

Práctica 4



Sesiones 5, 6, 7 y 8

(Continuación)

Acceso a la base de datos MySQL desde PHP

Existen dos filosofías de acceso a MySQL desde PHP:

- `mysqli` (MySQL improved): extensión que ofrece una dualidad procedural y orientada a objetos, por medio de un conjunto de funciones y de clases e interfaces (<https://www.php.net/manual/es/book.mysqli.php>).
- `PDO` (PHP Data Object): extensión que ofrece un conjunto de clases y métodos (<https://es1.php.net/manual/es/book.pdo.php>).

Centrándonos en la extensión `PDO`, para realizar una conexión hay que crear un nuevo objeto `PDO`, pasándole como argumento del constructor la base de datos donde conectar, el nombre de usuario y la contraseña:

```
$dsn = "mysql:host=betatun.ugr.es;dbname=frutas";  
$usuario= "root";  
$password= "contrasenia";  
$conexion = new PDO( $dsn, $usuario, $password );
```

Aunque PHP cierra las conexiones cuando finaliza el guión, es una buena idea cerrarla explícitamente, para lo cual hay que asignarle el valor `null`: `$conexion=null`.

Para gestionar los errores que ofrece MySQL, se pueden emplear excepciones, para lo cual hay que utilizar el método `setAttribute` y seguidamente emplear los `try/catch` correspondientes, de la siguiente forma:

```
try {  
    $conexion = new PDO( $dsn, $usuario, $password );  
    $conexion->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );  
}
```

```
} catch ( PDOException $e ) { echo "Conexión fallida: " . $e->getMessage(); }
```

Una vez conectados a la base de datos correspondiente, se puede enviar una consulta mediante la sentencia SELECT de SQL:

```
$consultaSQL = "SELECT * FROM frutas";  
$resultados = $conexion->query( $consultaSQL );  
foreach ( $resultados as $fila ) {  
    echo "Nombre = " . $fila["nombre"] . "<br />";  
    echo "color = " . $fila["color"] . "<br />";  
}
```

Como se puede observar en el ejemplo, el resultado de la consulta llega en forma de array multidimensional, en el que cada fila (fila de la tabla) es a su vez otro array que contiene como claves los nombres de los atributos y sus valores correspondientes.

Para realizar una inserción en una tabla, hay que crear una cadena de caracteres conteniendo la sentencia INSERT de MySQL y luego ejecutar el método query del objeto que representa la conexión:

```
$consultaSQL = "INSERT INTO frutas VALUES ( 'chirimoyo', 'verde' );"  
try { $conexion->query( $consultaSQL ); }  
catch ( PDOException $e ) { echo "Consulta fallida: " . $e->getMessage(); }
```

De igual forma se pueden borrar registros empleando la sentencia DELETE de MySQL ó UPDATE, para modificar un registro de una base de datos.

Cuando hay que ejecutar una consulta varias veces, quizá es algo pesado o tedioso tener que prepararla empleando variables PHP o atributos de campos. Para aliviar esta tarea se emplean los denominados marcadores de parámetros. De esta forma, la consulta se crea de forma genérica una única vez y luego se le asignan los valores a dichos marcadores. Veamos el siguiente ejemplo:

```
// Se crea la consulta genérica colocando marcadores de parámetros.  
$consultaSQL = "SELECT * FROM frutas WHERE color = :color";  
// ó $consultaSQL = "UPDATE frutas SET color = :color WHERE nombre = :nombre";  
// Se obtiene un objeto de la clase PDOStatement, mediante el método prepare  
// del objeto conexión.  
$sentenciaSQL= $conexion->prepare($consultaSQL);  
try {  
    // Seguidamente se asignan valores a los marcadores de parámetros:  
    $sentenciaSQL->bindValue( ":color", $color, PDO::PARAM_STR );  
    // Y se ejecuta la consulta:
```

```
$sentenciaSQL->execute();

// Para recorrer los registros recuperados se emplea el método fetchAll:

$frutas = array();

foreach ( $sentenciaSQL->fetchAll() as $fruta ) {

    $frutas[] = new Fruta( $fruta); //Asumiendo que existe una clase Fruta.

}

} catch (PDOException $e) {echo "Consulta fallida: " . $e->getMessage(); }
```

Como se aprecia, el tercer parámetro del método *bindValue* es el tipo de dato del atributo (la clase PDO ofrece los siguientes tipos: PDO::PARAM_BOOL, PDO::PARAM_NULL, PDO::PARAM_STR y PDO::PARAM_LOB).

Estructura general de un guión PHP para acceso a bases de datos

Lo habitual es crear un primer fichero llamado *configuracion.inc*, el cual contiene la declaración de las constantes necesarias para conectar a la base de datos con la que vamos a trabajar y acceder a las tablas correspondientes:

```
<?php

define("DB_DSN", "mysql:host=localhost;dbname=nombreBD" );

define("DB_USUARIO", "root" );

define("DB_CONTRASENIA", "contrasenia" );

define("TAMANIO_PAGINA", 5 );

define("TABLA_FRUTAS", "frutas" );

define("TABLA_PROVEEDORES", "proveedores" );

?>
```

Suele existir una clase abstracta que ofrezca las operaciones más habituales para gestión de la base de datos (*datosObject.class.inc*), como conectar a la misma o desconectarse, así como datos miembros que almacenen los campos de la tabla correspondiente (Ejemplo tomado y adaptado del libro Beginning PHP 5.3. Mat Doyle. Wrox. 2010):

```
<?php

require_once('configuracion.php');

abstract class DataObject {

    protected $datos = array();

    public function __construct( $datos ) {

        foreach ( $datos as $clave => $valor )

            if ( array_key_exists( $clave, $this->datos ) ) $this->datos[$clave] = $valor;
```

```

    }

    public function devolverValor( $campo ) {

        if ( array_key_exists( $campo, $this->datos ) ) {

            return $this->datos[$campo];

        } else die( "Campo no encontrado" );

    }

    protected static function conectar() {

        try {

            $conexion = new PDO( DB_DSN, DB_USUARIO, DB_CONTRASENIA );

            // Se permite a PHP que mantenga la conexión MySQL abierta para
            // que se emplee en otras partes de la aplicación.

            $conexion->setAttribute( PDO::ATTR_PERSISTENT, true );

            $conexion->setAttribute(PDO::ATTR_ERRMODE,
                                   PDO::ERRMODE_EXCEPTION );

        } catch ( PDOException $e ) {

            die( "Conexión fallida: " . $e->getMessage() );

        }

        return $conexion;

    }

    protected static function desconectar( $conexion ) {

        $conexion = "";

    }

}

?>

```

Una vez hecho esto, se crea una clase por tabla a la que se acceda, modelando objetos que coincidirán con los registros almacenados en las tablas. En nuestro caso, *fruta.class.inc*:

```

<?php

require_once ('datosObject.class.inc');

class Fruta extends DataObject {

    protected $datos = array(

        "id" => "",

        "nombre" => "",

```

```

        "color" => "",
        "tipo" => "");

    public static function obtenerFrutas( $filaInicio, $numeroFilas, $orden ) {

        $conexion = parent::conectar();

        $sql = "SELECT SQL_CALC_FOUND_ROWS * FROM " . TABLA_FRUTAS . "
                ORDER BY " . $orden . "LIMIT :filaInicio, :numeroFilas";

        try {

            $st = $conexion->prepare( $sql );

            $st->bindValue( ":filaInicio", $filaInicio, PDO::PARAM_INT );
            $st->bindValue( ":numeroFilas", $numeroFilas, PDO::PARAM_INT );

            $st->execute();

            $frutas = array();

            foreach ( $st->fetchAll() as $fila ) {

                $frutas[] = new Fruta( $fila );

            }

            $st = $conexion->query( "SELECT found_rows() AS filasTotales" );
            $fila = $st->fetch();

            parent::desconectar( $conexion );

            return array( $frutas, $fila["filasTotales"] );

        } catch ( PDOException $e ) {

            parent::desconectar( $conexion );

            die( "Consulta fallida: " . $e->getMessage() );

        }

    }

    public static function obtenerFruta( $id ) {

        $conexion = parent::conectar();

        $sql = "SELECT * FROM " . TABLA_FRUTAS . " WHERE id = :id";

        try {

            $st = $conexion->prepare( $sql );

            $st->bindValue( ":id", $id, PDO::PARAM_INT );

            $st->execute();

            $fila = $st->fetch();

```

```

        parent::desconectar( $conexion );
        if ( $fila ) return new Fruta( $fila );
    } catch ( PDOException $e ) {
        parent::desconectar( $conexion );
        die( "Consulta fallada: " . $e->getMessage() );
    }
}

public static function insertarFruta(...)
{ /* Código para insertar una fruta en la tabla. */}
// Aquí también se meterían métodos que procesaran los campos de alguna forma.
public function obtenerTipoCadena() {
    return ( $this->datos["tipo"] == "s" ) ? "Seco" : "Carnoso";
}
}

?>

```

Seguidamente podemos escribir otra clase que confeccione y muestre los formularios correspondientes para interactuar con el usuario, así como para mostrar la cabecera de la página HTML y la parte final de la misma. Esta incluiría los ficheros configuracion.inc y Frutas.class.inc.

Ejercicio:

Crea, desde la línea de mandatos de MySQL (mysql) o desde phpmyadmin una tabla Libros en tu base de datos. El código para dicha creación es el siguiente:

```

CREATE TABLE Libros (
    isbn VARCHAR(20) NOT NULL,
    titulo VARCHAR(50) NOT NULL,
    autor VARCHAR(50) NOT NULL,
    editorial VARCHAR(30) NOT NULL,
    numPaginas SMALLINT UNSIGNED NOT NULL,
    anio SMALLINT UNSIGNED NOT NULL,
    PRIMARY KEY (isbn) );

```

Escribe un guión PHP (o los que necesites) que permita acceder a la base de datos para insertar registros, hacer consultas simples y borrar registros, a través todo de formularios.

Gestión de ficheros y directorios con PHP

PHP permite trabajar con ficheros y directorios en el servidor. Esto es muy útil porque nos permite almacenar información fuera de los guiones. La gestión de éstos es muy parecida a la que hace el lenguaje C. Véase <https://php.net/manual/es/book.filesystem.php> para una referencia de las funciones que permiten trabajar con ficheros y directorios.