Acceso a BBDD en el lenguaje PHP

PDO (PHP Data Objects)

- Disponible desde PHP 5.1.
- Proporciona una **capa de abstracción** de acceso a datos.
- Independientemente de la base de datos que se esté utilizando, se emplean las mismas funciones para realizar consultas y obtener datos.
- Cada controlador de bases de datos puede implementar la interfaz PDO.
  - http://php.net/manual/es/pdo.drivers.php
- Anteriormente, se utilizaban extensiones específicas para cada sistema gestor de base de datos (extensiones nativas).
- Las funciones y objetos a utilizar eran distintos para cada extensión.

MariaDB

- Nosotros usaremos MariaDB (sucesor de MySQL).
- SGBD relacional con licencia GPL.
- El **más empleado** con el lenguaje PHP.
- El driver nativo de PHP para MariaDB se llama mysqli (sucesor del driver mysql).
- Nosotros usaremos PDO, pero el funcionamiento es muy similar.

Características de MariaDB

- Incorpora múltiples motores de almacenamiento.
- Elegiremos el motor en función de las características propias de la aplicación.
- Por defecto utiliza **MyISAM**:
  - Muy rápido.
  - No contempla integridad referencial ni tablas transaccionales.
- El motor **InnoDB es un poco más lento, pero sí soporta estas dos características**.
- Casi siempre usaremos InnoDB.
- Documentación:
  - https://mariadb.com/kb/en/library/documentation/

Instalación y configuración de MariaDB

- No es necesario que lo instalemos puesto que lo tenemos **integrado en nuestro LAMPP**.
- Se ejecuta por defecto en el puerto 3306.
- El fichero de configuración se llama /opt/lampp/etc/my.cnf
- Su contenido se divide en secciones.
- Las directivas relacionadas con el servidor están en la sección [mysqld].

Herramientas de administración independientes

- **MySQL Workbench**: Herramienta genérica con interface gráfico nativo que permite administrar tanto el servidor como las bases de datos que éste gestiona.
  - http://dev.mysql.com/doc/workbench/en/index.html
- **PHPMyAdmin**: es una aplicación web muy popular para la administración de servidores MySQL.
  - http://www.phpmyadmin.net/

Instalación de PHPMyAdmin

- No se instala con el servidor MariaDB.
- Instalación en Debian:

  sudo apt-get install phpmyadmin
- Viene **integrada en XAMPP**, no necesitamos instalarla
- Documentación:
  - https://docs.phpmyadmin.net/en/latest/

Acceso a BBDD desde PHP

- Para acceder a BBDD MariaDB tenemos dos opciones:
  - El driver nativo mysqli
  - El driver genérico PDO
- Nosotros utilizaremos **PDO**, aunque el uso de mysqli es muy similar al de PDO.

Establecimiento de la conexión

- Hay que instanciar un objeto de la clase PDO pasándole los siguientes parámetros (sólo el primero es obligatorio):
  - **Origen de datos (DSN).** Cadena de texto que indica qué controlador se va a utilizar y, a continuación, separadas por el carácter dos puntos, los parámetros específicos necesarios por el controlador, como por ejemplo, el nombre o dirección IP del servidor y el nombre de la base de datos.
  - **Nombre de usuario** con permisos para establecer la conexión.
  - **Contraseña del usuario**.
  - **Opciones de conexión**, almacenadas en forma de array.

Cadena de conexión con coontrolador MariaDB

- Si utilizamos el controlador para MariaDB, la cadena DSN recibirá los parámetros siguientes (a continuación del prefijo mysql:):
  - **host**. Nombre o dirección IP del servidor
  - **port**. Número de puerto TCP en el que escucha el servidor
  - **dbname**. Nombre de la base de datos

Opciones de la conexión

En el array de opciones que le pasamos al crear la conexión podemos utilizar, entre otras, las siguientes:

- Para que extraiga los **datos** de la base de datos **en utf8**:
  - PDO::*MYSQL_ATTR_INIT_COMMAND* => "SET NAMES utf8"
- Para que devuelva los **errores en forma de excepciones**:
  - PDO::*ATTR_ERRMODE* => PDO::*ERRMODE_EXCEPTION*

Conexiones persistentes

- Las conexiones persistentes no se **cierran al final del script**.
- Son **almacenadas en caché y reutilizadas** cuando otro script solicite una conexión que use las mismas credenciales.
- La caché de conexiones persistentes permite evitar la carga adicional de establecer una nueva conexión cada vez que un script necesite comunicarse con la base de datos, dando como resultado una aplicación web más rápida.
- Para establecer una conexión persistente utilizaremos la siguiente opción al realizar la conexión:
  - PDO::*ATTR_PERSISTENT* => true

Ejemplo

```
$opciones = array(
        PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8",
        PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
        PDO::ATTR_PERSISTENT => true
);
$pdo = new PDO(
        'mysql:host=agenda.local;dbname=agenda;charset=utf8',
        'usuAgenda',
        'globalnet',
        $opciones);
```

- Más información:
  - http://es.php.net/manual/es/pdo.drivers.php

Obtener información de la conexión

- Con el método **getAttribute** obtenemos información del **estado de la conexión**.
- Ejemplo, obtener la versión del servidor:

```
$version = $dwes->getAttribute(PDO::ATTR_SERVER_VERSION);
echo "Versión: $version";
```

- Más información:
  - http://es.php.net/manual/es/pdo.getattribute.php

Establecer parámetros de configuración de la conexión

- Con el método **setAttribute** podemos **modificar algunos parámetros** que afectan a la misma.
- Ejemplo, que se devuelvan todos los nombres de columnas en mayúsculas:

  ```
  $version = $dwes->setAttribute(PDO::ATTR_CASE, PDO::CASE_UPPER);
  ```

- Para más información:
  - http://es.php.net/manual/es/pdo.setattribute.php

PDOException

- Todas las **excepciones** que genera PDO son **del tipo PDOException**.
- Hereda de la clase Exception.
- Si hubieran errores de conexión, se lanzará una excepción PDOException.
- Si no capturamos la excepción, se finalizará el script y **mostrará información de seguimiento**.
- Esto **podría revelar detalles de la conexión** a la base de datos, incluyendo el nombre de usuario y la contraseña.

Cierre de la conexión

- La conexión permanecerá **activa durante el tiempo de vida del objeto PDO**.
- Para cerrar la conexión es necesario destruir el objeto asegurándose de que todas las referencias a él 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**.
- Si la conexión es persistente se almacenará en la cache para posteriores conexiones.

Ejecución de consultas que no devuelven datos

- Consultas de acción: INSERT, DELETE o UPDATE.
- Se puede utiliza el método **PDO::exec**
- Ejecuta una sentencia SQL y devuelve el número de registros afectados.

```
$registros = $pdo->exec('DELETE FROM stock WHERE unidades=0');
echo "<p>Se han borrado $registros registros.</p>";
```

- Esta sentencia es **vulnerable a inyecciones SQL**.
- Para usarla tendremos que escapar la cadena de entrada con **PDO::quote**
- En su lugar se recomienda utilizar **consultas preparadas**.

Ejecución de consultas que devuelven datos

- Si la consulta genera un conjunto de datos (SELECT), se puede utilizar el método **query**.
- Ejecuta una sentencia SQL.
- Devuelve un conjunto de resultados como un objeto PDOStatement.

```
$resultado = $pdo->query("SELECT producto, unidades FROM stock");
```

- Esta sentencia es **vulnerable a inyecciones SQL**.
- Para usarla tendremos que escapar la cadena de entrada con **PDO::quote**
- En su lugar se recomienda utilizar **consultas preparadas**.

Obtención y utilización de conjuntos de resultados

- En PDO tenemos varias posibilidades para tratar el conjunto de resultados devuelto por el método query.
- La más utilizada es el método **fetch** de la clase PDOStatement
- Devuelve un **registro del conjunto de resultados**, o false si ya no quedan registros por recorrer.

```
$resultado = $pdo->query("SELECT producto, unidades FROM stock");
while ($registro = $resultado->fetch())
{
    echo "Producto ".$registro['producto'].": ";
    echo $registro['unidades']."<br />";
}
```

Consultas preparadas

Nos aportan **dos ventajas** importantes:

- Para sentencias que serán ejecutadas en múltiples ocasiones con diferentes parámetros **optimiza el rendimiento** de la aplicación.
- Ayuda a **prevenir inyecciones SQL** eliminando la necesidad de entrecomillar manualmente los parámetros.

Preparar la consulta

- Utilizamos el método **prepare** de la clase PDO.
- **Devuelve un objeto** de la clase **PDOStatement**.
- Los parámetros se pueden marcar utilizando signos de interrogación.

```
$pdoSt = $pdo->prepare('INSERT INTO familia (cod, nombre) VALUES (?, ?)');
```

- O podemos utilizar parámetros con nombre, precediéndolos por el símbolo de dos puntos.

```
$pdoSt = $pdo->prepare('INSERT INTO familia (cod, nombre) VALUES (:cod, :nombre)');
```

Asignar valor a los parámetros

- Antes de ejecutar la consulta, hay que **asignar un valor a los parámetros**.
- Utilizamos el **método bindParam de la clase PDOStatement**.
- Si hemos utilizado signo de interrogación pondremos el índice del parámetro empezando por 1:

```
$cod_producto = "TABLET";
$nombre_producto = "Tablet PC";
$pdoSt->bindParam(1, $cod_producto);
$pdoSt->bindParam(2, $nombre_producto);
```

- Si usamos parámetros con nombre, indicamos el nombre en la llamada a bindParam:

```
$pdoSt->bindParam(":cod", $cod_producto);
$pdoSt->bindParam(":nombre", $nombre_producto);
```

- Siempre debemos usar variables al pasar los parámetros ya que se trata de una referencia.

Ejecutar la consulta

- Una vez preparada la consulta y enlazados los parámetros con sus valores, se ejecuta la consulta utilizando el método **execute** de la clase PDOStatement.

- Es posible **asignar los valores de los parámetros en el momento de ejecutar la consulta**, utilizando un array (asociativo o con claves numéricas dependiendo de la forma en que hayas indicado los parámetros) en la llamada a **execute**.

```
$parametros = array(":cod" => "TABLET", ":nombre" => "Tablet PC");
$pdoSt->execute($parametros);
```

Obtener los resultados de la consulta

- Al igual que en las anteriores consultas podemos utilizar el método **fetch** para obtener los resultados de la consulta.
- También disponemos del método **fetchAll** que nos devolverá un array con el conjunto de resultados completo.