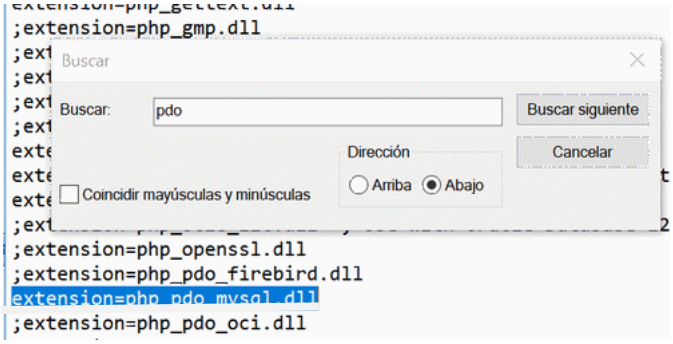
Conexión con la BD MySQL

Para poder acceder a MySQL mediante PDO, debe estar activada la extensión **php_pdo_mysql** en el archivo de configuración **php.ini**



Se va a estudiar de forma general la API y los métodos que proporciona y después se debe adaptar su utilización en cada caso

En el caso de MySQL, la creación de un objeto de la clase PDO incluye el nombre del servidor, el nombre de usuario y la contraseña.

```php
// Utilizando el objeto en una función de conexión con la BD MYSQL
function conectaDb()
{
    try {
        $db = new PDO("mysql:host=" . DB_SERVER . ";dbname=" .
                      DB_NAME, DB_USER, DB_PASS);
        // Se puede configurar el objeto
        $db->setAttribute(PDO::MYSQL_ATTR_USE_BUFFERED_QUERY, true);
        $db->exec("set names utf8mb4");
        return($db);

    } catch(PDOException $e) {
        echo "  <p>Error: " . $e->getMessage() . "</p>\n";
        exit();N

        //OTRA Opción podría ser enviar a otra página
        header('Location: vistas/error.php?error=ERROR');
        exit();

    }
}

// EJEMPLO DE USO DE LA FUNCIÓN ANTERIOR
// La conexión se debe realizar en cada página que acceda a la base
de datos
$db = conectaDB();
```

Desconexión con la base de datos

Para desconectar con la base de datos hay que destruir el objeto PDO. Si no se destruye el objeto PDO, PHP lo destruye al terminar la página.

```
$db = null;
```

Consultas a la base de datos

Una vez realizada la conexión a la base de datos, las operaciones se realizan a través de consultas.
El método general para efectuar consultas es `PDO->query($consulta)`, que devuelve el resultado de la consulta. Dependiendo del tipo de consulta, el dato devuelto debe tratarse de formas distintas.

- **Si es una consulta que no devuelve registros**, sino que simplemente realiza una acción que puede tener éxito o no (por ejemplo, insertar un registro), el método devuelve **`true`** o **`false`**. No es necesario guardar el resultado de la consulta en ninguna variable, pero se puede utilizar para sacar un mensaje diciendo que todo ha ido bien (o no). Por ejemplo,

```
// EJEMPLO DE CONSULTA DE INSERCIÓN DE REGISTRO
$db = conectaDB();

$consulta = "INSERT INTO $dbTabla
     (nombre, apellido)
     VALUES ("$nombre", "$apellido")";

if ($db->query($consulta)) {
     echo "  <p>Registro creado correctamente.</p>\n";
} else {
     echo "  <p>Error al crear el registro.<p>\n";
}

$db = null;
```

- **Pero si la consulta devuelve registros**, el método devuelve los registros correspondientes o **`false`**. En ese caso sí que es conveniente guardar lo que devuelve el método en una variable para procesarla posteriormente. Si contiene registros, la variable es de un tipo especial que no se puede acceder directamente, pero que se puede recorrer con un bucle **`foreach()`**

```php
// EJEMPLO DE CONSULTA DE SELECCIÓN DE REGISTROS
$db = conectaDB();
$consulta = "SELECT * FROM $dbTabla";
$result = $db->query($consulta);

if (!$result) { // o con try-catch
    echo "  <p>Error en la consulta.</p>\n";
} else {
    foreach ($result as $valor) {
        echo "  <p>$valor[nombre] $valor[apellido]</p>\n";
    }
}
$db = null;
```

En los ejemplos, se define una variable `$consulta` que contiene la consulta y a continuación se ejecuta la consulta, pero podría estar en una sola:

```php
// En dos líneas
$consulta = "SELECT * FROM $dbTabla";
$result = $db->query($consulta);

// En una sola línea
$result = $db->query("SELECT * FROM $dbTabla");
```

Se recomienda utilizar la primera versión, que permite visualizar la consulta mientras se está programando para comprobar que no tiene errores, también con el depurador.

Seguridad en las consultas: consultas preparadas

Para evitar ataques de inyección SQL (en la lección Inyecciones SQL se comentan los ataques más elementales), se recomienda el uso de sentencias preparadas, en las que PHP se encarga de "desinfectar" los datos en caso necesario. En general, cualquier consulta que incluya datos introducidos por el usuario debe realizarse mediante consultas preparadas.

El método para efectuar consultas es primero preparar la consulta con PDO->prepare($consulta) y después ejecutarla con PDO->execute(array(parámetros)), que devuelve el resultado de la consulta.

```php
// Base de la Consulta preparada
$consulta = "SELECT * FROM $dbTabla";
$result = $db->prepare($consulta);
$result->execute();
```

Dependiendo del tipo de consulta, el dato devuelto debe tratarse de formas distintas, como se ha explicado en el apartado anterior.

Si la consulta incluye datos introducidos por el usuario, los datos pueden incluirse directamente en la consulta, pero en ese caso, PHP no realiza ninguna "desinfección" de los datos, por lo que estaríamos corriendo riesgos de ataques:

```
$nombre    = $_POST["nombre"];
$apellido = $_POST["apellido"];

$consulta = "SELECT COUNT(*) FROM $dbTabla
    WHERE nombre=$nombre
    AND apellido=$apellido";

 // DESACONSEJADO: PHP NO DESINFECTA LOS DATOS
$result = $db->prepare($consulta);
$result->execute();
if (!$result) {  // o con try-catch
    echo "  <p>Error en la consulta.</p>\n";
 ...
```

Para que PHP desinfecte los datos, **estos deben enviarse al ejecutar la consulta, no al prepararla**. Para ello es necesario indicar en la consulta la posición de los datos. Esto se puede hacer de dos maneras, mediante parámetros o mediante interrogantes, aunque se aconseja la utilización de parámetros:

1. mediante interrogantes (?)

```
$consulta = "SELECT COUNT(*) FROM $dbTabla
    WHERE nombre=? AND apellido=?";
$result = $db->prepare($consulta);
$args = array($nombre,$apellido);
$result->execute($args);
if (!$result) {  // o con try-catch
    echo "  <p>Error en la consulta.</p>\n";
...
```

2. mediante parámetros (**:**parametro)

   En este caso el array debe incluir los nombres de los parámetros y los valores que sustituyen a los parámetros (el orden no es importante al estar identificados), como muestra el siguiente ejemplo:

```
$nombre    = $_POST["nombre"];
$apellido = $_POST["apellido"];

$consulta = "SELECT COUNT(*) FROM $dbTabla
    WHERE nombre=:nombre
    AND apellido=:apellido";
$result = $db->prepare($consulta);
$result->execute(array(":nombre" => $nombre, ":apellido" =>
$apellido));
if (!$result) { // o con try-catch
    echo "  <p>Error en la consulta.</p>\n";
 ...
```

3. mediante parámetros con el método bindParam

```
$nombre    = $_POST["nombre"];
$apellido = $_POST["apellido"];

$consulta = "SELECT COUNT(*) FROM $dbTabla
    WHERE nombre=:nombre
    AND apellido=:apellido";
$result = $db->prepare($consulta);
$result->bindParam(':nombre',$nombre,PDO::PARAM_STR,50 );
$result->bindParam(':apellido',$apellido,PDO::PARAM_STR,50 );
$result->execute();
if (!$result) { // o con try-catch
    echo "  <p>Error en la consulta.</p>\n";
 ...
```

Utilizar en los ejercicios preferentemente uno de estos tres métodos, el último es el más recomendable.

Restricciones en los parámetros de consultas preparadas

Debido a que las consultas preparadas se idearon para optimizar el rendimiento de las consultas, el uso de parámetros tiene algunas restricciones. Por ejemplo

- los identificadores (nombres de tablas, nombres de columnas, etc) no pueden sustituirse por parámetros.
- los dos elementos de una igualdad no pueden sustituirse por parámetros
- en general no pueden utilizarse parámetros en las consultas DDL (lenguaje de definición de datos) (nombre y tamaño de los campos, etc.)

Si no podemos usar parámetros, no queda más remedio que incluir los datos en la consulta. Como en ese caso PHP no hace ninguna desinfección de los datos, la tenemos que hacer nosotros previamente.

Como en estos casos los valores introducidos por el usuario suelen tener unos valores restringidos (por ejemplo, si el usuario puede elegir una columna de una tabla, los nombres de las columnas están determinadas y el usuario sólo puede elegir uno de ellos). Podemos crear una función de recogida de datos específica que impida cualquier tipo de ataque de inyección por parte del usuario, como muestra el siguiente ejemplo….*(Bartolomé Sintes) leer más>*