Acceso a la base de datos mediante PHP orientado a objetos

1. Conexión mediante PHP en modo orientado a objetos con MySQLi

```php
<?php
$usuarioBD = "root" ;
$passBD = "root" ;
$servidorBD = "127.0.0.1" ;
$nombreBD = "prueba" ;
// conexión al servidor de base de datos
$conn = new mysqli($servidorBD, $usuarioBD, $passBD, $nombreBD);

if ($conn->connect_error) {
 die("ERROR: No se puede conectar al servidor: " . $conn->connect_error);
}

echo 'Conectado a la base de datos.<br>';

// se cierra la conexión cuando terminamos
$conn->close();
```

2. Acceso a la base de datos mediante PHP con la librería PDO. (PHP Data Objects)

Para representar **la conexión** se usa un objeto de la clase PDO.

El constructor es:

public PDO::__construct (string $dsn , string $username = ? , string $password = ? , array $options = ?)

La conexión es un proceso sencillo de dos pasos:

- Conectar con el servidor de MySQL
- Solicitar la conexión a un base de datos específica.

Necesitamos enviar como parámetros la dirección del servidor en el que se encuentra ejecutándose MySQL, el usuario, la contraseña y la base de datos o esquema al que conectarse.

Ejemplo de conexión a la base de datos llamada empresa:

```php
// Si se puede establecer la conexión, se usará el nuevo objeto PDO para manejar
// la BD. Si no se puede conectar con la BD, el constructor lanza
// una excepción PDOException.

$cadena_conexion = 'mysql:dbname=empresa;host=127.0.0.1';
$usuario = 'root';
$clave = '';
try {
    $bd = new PDO($cadena_conexion, $usuario, $clave);
    echo "Conexión realizada con éxito<br>";
} catch (PDOException $e) {
    echo 'Error con la base de datos: ' . $e->getMessage();
}
```

Si la base de datos no se va a utilizar más durante nuestro proceso se recomienda cerrar la conexión.

Para cerrar la conexión, es necesario destruir el objeto asegurándose de que todas las referencias a él existentes sean eliminadas; esto se puede hacer asignando NULL a la variable que contiene el objeto. **Si no se realiza explícitamente, PHP cerrará automáticamente la conexión cuando el script finalice**.

Para ello, podemos poner al final del fichero: `$bd = null;`

Y destruimos todo su contenido con unset($bd);

Hay que recordar que los **recursos son limitados** y que si tenemos muchos usuarios simultáneos pueden surgir problemas. Por ejemplo, al haber alcanzado el número máximo de conexiones con el servidor.

Al cerrar la conexión de forma explícita aceleramos la liberación de recursos para que estén disponibles para otros usuarios.

3. Errores y su manejo

PDO puede utilizar las excepciones para gestionar los errores, por lo que cualquier cosa que hagamos con PDO podríamos encapsularla en un bloque **try/catch** para gestionar si produce algún error.

Podemos forzar PDO para que trabaje en cualquier de los tres modos siguientes:

- **PDO::ERRMODE_SILENT** -> Es el modo predeterminado. Aquí tendremos que chequear los errores usando **->errorCode()** y **->errorInfo()**.
- **PDO::ERRMODE_WARNING** -> Genera errores warning PHP pero permitiría la ejecución normal de la aplicación.
- **PDO::ERRMODE_EXCEPTION** -> Será la forma más utilizada en PDO. Dispara una excepción permitiéndonos gestionar el error de forma amigable.

Se puede, y se debe, especificar el modo de error estableciendo el atributo ***error mode***:

```
$bd->setAttribute(PDO::ATTRR_ERRMODE, PDO::ERRMODE_SILENT);
$bd->setAttribute(PDO::ATTRR_ERRMODE, PDO::ERRMODE_WARNING);
$bd->setAttribute(PDO::ATTRR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

Existen otros atributos en el manual de PHP que también debemos de tener en cuenta. Esto podría ser un ejemplo de su uso:

```
try {
    $db = new PDO("mysql:host=mihost;dbname= mi_base_de_datos;charset=utf8", Pedro, mipassw);
    $db->setAttribute(PDO::MYSQL_ATTR_USE_BUFFERED_QUERY, true);
    $db->setAttribute(PDO::NULL_TO_STRING, true);
   } catch (PDOException $e) {
    die ("<p>No se ha podido establecer la conexión.
    <P>Compruebe si está activado el servidor de bases de
    datos MySQL.</p>\n <p>Error: " . $e->getMessage() .
    "</p>\n");
   }
```

Para evitar tener repetida la configuración del acceso a la base de datos se sugiere crear un archivo, donde declaremos los parámetros de la conexión como constantes, así las tendremos disponibles en toda la aplicación. Incluso podemos crear la clase BaseDatos con dicho código, añadiendo un método para realizar consultas y otro para extraer registros de la base de datos.

4. Consultas

Existen diferentes formas de generar consultas desde MySQL, pero la más usada es el método query().
El método query($cad) de la clase PDO recibe una instrucción SQL válida. Devuelve FALSE si hubo algún error o un objeto PDOStatement si se ejecutó con éxito.
Si se trata de una consulta, es posible recorrer las filas devueltas con un foreach. En cada iteración del bucle se tendrá una fila, representada como un array en que las claves son los nombres que aparecen en la cláusula SELECT.

Para ejecutar la consulta SELECT si no tenemos parámetros en la consulta podremos usar ->**query**() del objeto PDO

```php
$cadena_conexion = 'mysql:dbname=empresa;host=127.0.0.1';
$usuario = 'root';
$clave = '';

try {
    $bd = new PDO($cadena_conexion, $usuario, $clave);
    echo "Conexión realizada con éxito<br>";
    $sql = 'SELECT nombre, clave, rol FROM usuarios';
    $usuarios = $bd->query($sql);
    echo "Número de usuarios: " . $usuarios->rowCount() . "<br>";
    foreach ($usuarios as $usu) {
        print "Nombre : " . $usu['nombre']."<br>";
        print " Clave : " . $usu['clave'] . "<br>";
    }

    } catch (PDOException $e) {
        echo 'Error con la base de datos: ' . $e->getMessage();
    }
```

4.1. Consultas preparadas:

Una consulta preparada es una sentencia SQL precompilada que se puede ejecutar múltiples veces simplemente enviando datos al servidor. La sentencia puede ser la misma o similar.
Las consultas preparadas se usan como plantillas que podemos rellenar más adelante con valores reales que previamente son obtenidos desde formularios.
Con las consultas preparadas se mejora la seguridad y en el rendimiento de la aplicación. Por ejemplo, nos ayudará a evitar la inyección SQL.
Estas consultas se inicializan una sola vez con el método prepare() y se ejecutan las veces que sea necesario con execute() con diferentes valores para los parámetros.

Las sentencias preparadas se realizan en 3 pasos:

1. **Preparación:** Se envía una plantilla de la consulta al servidor, quien revisa la sintaxis y prepara los recursos necesarios. Algunos valores, llamados parámetros, se dejan sin especificar.

2. **Vinculación:** Se revisan los tipos de datos de cada parámetro. La base de datos analiza, compila y realiza la optimización de la consulta sobre la sentencia SQL, y guarda el resultado sin ejecutarlo.
3. **Ejecución:** La aplicación enlaza valores con los parámetros**,** y la base de datos ejecuta la sentencia. La aplicación puede entonces ejecutar la sentencia tantas veces como quiera con valores diferentes**.**

Para **construir una sentencia preparada** hay que hacerlo incluyendo unos **marcadores** en nuestra sentencia SQL.

Hay dos tipos de marcadores: por posición y por nombre.

Ejemplo de consulta preparada, con marcadores **anónimos** o de **posición**. Se usa el símbolo "?" para indicar un parámetro. Al ejecutar se asocian por orden los símbolos de interrogación con los valores del array que se pasa como argumento a execute()

```
$preparada = $bd->prepare("select nombre from usuarios where rol = ?")
$preparada->execute( array(0));
echo "Usuarios con rol 0: " .  $preparada->rowCount() . "<br>";
foreach ($preparada as $usu) {
    print "Nombre : " . $usu['nombre'] . "<br>";
}
```

Ejemplo de consulta preparada, parámetros **POR NOMBRE**. En este caso se usaran esos nombres como clave del array de execute().

```
$preparada_nombre = $bd->prepare("select nombre from usuarios where rol = :rol");
$preparada_nombre->execute( array(':rol' => 0));
echo "Usuarios con rol 0: " .  $preparada->rowCount() . "<br>";
foreach ($preparada_nombre  as $usu) {
    print "Nombre : " . $usu['nombre'] . "<br>";
}
```

Estas consultas preparadas también se pueden usar para insertar, modificar o borrar datos.

La elección de usar marcadores anónimos o conocidos afectará a cómo se asignan los datos a esos marcadores.

4.2. Cómo obtener los datos de la consulta:

Los datos que resultan de realizar consultas SELECT se obtienen a través del método fetch() o del método fetchAll().

- **->fetch()** : Obtiene la siguiente fila de un conjunto de resultados
- **->fetchAll()**: Devuelve un array que contiene todas las filas del conjunto de resultados (el tipo de datos a devolver se puede indicar como parámetro).

Antes de llamar a fetch (o durante) hay que especificar como se quieren devolver los datos:

- **PDO::FETCH_ASSOC**: devuelve un array indexado por el nombre de las columnas del conjunto de resultados de la tabla.
- **PDO::FETCH_NUM**: Devuelve un array indexado por el número de columna tal como fue devuelto en el conjunto de resultados, comenzando por la columna 0.
- **PDO::FETCH_BOTH**: valor por defecto. Devuelve un array indexado tanto por nombre de columna, como numéricamente con índice de base 0 tal como fue devuelto en el conjunto de resultados.
- **PDO::FETCH_BOUND**: Devuelve TRUE y asigna los valores de las columnas del conjunto de resultados a las variables de PHP a las que fueron vinculadas con el con el método PDOStatement::bindColumn.
- **PDO::FETCH_CLASS**: Devuelve una nueva instancia de la clase solicitada, haciendo corresponder las columnas del conjunto de resultados con los nombres de las propiedades de la clase. Creará las propiedades si éstas no existen.
- **PDO::FETCH_INTO**: Actualiza una instancia existente de la clase solicitada, haciendo coincidir el nombre de las columnas con los nombres de las propiedades de la clase..
- **PDO::FETCH_OBJ**: devuelve un objeto anónimo con nombres de propiedades que corresponden a las columnas.
- **PDO::FETCH_LAZY**: combina **PDO::FETCH_BOTH** y **PDO::FETCH_OBJ**, creando los nombres de las propiedades del objeto tal como se accedieron.

En el manual de PHP puedes consultar ejemplos sobre:

- fetch
- fetchAll

5. Insertar datos

Para insertar, borrar o actualizar simplemente hay que ejecutar la sentencia SQL, correspondiente. Puede ser una sentencia preparada o no.
Cuando se **obtienen, insertan o actualizan datos** con PDO se suele hacer en un proceso de 2 pasos, el esquema es: **PREPARE -> [BIND] -> EXECUTE**.

Ejemplo, sin sentencia preparada:

```php
$ins = "insert into usuarios(nombre, clave, rol) values('Susana', '44444', '2');";
$resul = $bd->query($ins);

    // errores
if($resul) {
    echo "Se ha añadido un nuevo usuario correctamente <br>";
    echo "Filas insertadas: " . $resul->rowCount() . "<br>";
}else print_r( $bd -> errorinfo());
// para los autoincrementos
echo "Código de la fila insertada " . $bd->lastInsertId() . "<br>";
```

Ejemplos con sentencias preparadas:

- Con **marcadores anónimos**

Para **vincular** los marcadores anónimos con su correspondiente valor se puede utilizar **bindParam** o **bindValue**.

```php
$ins = $bd->prepare("INSERT INTO usuarios(nombre, clave, rol) values (?, ?, ?)");
# Asignamos variables a cada marcador, indexados del 1 al 3
$ins->bindParam(1, $nombre);
$ins->bindParam(2, $passw);
$ins->bindParam(3, $rol);

# Insertamos una fila.
$nombre = "Daniel";
$passw = "777777";
$rol= "1";
$ins->execute();
```

- Con **marcadores** conocidos o **por nombre**. Es la forma más recomendable de trabajar con PDO, ya que a la hora de hacer el bindParam o el bindValue se puede especificar el tipo de datos y la longitud máxima de los mismos.

```php
$stmt = $bd->prepare("INSERT INTO usuarios(nombre, clave, rol) VALUES (:nombre, :clave, :rol)");
// asignamos los marcadores

$stmt->bindParam(':nombre', $nombre);
$stmt->bindParam(':clave', $clave);
$stmt->bindParam(':rol', $rol);
 // insertamos un usuario nuevo
$nombre = "Carmen";
$clave = "25252525";
$rol = "1";

$stmt->execute();
```

Nunca uses la sentencia prepare sin marcadores porque permitiría realizar inyección SQL. Por ejemplo, NO LA USES ASÍ:

```php
$ins = $bd->prepare("INSERT INTO usuarios (nombre, clave, rol) values ($nombre, $clave, $rol)");
```

Como acabamos de ver, existen dos métodos para enlazar valores: **bindParam()** y **bindValue()**

Diferencia entre el uso de bindParam y bindValue:

- Con bindParam() la variable es enlazada como una referencia y sólo será evaluada cuando se llame a execute(), es decir, entonces se asigna realmente el valor de la variable a ese parámetro.

- Con bindValue se asigna el valor de la variable a ese parámetro justo en el momento de ejecutar la instrucción bindValue. Es decir, se enlaza el valor de la variable y permanece hasta execute().

En la práctica bindValue() se suele usar cuando se tienen que insertar datos sólo una vez, y bindParam() cuando se tienen que pasar datos múltiples (desde un array por ejemplo).

Ambas funciones aceptan un tercer parámetro, que define el tipo de dato que se espera. Los tipos de datos más utilizados son: PDO::PARAM_BOOL (booleano), PDO::PARAM_NULL (null), PDO::PARAM_INT (integer) y PDO::PARAM_STR (string).

Por ejemplo:

```php
$stmt->bindParam(':edad', $miedad, PDO::PARAM_INT);
$stmt->bindParam(':apellidos', $misApellidos, PDO::PARAM_STR, 40);  // 40 caracteres como máximo.
```

Otra característica de los **marcadores conocidos** es que nos permitirán trabajar con **objetos** directamente en la base de datos, asumiendo que las propiedades de ese objeto coinciden con los nombres de los campos de la tabla en la base de datos.

6. Modificar datos

Ejemplo:

```php
//Ejemplo 1
// ACTUALIZAR sin sentencia preparada
echo 'Ejemplo 1:<br>';
$upd = "update usuarios set rol =  0 where rol = 1";
$resul = $bd->query($upd);
//comprobar errores
if($resul){
        echo "update correcto <br>";
        echo "Filas actualizadas: " .$resul->rowCount(). "<br>";
}else print_r( $bd -> errorinfo());

// Ejemplo 2 con sentencia preparada
echo 'Ejemplo con sentencia preparada <br>';
$codigo = 6;
$rol = "1";
$stmt = $bd->prepare('UPDATE usuarios SET rol = :rol WHERE codigo = :codigo');
$stmt->execute([':codigo'   => $codigo,
  ':rol' => $rol
  ]);
echo "Filas actualizadas: " .$stmt->rowCount();
```

7. Borrar datos

Ejemplo:

```php
// Ejemplo 1: BORRAR
    echo 'Ejemplo 1 borramos el usuario Susana <br>';
$del = "delete from usuarios where nombre = 'Carmen'";
$resul = $bd->query($del);
//comprobar errores
if($resul){
    echo "delete correcto <br>";
    echo "Filas borradas: " . $resul->rowCount() . "<br>";
}else print_r( $bd -> errorinfo());

    // Ejemplo 2 borrar con sentencia preparada
    echo 'Ejemplo 2: borramos con una sentencia preparada el usuario Angelines <br>';
    $nombre = "Angelines";
    $stmt = $bd->prepare('DELETE FROM usuarios WHERE nombre = :nombre');
    $stmt->bindParam(':nombre', $nombre);
    $stmt->execute();
    echo "Filas borradas: " .$stmt->rowCount();
```

8. Otros métodos interesantes:

8.1. **lastInsertId():**Devuelve el id del último registro insertado en la tabla. Es un método de PDO
$pdo->lastInsertId();

8.2. **quote():** Si no usas consultas preparadas, la forma de protegerte contra la inyección SQL es usando este método:

```
$sqlSegura= $pdo->quote($sqlInsegura);
```

8.3. **rowCount():**Devuelve un entero indicando el número de filas afectadas por la última operación.
$rows_affected = $stmt->rowCount();

9. Transacciones

Una transacción consiste en un conjunto de operaciones que deben realizarse de forma atómica. Es decir, o se realizan todas o no se realiza ninguna. Por ejemplo, una transferencia de saldo entre dos clientes implica dos operaciones:

- Quitar saldo al cliente que envía la transferencia
- Aumentar el saldo del cliente que recibe la transferencia.

Si por cualquier motivo la segunda operación falla, hay que deshacer la primera operación para que el sistema quede en estado consistente. Las dos operaciones forman una única transacción.

Para indicar que comienza una transacción se usa el método PDO::beginTransaction(). La transacción se salva usando el método commit(). Con el método PDO::rollback() se deshacen las operaciones realizadas desde que se inició la transacción. Cuando este método es llamado, todas las acciones que estuvieran pendientes se activan y la conexión a la base de datos vuelve de nuevo a su estado por defecto que es auto-commit.

Ejemplo:

```php
// comenzar la transacción
$bd->beginTransaction();
$ins = "insert into usuarios(nombre, clave, rol) values('Fernando', '33333', '1')
$resul = $bd->query($ins);
// se repite la consulta
// falla porque el nombre es unique
$resul = $bd->query($ins);
if(!$resul){
    echo "Error: " . print_r($bd->errorinfo());
    // deshace el primer cambio
    $bd->rollback();
    echo "<br>Transacción anulada<br>";
}else{
    // si hubiera ido bien...
    $bd->commit();
}
```