamente con el usuario que lo realizó.

Ejemplo:

Inserción de usuario:

```
$sqlUsuario = "INSERT INTO usuarios (nombre, email) VALUES ('Juan
Pérez', 'juan@example.com')";
mysqli_query($conexion, $sqlUsuario);
$usuario_id = mysqli_insert_id($conexion);  // Se obtiene el
                                            // id autoincrementado
```

Inserción de pedido para el usuario:

```
$sqlPedido = "INSERT INTO pedidos (usuario_id, producto, cantidad)
VALUES ($usuario_id, 'Laptop', 1)";
mysqli_query($conexion, $sqlPedido);
```

¿Por qué es necesario?

- **Unicidad y seguridad:** El id autoincrementado garantiza que cada registro tenga un identificador único, lo que es fundamental para mantener la integridad referencial y evitar duplicidades.
- **Relaciones entre tablas:** Permite establecer relaciones de uno a muchos (por ejemplo, un usuario puede tener múltiples pedidos) de manera sencilla y segura.
- **Automatización:** Se simplifica la inserción de nuevos registros, ya que no hay que preocuparse por generar manualmente un valor único para cada registro.

En resumen, el uso de un id autoincrementado es esencial para asignar automáticamente un identificador único a cada nuevo registro, lo que facilita la gestión y las relaciones entre datos en una base de datos relacional.

Comprobar si existe el registro antes de insertarlo utilizando su campo clave

```
// Definir el valor de la clave que se usará para comprobar la
// existencia del registro

$cliente_id = 123;

// Consulta para comprobar si ya existe
// un registro con ese cliente_id

$sql_check = "SELECT cliente_id FROM clientes WHERE cliente_id =
$cliente_id";

$result_check = mysqli_query($conexion, $sql_check);

if (!$result_check) {
    die("Error en la consulta: " . mysqli_error($conexion));
}

if (mysqli_num_rows($result_check) > 0) {
    // Si existe al menos una fila,
    // significa que el registro ya existe
    echo "El registro con cliente_id = $cliente_id ya existe.<br>";
} else {
    // Si no existe, se procede a insertar el registro
    $sql_insert = "INSERT INTO clientes (cliente_id, nombre, email)
                   VALUES ($cliente_id, 'Juan Perez',
                   'juan@example.com')";
    $result_insert = mysqli_query($conexion, $sql_insert);

    if (!$result_insert) {
        echo "Error al insertar el registro: " .
                 mysqli_error($conexion) . "<br>";
    } else {
        echo "Registro insertado exitosamente.<br>";
    }
}

// Cierra la conexión
mysqli_close($conexion);
?>
```

Explicación

1. **Conexión:**
   Se establece la conexión con la base de datos usando `mysqli_connect()` y se comprueba si fue exitosa. En caso de error, se muestra el mensaje de error con `mysqli_connect_error()` y se detiene la ejecución.
2. **Comprobación de existencia:**
   - Se define un valor para la clave primaria, en este ejemplo `$cliente_id = 123`.
   - Se ejecuta una consulta SELECT para buscar registros en la tabla `clientes` con ese `cliente_id`.
   - Se utiliza `mysqli_num_rows()` para comprobar si la consulta devolvió alguna fila. Si el número de filas es mayor a 0, significa que el registro ya existe y se imprime un mensaje.
3. **Inserción condicional:**
   - Si la consulta de comprobación no devuelve resultados, se procede a insertar el registro utilizando `mysqli_query()` con una consulta INSERT.
   - Se comprueba el resultado de la inserción y se muestra un mensaje en función de si se realizó con éxito o se produjo algún error.
4. **Cierre de conexión:**
   Finalmente, se cierra la conexión a la base de datos con `mysqli_close()`.

Este enfoque permite evitar duplicidades en la tabla, comprobando previamente la existencia del registro mediante su campo clave antes de realizar la inserción.

Eliminar registros

A continuación se muestra un ejemplo en estilo procedimental con **mysqli** para borrar un registro de la base de datos, comprobando primero si existe:

```
// Suponiendo que queremos borrar un registro en la tabla 'clientes'
por su clave primaria (cliente_id)
$cliente_id = 123;

// Primero, comprobamos si existe el registro
$sql_check = "SELECT cliente_id FROM clientes WHERE cliente_id =
$cliente_id";
$result_check = mysqli_query($conexion, $sql_check);

if (!$result_check) {
    die("Error en la consulta: " . mysqli_error($conexion));
}

if (mysqli_num_rows($result_check) > 0) {
    // El registro existe, se procede a borrarlo
    $sql_delete = "DELETE FROM clientes WHERE cliente_id =
$cliente_id";
    $result_delete = mysqli_query($conexion, $sql_delete);

    if (!$result_delete) {
        echo "Error al borrar el registro: " .
mysqli_error($conexion);
    } else {
        echo "Registro con cliente_id = $cliente_id borrado
exitosamente.<br>";
    }
} else {
    echo "No se encontró registro con cliente_id =
$cliente_id.<br>";
}

// Cierra la conexión
mysqli_close($conexion);
```

Explicación

1. **Comprobación de existencia:**
   Se realiza una consulta SELECT para verificar si existe un registro en la tabla `clientes` con el `cliente_id` especificado.
   - Si la consulta retorna filas (usando `mysqli_num_rows()`), significa que el registro existe.

2. **Borrado del registro:**
   Si el registro existe, se ejecuta la consulta DELETE para borrar el registro.
   Se comprueba si la consulta DELETE se ejecutó correctamente y se muestra el mensaje correspondiente.
3. **Mensaje si no existe:**
   Si no se encuentra el registro, se muestra un mensaje indicando que no se encontró ningún registro con ese `cliente_id`.
4. **Cierre de conexión:**
   Se cierra la conexión con `mysqli_close()`.

Este ejemplo permite verificar de forma segura la existencia de un registro antes de intentar borrarlo, evitando errores y operaciones innecesarias.

Lectura de datos / consulta de registros

Este ejemplo recupera todos los registros de una tabla (por ejemplo, `clientes`) y los muestra en una lista HTML.

```
// Consulta simple: obtener todos
// los registros de la tabla 'clientes'

$sql = "SELECT * FROM clientes";
$result = mysqli_query($conexion, $sql);

if (!$result) {
    die("Error en la consulta: " . mysqli_error($conexion));
}

echo "<h2>Listado de clientes</h2>";

if (mysqli_num_rows($result) > 0) {      // ¿ HAY REGISTROS ?
    echo "<ul>";
    while ($row = mysqli_fetch_assoc($result)) {
        // Suponiendo que la tabla tiene las columnas:
        // cliente_id, nombre y email
        echo "<li>ID: " . $row['cliente_id'] . " - Nombre: " .
$row['nombre'] . " - Email: " . $row['email'] . "</li>";
    }
    echo "</ul>";
} else {
    echo "No se encontraron registros en la tabla clientes.";
}
```

Explicación

Se ejecuta una consulta SELECT para obtener todos los registros y se recorren con `mysqli_fetch_assoc()`, mostrando los resultados en una lista HTML.

Se utiliza un bucle WHILE para obtener los datos de cada fila. No devuelve un bloque de registros completo. Devuelve una sola fila.
La fila es un vector asociativo del que se pueden recuperar los datos utilizando su clave nombre_de_campo:

```
$row['cliente_id']
```

Si queremos obtener todos los datos en un array para después recorrerlo con FOR EACH debemos utilizar otra función para obtener los datos a partir de la consulta:

Puedes utilizar `mysqli_fetch_all()` con el modo `MYSQLI_ASSOC` para obtener un array asociativo de todas las filas, y luego iterar con `foreach`:

```php
// Consulta simple: obtener todos
// los registros de la tabla 'clientes'

$sql = "SELECT * FROM clientes";
$result = mysqli_query($conexion, $sql);

if (!$result) {
    die("Error en la consulta: " . mysqli_error($conexion));
}

echo "<h2>Listado de clientes</h2>";

$resultado = mysqli_query($conexion, $sql);
$filas = mysqli_fetch_all($resultado, MYSQLI_ASSOC);
foreach ($filas as $fila) {
    echo "Nombre: " . $fila['nombre'] . "<br>";
}
```

IMPORTANTE: Esta técnica es útil cuando deseas trabajar con todos los registros a la vez, y es especialmente conveniente cuando **el conjunto de resultados es pequeño o moderado**.

Consulta que devuelve un único valor

Aquí tienes un ejemplo en estilo procedimental con **mysqli** para obtener el valor del promedio del importe de los pedidos. Se asume que en la tabla `pedidos` existe una columna llamada `importe`:

```php
// Consulta para calcular el promedio del importe de pedidos
$sql = "SELECT AVG(importe) AS promedio FROM pedidos";
$resultado = mysqli_query($conexion, $sql);
if (!$resultado) {
    die("Error en la consulta: " . mysqli_error($conexion));
}

// Obtener el resultado
$fila = mysqli_fetch_assoc($resultado);
$promedio = $fila['promedio'];

// Mostrar el valor del promedio
echo "El promedio del importe de pedidos es: " . $promedio;
```

Explicación

- **Consulta SQL:**
  Se utiliza la función SQL `AVG(importe)` para calcular el promedio de la columna `importe` de la tabla `pedidos`, y se le asigna un alias `promedio`.
- **Ejecución y obtención del resultado:**
  La consulta se ejecuta con `mysqli_query()`. Luego, se usa `mysqli_fetch_assoc()` para obtener el resultado como un array asociativo (una única fila), de donde se extrae el valor del promedio mediante `$fila['promedio']`.
- **Salida:**
  Se imprime el promedio calculado.

Actualización de datos

Ejemplo en estilo procedimental con **mysqli** en el que se comprueba primero si existe el registro (en este caso, usando la clave primaria) antes de realizar la actualización. Recuerda que este ejemplo utiliza concatenación directa en la consulta, lo que no es seguro para datos provenientes de entradas de usuario, pero es válido para fines educativos.

```
// Datos para la actualización
$cliente_id = 123; // Clave primaria del registro a actualizar
$nuevoEmail = "nuevoemail@example.com"; // Nuevo email a asignar

// Comprobar si existe el registro con ese cliente_id
$sql_check = "SELECT cliente_id FROM clientes WHERE cliente_id =
$cliente_id";
$result_check = mysqli_query($conexion, $sql_check);
if (!$result_check) {
    die("Error en la consulta de comprobación: " .
mysqli_error($conexion));
}

if (mysqli_num_rows($result_check) > 0) {
    // El registro existe, se procede a actualizar
    $sql_update = "UPDATE clientes SET email = '$nuevoEmail' WHERE
                   cliente_id = $cliente_id";
    $result_update = mysqli_query($conexion, $sql_update);

    if (!$result_update) {
        echo "Error al actualizar el registro: " .
             mysqli_error($conexion) . "<br>";
    } else {
        if (mysqli_affected_rows($conexion) > 0) {
            echo "Registro actualizado exitosamente.<br>";
        } else {
            echo "No se actualizó el registro. Puede que el email s
                  ea el mismo.<br>";
        }
    }
} else {
    echo "No se encontró registro con cliente_id =
          $cliente_id.<br>";
}
```

Explicación

1. **Definición de datos:**
   Se define el valor de la clave primaria (`$cliente_id`) que se utilizará para identificar el registro, y el nuevo valor del email (`$nuevoEmail`).
2. **Comprobación de existencia:**
   Se realiza una consulta SELECT para verificar si existe algún registro en la tabla `clientes` con el `cliente_id` indicado.
   - Si la consulta devuelve filas, se asume que el registro existe y se procede a la actualización.
   - Si no se encuentra ningún registro, se informa que no se encontró el registro.
3. **Actualización del registro:**
   Si el registro existe, se construye y ejecuta la consulta UPDATE para cambiar el valor del email.
   - Se comprueba el resultado de la actualización y se muestra un mensaje adecuado.
   - Se utiliza `mysqli_affected_rows()` para determinar si la consulta modificó alguna fila (esto podría no suceder si el nuevo valor es el mismo que el actual).

Qué es la inyección SQL y cómo evitarla

La **inyección SQL** es una vulnerabilidad en las aplicaciones web que utilizan bases de datos, donde un atacante puede insertar o "inyectar" código SQL malicioso en las consultas, lo que le permite manipular la base de datos. Esto puede resultar en:

- Acceso no autorizado a datos.
- Creación, modificación o eliminación de registros.
- Ejecución de comandos que comprometan la integridad o confidencialidad de la información.

¿Cómo se produce?

Imagina que tienes un formulario de login donde el usuario introduce su nombre y contraseña. Si el código PHP construye una consulta SQL concatenando directamente esos valores, como:

```
$sql = "SELECT * FROM usuarios WHERE nombre = '" . $_POST['nombre']
. "' AND contrasena = '" . $_POST['contrasena'] . "'";
```

Un atacante podría introducir un valor malicioso en el campo "nombre", por ejemplo:

```
' OR '1'='1
```

Lo que generaría una consulta como:

```
SELECT * FROM usuarios WHERE nombre = '' OR '1'='1' AND contrasena =
'...'
```

Esta consulta podría devolver todos los registros y permitir el acceso sin credenciales válidas.

¿Cómo evitar la inyección SQL?

La forma más eficaz es **utilizar consultas preparadas (prepared statements)** que separen el código SQL de los datos. Tanto **mysqli** como **PDO** permiten utilizar este mecanismo. Otros métodos complementarios incluyen:

- Escapar correctamente los datos mediante funciones como `mysqli_real_escape_string()`, aunque esto es menos seguro que usar consultas preparadas.
- Validar y sanitizar todas las entradas del usuario.

Ejemplo con MySQLi y consultas preparadas

A continuación, un ejemplo de cómo evitar la inyección SQL utilizando **mysqli** en estilo procedimental:

```
<?php
// Configuración de conexión
$host = "localhost";
$usuario = "tu_usuario";
$contrasena = "tu_contrasena";
$nombreBD = "nombre_de_la_base";

$conexion = mysqli_connect($host, $usuario, $contrasena, $nombreBD);
if (!$conexion) {
    die("Error de conexión: " . mysqli_connect_error());
}

// Supongamos que recibimos 'nombre' y 'contrasena' del usuario a
través de POST
$nombre = $_POST['nombre'];
$contrasena_usuario = $_POST['contrasena'];

// Preparar la consulta para evitar inyección SQL
$stmt = mysqli_prepare($conexion, "SELECT * FROM usuarios WHERE
nombre = ? AND contrasena = ?");
if (!$stmt) {
    die("Error en la preparación de la consulta: " .
mysqli_error($conexion));
}

// Vincular los parámetros (ambos son cadenas)
mysqli_stmt_bind_param($stmt, "ss", $nombre, $contrasena_usuario);
mysqli_stmt_execute($stmt);

// Obtener los resultados
$resultado = mysqli_stmt_get_result($stmt);

if (mysqli_num_rows($resultado) > 0) {
    echo "Acceso concedido.";
} else {
    echo "Acceso denegado.";
}
```

```
mysqli_stmt_close($stmt);
mysqli_close($conexion);
?>
```

Modelo-Vista-Controlador

El patrón MVC (Modelo-Vista-Controlador) se utiliza para separar la lógica de la aplicación en tres componentes:

- **Modelo:** Se encarga de la interacción con la base de datos y de la lógica de negocio. Aquí se encuentran todas las operaciones CRUD (consultar, insertar, actualizar, borrar).
- **Vista:** Se encarga de la presentación de los datos. Genera el HTML (u otro formato) que se envía al usuario.
- **Controlador:** Actúa de intermediario, recibiendo las solicitudes del usuario, invocando al Modelo para obtener o modificar datos y luego llamando a la Vista para presentar la información.

A continuación, se muestra cómo se puede aplicar el patrón MVC a las operaciones de bases de datos en PHP (utilizando mysqli y estilo procedimental). La idea es organizar el código en archivos separados:

1. Modelo

Crea un archivo, por ejemplo, `ClienteModel.php`, que contenga funciones para conectar a la base de datos y realizar operaciones.

```
<?php
// ClienteModel.php

function conectarDB() {
    $host = "localhost";
    $usuario = "tu_usuario";
    $contrasena = "tu_contrasena";
    $nombreBD = "nombre_de_la_base";

    $conexion = mysqli_connect($host, $usuario, $contrasena, $nombreBD);
    if (!$conexion) {
        die("Error de conexión: " . mysqli_connect_error());
    }
    mysqli_set_charset($conexion, "utf8");
    return $conexion;
}

function obtenerClientes() {
    $conexion = conectarDB();
    $sql = "SELECT * FROM clientes";
    $resultado = mysqli_query($conexion, $sql);
    $clientes = mysqli_fetch_all($resultado, MYSQLI_ASSOC);
    mysqli_close($conexion);
```

```
    return $clientes;
}

function insertarCliente($cliente_id, $nombre, $email) {
    $conexion = conectarDB();
    $sql = "INSERT INTO clientes (cliente_id, nombre, email) VALUES
($cliente_id, '$nombre', '$email')";
    $resultado = mysqli_query($conexion, $sql);
    mysqli_close($conexion);
    return $resultado;
}

// Otras funciones para actualizar o borrar registros pueden definirse
aquí...
```

2. Controlador

Crea un archivo, por ejemplo, `ClienteController.php`, que coordine la interacción entre el Modelo y la Vista.

```
<?php
// ClienteController.php

require_once 'ClienteModel.php';

function listarClientes() {
    // Llama al modelo para obtener los datos
    $clientes = obtenerClientes();
    // Luego invoca la vista para mostrar los datos
    require 'ClientesView.php';
}

function crearCliente() {
// Supongamos que los datos vienen de un formulario (usando $_POST)
    $cliente_id = $_POST['cliente_id'];
    $nombre = $_POST['nombre'];
    $email = $_POST['email'];

    // Comprobar si el registro existe antes de insertar (opcional)
    // Por simplicidad, llamamos directamente a la función de inserción
    $resultado = insertarCliente($cliente_id, $nombre, $email);

    if ($resultado) {
        $mensaje = "Cliente insertado exitosamente.";
    } else {
```

```
        $mensaje = "Error al insertar el cliente.";
    }
    // En este caso, podríamos cargar una vista que muestre el mensaje
    require 'MensajeView.php';
}
```

3. Vista

Crea un archivo, por ejemplo, `ClientesView.php`, que se encargue de presentar la información.

```
<!-- ClientesView.php -->
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Listado de clientes</title>
</head>
<body>
    <h1>Listado de Clientes</h1>
    <?php if (!empty($clientes)): ?>
        <ul>
            <?php foreach ($clientes as $cliente): ?>
                <li>ID: <?= htmlspecialchars($cliente['cliente_id']) ?> -
                    Nombre: <?= htmlspecialchars($cliente['nombre']) ?> -
                    Email: <?= htmlspecialchars($cliente['email']) ?>
                </li>
            <?php endforeach; ?>
        </ul>
    <?php else: ?>
        <p>No hay clientes registrados.</p>
    <?php endif; ?>
</body>
</html>
```

O bien, un archivo `MensajeView.php` para mostrar mensajes:

```
<!-- MensajeView.php -->
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Resultado</title>
```

```
</head>
<body>
    <h1><?= htmlspecialchars($mensaje) ?></h1>
    <p><a href="index.php">Volver al inicio</a></p>
</body>
</html>
```

Flujo general

1. **Solicitud del usuario:**
   El usuario accede a una URL, por ejemplo, `index.php?action=listarClientes`.
2. **Controlador:**
   El controlador (`ClienteController.php`) detecta la acción solicitada y llama a la función correspondiente (por ejemplo, `listarClientes()`).
3. **Modelo:**
   La función en el modelo (`obtenerClientes()`) se encarga de conectar a la base de datos, ejecutar la consulta y devolver los resultados.
4. **Vista:**
   La vista (`ClientesView.php`) recibe los datos (en este caso, la variable `$clientes`) y genera la presentación en HTML para el usuario.

Este es un ejemplo básico de cómo aplicar el modelo MVC en PHP para operaciones de bases de datos. Cada capa se encarga de una responsabilidad específica, lo que facilita el mantenimiento y escalabilidad de la aplicación.

Referencia rápida de funciones para la gestión básica de bases de datos

Consulta la referencia oficial en:

https://www.php.net/manual/es/ref.mysql.php

1. mysqli_connect

Descripción: Establece una conexión al servidor MySQL.

Sintaxis:

```
mysqli_connect(string $host, string $username, string $password,
string $database, int $port = ?, string $socket = ?)
```

2. mysqli_connect_error

Descripción: Retorna el mensaje de error de la última conexión fallida.

Sintaxis:

```
mysqli_connect_error(): string
```

3. mysqli_set_charset

Descripción: Establece el juego de caracteres predeterminado para la conexión.

Sintaxis:

```
mysqli_set_charset(mysqli $connection, string $charset): bool
```

4. mysqli_query

Descripción: Ejecuta una consulta SQL en la base de datos.

Sintaxis:

```
mysqli_query(mysqli $connection, string $query)
```

5. mysqli_fetch_assoc

Descripción: Obtiene una fila de resultados como un array asociativo.

Sintaxis:

```
mysqli_fetch_assoc(mysqli_result $result): ?array
```

6. mysqli_fetch_all

Descripción: Obtiene todas las filas de un conjunto de resultados como un array.

Sintaxis:

```
mysqli_fetch_all(mysqli_result $result, int $resulttype =
MYSQLI_NUM): array
```

7. mysqli_prepare

Descripción: Prepara una consulta SQL para su ejecución.

Sintaxis:

```
mysqli_prepare(mysqli $connection, string $query): mysqli_stmt|false
```

8. mysqli_stmt_bind_param

Descripción: Asocia variables a los parámetros de una sentencia preparada.

Sintaxis:

```
mysqli_stmt_bind_param(mysqli_stmt $stmt, string $types, mixed
&$var1, mixed &...$vars): bool
```

9. mysqli_stmt_execute

Descripción: Ejecuta una sentencia preparada.

Sintaxis:

```
mysqli_stmt_execute(mysqli_stmt $stmt): bool
```

10. mysqli_stmt_get_result

Descripción: Recupera el conjunto de resultados de una sentencia preparada.

Sintaxis:

```
mysqli_stmt_get_result(mysqli_stmt $stmt): mysqli_result|false
```

11. mysqli_stmt_close

Descripción: Cierra una sentencia preparada.

Sintaxis:

```
mysqli_stmt_close(mysqli_stmt $stmt): void
```

12. mysqli_close

Descripción: Cierra una conexión previamente establecida.

Sintaxis:

```
mysqli_close(mysqli $connection): bool
```

13. mysqli_insert_id

Descripción: Devuelve el identificador generado automáticamente en la última consulta de inserción.

Sintaxis:

```
mysqli_insert_id(mysqli $connection): int|string
```

14. mysqli_num_rows

Descripción: Devuelve el número de filas en un conjunto de resultados.

Sintaxis:

```
mysqli_num_rows(mysqli_result $result): int
```

15. mysqli_affected_rows

Descripción: Devuelve el número de filas afectadas por la última consulta ejecutada.

Sintaxis:

```
mysqli_affected_rows(mysqli $connection): int
```

16. mysqli_real_escape_string

Descripción: Escapa caracteres especiales en una cadena para utilizarla de forma segura en una consulta SQL.

Sintaxis:

```
mysqli_real_escape_string(mysqli $connection, string $string):
string
```

Objetivo de la Práctica

En esta práctica de BASE: Práctica 1, desarrollarás un sistema básico en **PHP** para capturar leads de usuarios interesados en el producto. Implementarás un formulario web (ya debes tenerlo hecho) que enviará los datos al servidor y generará una página de informe de salud para el usuario, agradeciendo al usuario por su interés en el producto que incluye los datos de entrada del usuario y el IMC calculado.

Datos que se piden al usuario:

- **Peso (en kg)**.
- **Altura (en cm)**.
- **Fecha de nacimiento**.
- **Email**.
- **Sexo**: Mujer, Mujer embarazada, Hombre.
- **Actividad física**: Movilidad casi nula, Movilidad muy reducida, Normal, Activa (1,5 a 2,5 h/sem.), Muy activa (>2,5 h/sem.), Deportista.
- **Mi objetivo**: Perder peso, Mejorar mi salud, Ganar peso/músculo.

Datos que calculan:

- **IMC**:

$$\text{IMC} = \frac{\text{Peso (kg)}}{(\text{Altura (m)})^2}$$

    - Altura en metros debe ser calculada a partir de la altura ingresada en centímetros:

$$\text{Altura (m)} = \text{Altura (cm)}/100.$$

    - Redondear el resultado a un decimal para una presentación clara.

Aquí lo nuevo:

El sistema, además de todo lo anterior, debe guardar todos los datos en una base de datos que debes crear y diseñar en MYSQL. Debes garantizar que admite caracteres especiales, como tildes.

El servidor y puerto debe ser localhost:3306, la base de datos debe llamarse leads, el usuario de la base de datos juandeherrera, la contraseña IESjdh2025*. Los nombres de los campos los dejo a tu elección.

Inserta (UTILIZANDO el formulario que has diseñado) 10 registros en la base de datos utilizando tu formulario, con los datos completos. Usa tildes para probar que se almacenan bien.

Diseña una página PHP de informe (informe.php) que nos dé información de lo siguiente:

Listado de **todos los datos** de la base de datos en una doble lista anidada de esta forma:

1. email1
    - **Peso (en kg)**: XXXXXX
    - **Altura (en cm)**: XXXXXX
    - **Fecha de nacimiento**: XXXXXX
    - **Sexo**: XXXXXX
    - **Actividad física**: XXXXXX
    - **Mi objetivo**: XXXXXX
2. email2
    - **Peso (en kg)**: XXXXXX
    - **Altura (en cm)**: XXXXXX
    - **Fecha de nacimiento**: XXXXXX
    - **Sexo**: XXXXXX
    - **Actividad física**: XXXXXX
    - **Mi objetivo**: XXXXXX

Porcentaje de hombres y mujeres

Promedio de IMC de hombres y mujeres

Número de mujeres embarazadas que hay

Promedio de edad de hombres y de mujeres

Qué se espera que hagáis en esta práctica

Aplicar todo lo que hemos visto en clase sobre el tratamiento de bases de datos.

Desempolvar SQL

Explicar comentando el código.

Entregables

En el aula virtual.

ZIP con:

Exportación de la base de datos en formato SQL
Las páginas PHP.
Un documento con una captura de pantalla de PHPMyAdmin de los 10 registros y sus datos completos. En el documento haz una captura de pantalla de la salida de la página de informe que se vea completa.