Database access from PHP(MySQL II)

DWES UD4.2

1.- PHP Data Objects → PDO

The MySQLi extension is a good option to work with MySQL databases as seen in the previous part of the topic.

The problem is that if the DBMS is changed in the future, a large part of the web application code will have to be reprogrammed.

Before starting the development of the web application, it is necessary to assess whether it is possible that the DBMS can be changed in the future. In that case it would be convenient to use an abstraction layer like the one offered by PDO.

1.- PHP Data Objects → PDO

PDO is a class that offers a set of properties and methods to perform operations on the database.

A PDO object (an instance of the class) represents a connection to the database.

PDO offers a data access abstraction layer that allows the same mechanisms to be used to perform queries regardless of the database used.

1.- PHP Data Objects → PDO

Establishing the connection

An object of the PDO class must be instantiated using its constructor. The constructor supports the following parameters (only the DSN is required):

- Data Source Name (DSN): A text string that indicates the driver and specific parameters that the driver uses.
- User with permissions on the database.
- User password.
- Connection options (array).

```
$dwes = new PDO('mysql:host=localhost;dbname=dwes', 'dwes', 'dwes');
```

1.- PHP Data Objects → PDO

Establishing the connection

The DSN chain is composed of the following:

PDO Driver: the specific PDO driver for the DBMS (mysql in this case)

- PDO connection string, includes the following:
    - **host**: server name or IP
    - **port**: server listening TCP port (optional)
    - **dbname**: database name
    - **unix_socket**: MySQL socket on UNIX systems, not used if 'port’ is using

mysql:**host**=hotsname;**port**=3309;**dbname**=dbname

mysql:**unix_socket**=/tmp/mysql.sock;**dbname**=dbname

1.- PHP Data Objects → PDO

Connection Options

A typical example of using connection options is the following. When you want to use UTF-8 encoding for all data transmitted:

```
$opc = array(PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES utf8’);
$dwes = new PDO('mysql:host=localhost;dbname=dwes', 'dwes', 'dwes', $opc);
```

The PDO constructor will throw an **exception** on error:

```
$opc = array(PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES utf8’);
try {
    $dwes = new PDO('mysql:host=localhost;dbname=dwes', 'dwes', 'dwes', $opc);
} catch (PDOException $e) {
    echo 'Falló la conexión: ' . $e->getMessage();
}
```

1.- PHP Data Objects → PDO

With the **getAttribute** method you can obtain connection status information and with the **setAttribute** method you can modify parameters:

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

Set field names to be received in uppercase:

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

1.- PHP Data Objects → PDO

Queries that do NOT return data → INSERT, DELETE, UPDATE

In MySQLi all queries are executed the same, but in PDO you have to take into account whether the query returns data or not.

For INSERT, DELETE and UPDATE queries, the exec method is used, which returns the number of affected rows.

```
$registros = $dwes->exec('DELETE FROM stock WHERE unidades=0;');
echo 'Se han borrado .' $registros .' registros';
```

1.- PHP Data Objects → PDO

Queries that DO return data→ SELECT

To execute SELECT queries, the **query** method is used.

This method returns an object of the **PDOStatement** class.

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

To access the returned data you can use the **fetch** method, it will return the next record if it exists, or false if there are no more records left.

```
while ($registro = $resultado->fetch()) {
        echo 'Producto '. $registro['producto'];
        echo ' ('. $registro['unidades'] .' unidades)<br>';
}
```

1.- PHP Data Objects → PDO

The **fetch** method returns by default an array with numeric and associative keys. You can change this behavior with an optional parameter of the **fetch** method.

- PDO::FETCH_BOTH
- PDO::FETCH_ASSOC
- PDO::FETCH_NUM
- PDO::FETCH_OBJECT

```
while ($registro = $resultado->fetch(PDO::FETCH_ASSOC)) {
  // instructions
                                                       }
```

1.- PHP Data Objects → PDO

Prepared queries that return NO data

```
$dwes = new  PDO('mysql:host=localhost;dbname=dwes', 'dwes', 'dwes');
$cod_producto = "TABLET";
$nombre_producto = "Tablet PC";

// Option 1
$consulta = $dwes->prepare('INSERT INTO familia (cod, nombre) VALUES (?, ?);');
$consulta->bindParam(1, $cod_producto);
$consulta->bindParam(2, $nombre_producto);

// Option 2
$consulta = $dwes->prepare('INSERT INTO familia (cod, nombre) VALUES (:cod, :nombre);');
$consulta->bindParam(':cod', $cod_producto);
$consulta->bindParam(':nombre', $nombre_producto);

$consulta->execute();
```

1.- PHP Data Objects → PDO

Prepared queries that return data

```
$dwes = new  PDO('mysql:host=localhost;dbname=dwes', 'dwes', 'dwes');
$cod_producto = "TABLET";
$nombre_producto = "Tablet PC";

// Option 2
$consulta = $dwes->prepare('SELECT nombre, precio FROM productos WHERE precio>:prec');
$consulta->bindParam(':prec', $prec);

$consulta->execute();  // Devuelve true/false según se ejecute con éxito o no

while(($resultado = $consulta->fetch(PDO::ASSOC)) != null) {
                echo $resultado['nombre'] .'(pvp: '. $resultado['precio'] .')<br>';
}
```

1.- PHP Data Objects → PDO

Transactions

```
$ok = true;

$dwes->beginTransaction();          // Returns true or false, whether or not the mode changes
if($dwes->exec($consulta1) == 0)
        $ok = false;
if($dwes->exec($consulta2) == 0)
        $ok = false;
// ...
if($dwes->exec($consultaN) == 0)
        $ok = false;

if ($ok)
        $dwes->commit();                        // If everything went well, confirm the changes
else
        $dwes->rollback();                      // and if not, it reverses them
```

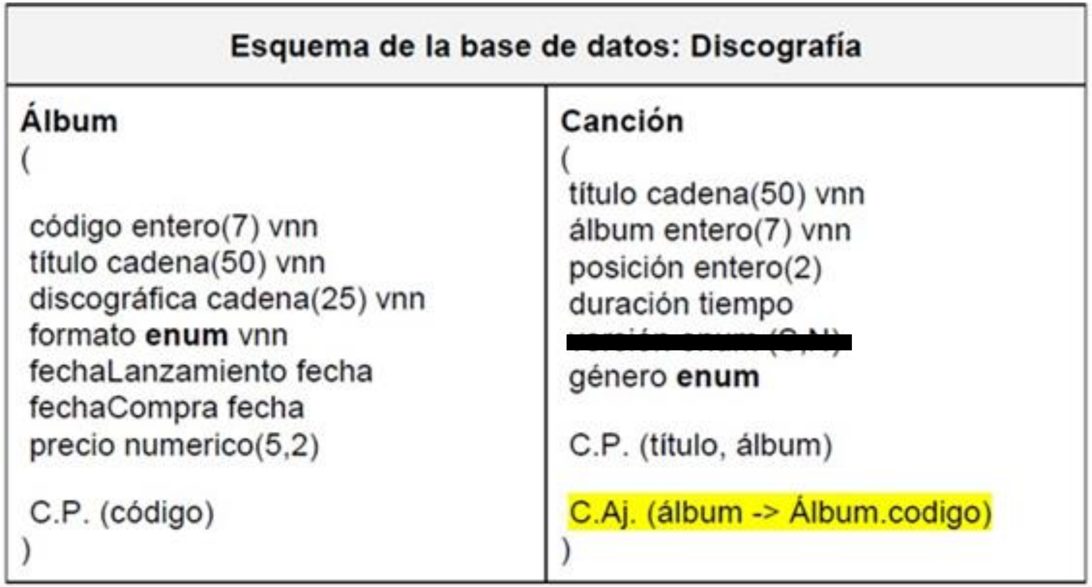
// After **commit** or **rollback** the DBMS returns to **autocommit** mode

Exercise - Discografia

Create a **virtual host** whose name is **discografia.local** and root directory **htdocs/discografia**.
Create the **discografia database** with the **discografia** user who has permissions on it.

format field values:
vinilo, cd, dvd, mp3

gender field values:
Clásica, BSO, Blues, Electrónica, Jazz, Metal, Pop, Rock



Esquema de la base de datos: Discografía

Álbum
(
código entero(7) vnn
título cadena(50) vnn
discográfica cadena(25) vnn
formato **enum** vnn
fechaLanzamiento fecha
fechaCompra fecha
precio numerico(5,2)

C.P. (código)
)

Canción
(
título cadena(50) vnn
álbum entero(7) vnn
posición entero(2)
duración tiempo
[redacted]
género **enum**

C.P. (título, álbum)

C.Aj. (álbum -> Álbum.codigo)
)

Exercise

The **discografia** application must have a main **index.php** page that displays a list of all the albums in the database.

In the list of albums, each album has to be converted into a link to the **album.php** page. This page should show all the songs on the album you receive as a parameter, as well as all the information on the album.

The album.php page will have, in addition to the above, two additional options:
1) One to add songs, it will be called **cancionnueva.php**, which will have a form to enter songs. In the header you must inform which album the song is being added to. It must be the file itself that receives the information from the form and saves the data in the database. After saving the song it should report that it was done successfully and show the form again.

Exercise

2) A link to delete the disk and all its songs (use a transaction), which must be called **borraralbum.php**. If there is an error, it will return to the disk page and report the error. If the transaction is completed correctly it will return to the main page reporting the deletion.

The main page should have another option to add a new disk, which will make a call to the **albumnuevo.php** page, which will show a form that allows you to insert disks. It must be the albumnuevo.php file itself that receives the information from the form and saves the data in the database.

After saving the new disk, it must be redirected to the main page and report that the disk has been created correctly, if any error has occurred it must be reported on the albumnuevo.php page.

Redirects in PHP

- Redirects in PHP are done with the header command:

```
header('Location: http://discografia.local/index.php');
header('Location: '.$nuevaURL.php);
```

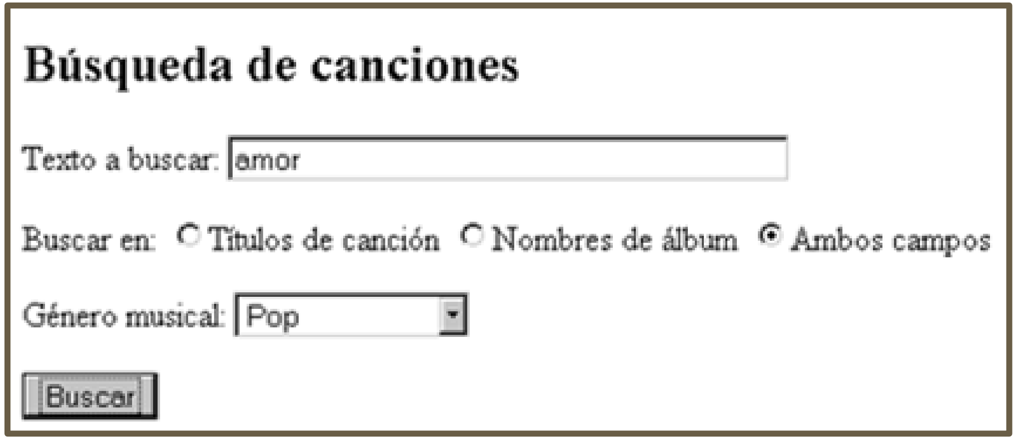
- Header must be called before any actual output is sent
- Add die or exit to ensure that code below header does not get executed when the redirection is done

```
<?php
  header("Location: http://www.example.com/");
  exit;
?>
```

Exercise

On the main page you must add an option to search for songs, which will call the **canciones.php** page, which will have a form that will allow you to search for songs.

It must be the file itself canciones.php the one you receive the form information and show the songs found.



Búsqueda de canciones

Texto a buscar: amor

Buscar en: ○ Títulos de canción ○ Nombres de álbum [✓ ● Ambos campos]

Género musical: Pop

Buscar

Make all connections to the database with **PDO**. Handle all exceptions and report all errors.