UT04 – Acceso a datos en aplicaciones Web:

En PHP hay drivers para manejar los sistemas gestores de bases de datos más extendidos. También hay varias extensiones que proveen una capa de abstracción sobre la base de datos, entre la que destaca **PHP Data Objects (PDO).** Permite manejar diferentes bases de datos con una interfaz común. Tiene la ventaja de que si se cambia de base de datos no hay que modificar el código.

1.- Conexión a la base de datos:

El primer paso para trabajar con una base de datos es obtener una conexión a la misma. Para representar la conexión se usa un objeto de clase PDO. El constructor es:

public PDO::__construct (string $dsn [, string $username [, string $passwd [, array $options]]])

el primer parámetro es una cadena que especifica que **driver** hay que usar, la localización y el nombre de la base de datos. Para una base de datos **MySQ**L se usaría:

mysql:dbname=<base de datos>;host=<ip o nombre>

Los siguientes parámetros son el nombre de usuario y clave para acceder a la base de datos. El último parámetro, opcional, es un array de opciones.

Si se puede establecer la conexión, se usará el nuevo objeto PDO para manejar la base de datos.

Si no se puede conectar con la base de datos, el constructor lanza una excepción **PDOException.**

```
<?php
$cadena_conexion = 'mysql:dbname=empresa;host=127.0.0.1';
$usuario = 'root';
$clave = '';
try {
    $bd = new PDO($cadena_conexion, $usuario, $clave);
} catch (PDOException $e) {
```

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

En principio, al acabar el script se cierra la conexión a la base de datos, pero también es posible usar conexiones persistentes, que no se cierran automáticamente al terminar el script.

Quedan abiertas y si se vuelven a utilizar no es necesario reestablecer la conexión, por lo que pueden ser más eficientes en determinadas ocasiones. Para crear una conexión persistente se utiliza la opción **PDO::ATTR_PERSISTENT** en el constructor:

```
$bd = new PDO ($cadena_conexion, $usuario, $clave,
array(PDO::ATTR_PERSISTENT => true));
```

Nota: Los ejemplos de este tema utilizan una base de datos empresa, que contiene la tabla:

Usuarios (Código, Nombre, Clave, Rol)

Código: es la clave primaria, un campo autonumérico.

Nombre: es el nombre de usuario y este marcado como '**unique'**.

Clave: es una cadena con la clave de acceso al sistema.

Rol: es un entero que representa el rol del usuario dentro del sistema.

2.- Recuperación y presentación de datos:

El método **query($cad)** de la clase PDO ejecuta la cadena que recibe como argumento en la base de datos, como se haría desde la línea de comando SQL. El argumento **$cad** tiene que ser una instrucción SQL válida. Devuelve FALSE si hubo algún error o un objeto **PDOStatement** si la cadena 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.**

```
<?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'];
            print " Clave : " . $usu['clave'] . "<br>";
        }
```

También es posible obtener instrucciones preparadas que permiten utilizar parámetros.

Se inicializan una sola vez con el método **prepare()** y luego se ejecutan las veces que sea necesario con **execute(),** con diferentes valores para los parámetros. Las instrucciones preparadas permiten reutilizar las consultas, previenen la inyección de código y mejoran el rendimiento.

Hay dos opciones para indicar los parámetros de la consulta, por posición y por nombre. En el primer caso se utiliza el símbolo de interrogación para indicar un parámetro. Al ejecutarla, se asocian por orden los símbolos de interrogación con los valores del array que se pasa como argumento a **execute().**

```
        /* consulta preparada, parametros por orden */
        $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>";
        }
```

Si se utilizan nombres en la instrucción preparada, utilizando: nombre, se deberán usar esos nombres como claves del array de execute().

```
        /* consulta preparada, parametros por nombre */
        $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>";
        }
        } catch (PDOException $e) {
            echo 'Error con la base de datos: ' . $e->getMessage();
        }
```

3.- Inserción:

Para insertar, borrar o actualizar simplemente hay que ejecutar la sentencia SQL correspondiente. Puede ser una sentencia preparada o no.

$ins: contiene la sentencia query que se va a ejecutar.

Para ejecutar la sentencia **$db->query($ins).**

La sintaxis genérica de INSERT para crear un nuevo registro es la siguiente:

INSERT INTO NombreTabla [(Campo1, …, CampoN)] VALUES (Valor1, …, ValorN)

- **NombreTabla:** la tabla en la que se van a insertar las filas.
- **(Campo1, …, CampoN):** representa el campo o campos en los que vamos a introducir valores.
- **(Valor1, …, ValorN):** representan los valores que se van a almacenar en cada campo.

En realidad, la lista de campos es opcional especificarla (por eso la hemos puesto entre corchetes en la sintaxis general). Si no se indica campo alguno se considera que por defecto vamos a introducir información en todos los campos de la tabla, y por lo tanto se deben introducir valores para todos ellos y en el orden en el que han sido definidos. En la práctica se suelen especificar siempre por claridad y para evitar errores.

```
<?php
// datos conexión
$cadena_conexion = 'mysql:dbname=empresa;host=127.0.0.1';
$usuario = 'root';
$clave = '';
try {
        // conectar
   $bd = new PDO($cadena_conexion, $usuario, $clave);
        echo "Conexión realizada con éxito<br>";
        // insertar nuevo usario
        $ins = "insert into usuarios(nombre, clave, rol) values('Yeray', '1234',
'1');";
        $resul = $bd->query($ins);
        //comprobar errores
        if($resul) {
                echo "insert correcto <br>";
                echo "Filas insertadas: " . $resul->rowCount() . "<br>";
        }else print_r( $bd -> errorinfo());
        // para los autoincrementos
        echo "Código de la fila insertada" . $bd->lastInsertId() . "<br>";
```

4.- Actualización:

Esta instrucción nos permite actualizar los valores de los campos de una tabla, para uno o varios registros, o incluso para todos los registros de una tabla.

Su sintaxis general es:

UPDATE NombreTabla

SET Campo1 = Valor1, ..., CampoN = ValorN

WHERE Condición

- **NombreTabla:** el nombre de la tabla en la que vamos a actualizar los datos.
- **SET:** indica los campos que se van a actualizar y con qué valores lo vamos a hacer.
- **WHERE:** Selecciona los registros de la tabla que se van a actualizar. Se puede aplicar todo lo visto para esta cláusula anteriormente, incluidas las sub-consultas.

```
// actualizar
$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());
```

5.- Borrar:

La instrucción DELETE permite eliminar uno o múltiples registros. Incluso todos los registros de una tabla, dejándola vacía.

Su sintaxis es general es:

DELETE [FROM] NombreTabla

WHERE Condición

- **NombreTabla:** el nombre de la tabla en la que vamos a actualizar los datos.
- **WHERE:** Selecciona los registros de la tabla que se van a actualizar. Se puede aplicar todo lo visto para esta cláusula anteriormente, incluidas las sub-consultas.

```
        // borrar
$del = "delete from usuarios where nombre = 'Luisa'";
$resul = $bd->query($del);
//comprobar errores
if($resul){
        echo "delete correcto <br>";
        echo "Filas borradas: " . $resul->rowCount() . "<br>";
}else print_r( $bd -> errorinfo());

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

6.- 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 que en un estado consistente. Las dos operaciones forman una única transacción.

Para indicar el comienzo de una transacción se utiliza el método **beginTransaction().**

La transacción se salva usando el método **commit().** Con el método **roolBack()** se deshacen las operaciones realizadas desde que se inició la transacción.

En el ejemplo **transaccion.php** se realizan dos inserciones como una transacción. Como la segunda falla (tiene un valor repetido para un campo **unique),** la primera se deshace.

```
<?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>";
        // comenzar la transacción
        $bd->beginTransaction();
        $ins = "insert into usuarios(nombre, clave, rol) values('Yeray', '1234',
'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();
```

```
        }
} catch (PDOException $e) {
    echo 'Error al conectar: ' . $e->getMessage();
}
```