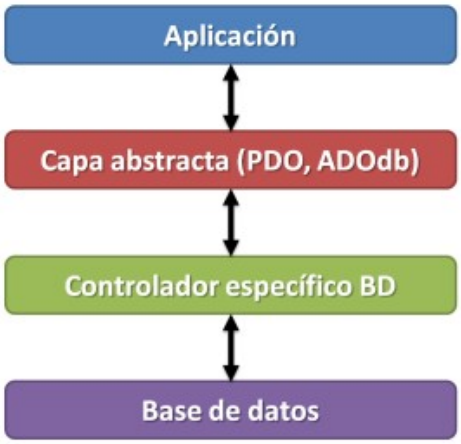
# ACCESO A DATOS CON PDO

## ¿Qué es PDO?

Las siglas **PDO (PHP Data Objects)** hacen referencia a una interfaz de PHP que nos permite acceder a bases de datos de cualquier tipo en PHP.

Cada **controlador de bases de datos que implemente la interfaz PDO** puede exponer características específicas de la base de datos, como las funciones habituales de la extensión. Obsérvese que no se puede realizar ninguna de las funciones de la bases de datos utilizando la extensión PDO por sí misma; se debe utilizar un **controlador de PDO específico de la base de datos** para tener acceso a un servidor de bases de datos.


**PDO** proporciona una capa de abstracción de acceso a datos, lo que significa que, independientemente de la base de datos que se esté utilizando, se usan las mismas funciones para realizar consultas y obtener datos.




Para saber los controladores PDO disponibles en nuestro sistema:

```
print_r(PDO::getAvailableDrivers());
```

## Clases PDO, PDOStatement y PDOException

- **Clase PDO:** se encarga de gestionar las conexiones entre PHP y un servidor de bases de datos. Proporciona métodos para gestionar las transacciones, obtener información sobre la conexión a la base de datos y preparar y ejecutar sentencias. http://php.net/manual/es/class.pdo.php
- **Clase PDOStatement**: representa una sentencia preparada y el resultado asociado a una consulta. Proporciona métodos para asignar valores a una sentencia preparada, para obtener información sobre un resultado (número de columnas, número de filas) y para recorrer un resultado.http://php.net/manual/es/class.pdostatement.php
- **Clase PDOException**: representa una excepción, un error generado por PDO. Proporciona métodos para obtener información sobre el error producido. http://php.net/manual/es/class.pdoexception.php

# Clase PDO

Las conexiones se establecen creando instancias de la clase PDO. El constructor de esta clase acepta parámetros para especificar el origen de datos (conocido como DSN, Data Source Name) y,opcionalmente, el nombre de usuario, la contraseña y opciones para el controlador. Los métodos más importantes de esta clase son:

- **exec** (sentencia): ejecuta una sentencia SQL que no devuelva un resultado (por ejemplo, INSERT, UPDATE o DELETE) y devuelve el número de filas afectadas.
- lastInsertId(): devuelve el ID autonumérico de la última fila insertada.
- prepare(sentencia): crea y devuelve una sentencia preparada para su posterior ejecución. Devuelve el resultado como un objeto de tipo PDOStatement.
- query (sentencia): ejecuta una sentencia SQL y devuelve el resultado como un objeto de tipo PDOStatement.

# Clase PDOStatement

La clase **PDOStatement** posee los siguientes métodos principales:

- bindParam (parametro, variable): vincula una variable a un parámetro en una sentencia preparada. Devuelve true en caso de éxito o false en caso de error.
- bindValue(parametro, valor): vincula un valor a un parámetro en una sentencia preparada. Devuelve true en caso de éxito o false en caso de error.
- columnCount(): devuelve el número de columnas de un resultado. 0 en caso de que no haya resultado
- execute(): ejecuta una sentencia preparada. Devuelve true en caso de éxito o false en caso de error.
- fetch(estilo): devuelve la siguiente fila en un resultado. La forma de devolver la fila se controla con el parámetro **estilo** que puede tomar los valores:
    - **PDO::FETCH_ASSOC**: devuelve un array indexado por los nombres de las columnas del resultado.
    - **PDO::FETCH_BOTH** (predeterminado): devuelve un array indexado tanto por los nombres de las columnas como por las

posiciones de las columnas en el resultado comenzando por la columna 0.

- **PDO::FETCH_NUM**: devuelve un array indexado por las posiciones de las columnas en el resultado comenzando por la columna 0.
- **PDO::FETCH_OBJ:** devuelve un objeto anónimo con nombres de propiedades que se corresponden a los nombres de las columnas del resultado.

- fetchAll(estilo): devuelve un array que contiene todas las filas de un resultado. La forma de devolver las filas se controla con el parámetro estilo que puede tomar los mismos valores que el método fetch(estilo).
- rowCount(): devuelve el número de filas afectadas por la última sentencia SQL.

# Clase PDOException

La clase PDOException posee los siguientes métodos principales:

- getFile(): devuelve la ruta y el nombre del fichero en el que se ha producido la excepción.
- getLine(): devuelve el número de la línea de código en la que se ha producido la excepción.
- getCode(): devuelve el código de la excepción.
- getMessage(): devuelve el mensaje de la excepción.
- __toString(): Obtiene un representación de string del objeto lanzado

```
<?php
try {
    throw new Exception("Some error message");
} catch(Exception $e) {

//Provoca que se ejecute el método __toString
    echo $e;
}
?>
```

Por defecto, la generación de excepciones está desactivada y no se producen excepciones (sin embargo, cuando se produce un error en la conexión, siempre se produce una excepción). Para activar la generación de excepciones se debe emplear el método setAttribute para modificar el atributo de configuración de errores PDO::ATTR_ERRMODE con el valor PDO::ERRMODE_EXCEPTION:

- PDO::ERRMODE_SILENT: el valor por defecto, no se generan excepciones (excepto para un error de conexión).
- PDO::ERRMODE_WARNING: los errores generan una advertencia de PHP y continúa la ejecución (útil para depurar un código).

- **PDO::ERRMODE_EXCEPTION**: se genera una excepción cuando se produce un error. **Utilizaremos este modo**

# CONEXIÓN BD

Para interactuar con la base de datos necesitamos conectarnos a la misma.

```
$db_hostname = "localhost";
$db_nombre = "usuarios";
/*
 * El usuario root núnca se puede usar, siempre cambiar por otro usuario
 * Nosotros lo usaremos para que nos funcionen a todos los ejemplos y los ejercicios
 */
$db_usuario = "root";
$db_clave = "";

//Ponemos la conexión dentro de un bloque try para poder capturar la excepción, si la hubiera

try{
    // Conectamos
    $db = new PDO('mysql:host=' . $db_hostname . ';dbname=' . $db_nombre . '', $db_usuario, $db_clave);
    // Realiza el enlace con la BD en utf-8
    $db->exec("set names utf8");
    //Accionamos el uso de excepciones
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

}
catch (PDOException $e) {
    // En este caso guardamos los errores en un archivo de errores log
    error_log($e->getMessage().microtime().PHP_EOL,3,"logerr.txt");
    //guardamos en ·errores el error que queremos mostrar a los usuarios
    $errores="Ha habido un error <br>";
}
```

Como podemos ver, el código de la conexión es sencillo. Simplemente creamos un objeto PDO y le pasamos 3 parámetros:

- Nombre del origen de datos
- El usuario
- La contraseña

Utilizamos un bloque **try** para capturar la excepción, en caso de error.

En el bloque **catch** optamos por almacenar la información concreta en un fichero de errores para posteriormente analizarlos. Para ello utilizamos la función error_log que se encarga de gestionar el envío del error, en nuestro caso a un fichero destino ya que $message_type lo ponemos a 3 e indicamos un destino.

# Cerrar la conexión a la base de datos

Se recomienda cerrar siempre la conexión a la base de datos cuando no se vaya a utilizar más durante nuestro proceso.

Hay que recordar que los **recursos son limitados** y cuando hay pocos usuarios no hay ningún problema, pero si tenemos muchos usuarios simultáneos entonces es cuando surgen problemas al haber alcanzado el número máximo de conexiones con el servidor, por ejemplo.

Al cerrar la conexión de forma explícita aceleramos la liberación de recursos para que estén disponibles para otros usuarios.

```
// Si quisiéramos cerrar la conexión con la base de datos simplemente podríamos
hacer.
$pdo = null;
```

# CREACIÓN DE LA BASE DE DATOS

Para crear la BD podemos hacerlo de varias maneras:

- Sino tienes un .sql con las sentencias de creación de la BD, puedes crearla con el entorno gráfico que proporciona phpMyAdmin. Recuerda que tanto la BD como los cotejamientos tienen que ser UTF8. (Ver el documento "tratamientoUTF8enMYSQL").
- Si tienes un fichero .sql que contenga las sentencias de creación de la BD:
  - ✓ Puedes importarlo desde phpMyAdmin o desde la consola.
  - ✓ Puedes crear un fichero .php que conecte con la BD y ejecute el fichero .sql. A continuación vemos como realizar este último.

```
//En config.php tenemos los valores de conexión a la BD
include ('config.php');
try {
    /*
     * Conectamos
     * No le pasamos nombre de BD porque vamos a crearla
     */
    $pdo = new PDO('mysql:host='.$db_hostname, $db_usuario, $db_clave);
    //UTF8
    $pdo->exec("set names utf8");
    // Accionamos el uso de excepciones
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    //Leemos el fichero que contiene el sql
    $sqlBD = file_get_contents("usuarios.sql");
    //Ejecutamos la consulta
    if ($pdo->exec($sqlBD)) {
        echo ("La BD ha sido creada");
    }
} catch (PDOException $e) {
    // En este caso guardamos los errores en un archivo de errores log
    error_log($e->getMessage() . "## Fichero: " . $e->getFile() . "## Línea: " .
$e->getLine() . "##Código: " . $e->getCode() . "##Instante: " . microtime() .
PHP_EOL, 3, "logerr.txt");
    // guardamos en ·errores el error que queremos mostrar a los usuarios
    $errores['datos'] = "Ha habido un error <br>";
}
```

# CONSULTAS SIN PREPARAR

En la clase PDO contamos con dos métodos que nos permiten realizar consultas directamente a la BD, sin preparar. **Sólo recomendable cuando utilicemos datos que sean de procedencia segura, en cualquier otro caso utilizar consultas preparadas.**

## exec

**exec** (sentencia): ejecuta una sentencia SQL que no devuelva un resultado (por ejemplo, INSERT, UPDATE o DELETE). **PDO::exec()** devuelve el número de filas modificadas o borradas por la sentencia SQL ejecutada. Si no hay filas afectadas, **PDO::exec()** devuelve 0, devuelve false en caso de error. **(Cuidado con confundir 0 con false, utiliza ===)**

```php
/* $pdo->exec("consulta SQL") -> Devolverá el número de registros afectados por la consulta
 * Recomendable paraINSERT, UPDATE o DELETE solo cuando los datos vengan de fuente segura.
 * Como resultado de la ejecución tendríamos en el número de filas afectadas por la instrucción.*/
require ("../conexion.php");


$consulta = "INSERT INTO `usuario` (nombre, fAlta, salario, activo) VALUES ('Juana', '2021-11-21', '1000', TRUE)";
// Podemos capturar las exepciones que pueda producir la cosulta

try {
    // Si ejecutamos la consulta con exec, devuelve el número de filas afectadas
    // Exec no sirve con SELECT
    $count = $pdo->exec($consulta);
    $pdo = NULL;
    if ($count === 1)
        echo "El usuario se ha insertado correctamente";
} catch (PDOException $e) {

    // En este caso guardamos los errores en un archivo de errores log
    error_log($e->getMessage() . "## Fichero: " . $e->getFile() . "## Línea: " . $e->getLine() . "##Código: " . $e->getCode() . "##Instante: " . microtime() . PHP_EOL, 3, "../logBD.txt");
    // guardamos en ·errores el error que queremos mostrar a los usuarios
    $errores['datos'] = "Ha habido un error <br>";
    // En este caso capturamos el caso de CP duplicada aunque al ser autonumérico no se puede duplicar
    if ($e->getCode() == 23000)
        // Guardamos los mensajes de errores para posteriormente mostrarlos
        $errores['datos'] = "Ya existe ese usuario en la BD";
    else
        $errores['datos'] = "Ha sucedido un error en la inserción";
}
```

query (sentencia): ejecuta una sentencia SQL y devuelve el resultado como un objeto de tipo PDOStatement. El objeto PDOStatement, implementa la interfaz Traversable, lo que significa que puede ser recorrida mediante un bucle foreach()

```php
<?php
include ('../conexion.php');
try {
    // Ejecuta una sentencia SQL
    $consulta = "SELECT * FROM usuario";
    If($resultado = $pdo->query($consulta)
    /*$resultado es un objeto de la clase PDOStatement.
```

```
     * Implementa la interfaz Traversable, lo que significa que puede ser recorrida
mediante un bucle foreach() sin ser un array
     */
    foreach ($resultado as $row) {
        echo $row['id'] . "<br> ";
        echo $row['nombre'] . "<br>";
        echo $row['fAlta'] . "<br>";
        echo $row['activo'] . "<br>";
    }

    /*
     *Otra opción sería emplear fetchAll(PDO::FETCH_ASSOC) para obtener todo el
resultado en forma de array bidimensional.
     * En nuetro caso será un array asociativo por el estilo que le hemos pasado. Lo
mostramos con print_r

    echo "<pre>";
    $arrayResultado=$resultado->fetchAll(PDO::FETCH_ASSOC);
    print_r( $arrayResultado);
    echo "</pre>";
    */

    // Para liberar un resultado simplemente hay que destruir el objeto
    $resultado = NULL; // Opcional
    // Para cerrar una conexión simplemente hay que destruir el objeto
    $pdo = NULL;
} catch (PDOException $e) {

    // En este caso guardamos los errores en un archivo de errores log
    error_log($e->getMessage() . "##Código: " . $e->getCode()."  ".  microtime() .
PHP_EOL, 3, "../logBD.txt");
    // guardamos en ·errores el error que queremos mostrar a los usuarios
    $errores['datos'] = "Ha habido un error <br>";

}
```

# CONSULTAS PREPARADAS

Una sentencia preparada o una sentencia parametrizada se usa para ejecutar la misma sentencia repetidamente con gran eficiencia. Además **las usaremos siempre que los datos no sean seguros** (de origen externo como formularios).

La ejecución de una sentencia preparada consta de tres partes:


- La **preparación**: se define una plantilla con marcadores (parámetros) que se envía al servidor para que realice una comprobación de su sintaxis e inicialice los recursos necesarios para su posterior ejecución. **PDO::prepare(consulta)**

- **Vinculación** de los valores a los parámetros. Puede hacerse con **PDOStatement::bindParam** o **PDOStatement::bindValue** o directamente en la ejecución.
- La **ejecución**: se envía al servidor para su ejecución con **PDOStatement::execute**

  El uso de las sentencias preparadas ofrece **dos ventajas** muy importantes:
  **1. Ofrece protección frente a la inyección SQL.**
  2. Ofrece un aumento de velocidad en la ejecución de una sentencia cuando se ejecuta varias veces: la sentencia no se tiene que analizar e interpretar cada vez, sino que está lista para ejecutarse tantas veces como se quiera.

## prepare

PDO::prepare(sentencia): crea y devuelve una sentencia preparada para su posterior ejecución. La sentencia SQL puede contener cero o más marcadores de parámetros con nombre (:name) o signos de interrogación (?) por los cuales los valores reales serán sustituidos cuando la sentencia sea ejecutada. Devuelve el resultado como un objeto de tipo PDOStatement.

### Preparadas con marcadores anónimos

```
# Marcadores anónimos
$stmt = $pdo->prepare("INSERT INTO usuario (nombre, fAlta, salario) values
(?, ?, ?)");
```

### Preparadas con marcadores conocidos

```
# Marcadores conocidos
$stmt = $pdo-> prepare("INSERT INTO usuarios (nombre, fAlta, salario) values
(:name, :addr, :city)");
```

La elección de usar marcadores anónimos o conocidos afectará a cómo se asignan los datos a esos marcadores.

## Asignación (bind) con parámetros anónimos

Al no dar nombre a los parámetros la asignación será posicional, de acuerdo al orden que se haya utilizado en prepare. La asignación de valores a los parámetros puede hacerse de varias formas:

```
//Ejemplo con bindValue. En este caso funcionaría igual con bindParam
 $nombre = "Luis";
    $fAlta = date('y-m-d');
    $salario = 1000;
try {
    // Preparamos consulta
    $stmt = $pdo->prepare("INSERT INTO usuario (nombre, fAlta, salario) values (?, ?,
?)");

    $stmt->bindValue(1, $nombre);
    $stmt->bindValue(2, $fAlta);
    $stmt->bindValue(3, $salario);
```

```
    // Excecute
    if ($stmt->execute()) {
        echo "Se ha insertado un usuario ";
    } else
        echo "No se ha insertado ningún usuario";
}
```

### Inserción de varios registros con bindParam o bindValue (ver diferencias)

Con **bindValue** se asigna el valor de la variable a ese parámetro justo en el momento de ejecutar la instrucción bindValue.

```
//Inserción de varios registros desde un array con bindValue
$usuarios = [
    ["Luis",date('y-m-d'),1000],["Ángela",date('y-m-d'),1200]
];
    //Preparamos la consulta sólo una vez
    $stmt = $pdo->prepare("INSERT INTO usuario (nombre, fAlta, salario) values (?, ?,
?)");

    // Vinculamos y ejecutamos dentro del bucle
    foreach ($usuarios as $valor){
        $stmt->bindValue(1, $valor[0]);
        $stmt->bindValue(2, $valor[1]);
        $stmt->bindValue(3, $valor[2]);
        // Excecute
        $stmt->execute();
    }
```

Con **bindParam** se vincula la variable al parámetro y en el momento de hacer el execute es cuando se asigna realmente el valor de la variable a ese parámetro.

```
//Inserción de varios registros desde un array con bindParam
$usuarios = [
    ["Luis",date('y-m-d'),1000],["Ángela",date('y-m-d'),1200]
];

$stmt = $pdo->prepare("INSERT INTO usuario (nombre, fAlta, salario) values
(?, ?, ?)");
    $stmt->bindParam(1, $valor0);
    $stmt->bindParam(2, $valor1);
    $stmt->bindParam(3, $valor3);

    //Dentro del bucle van cambiando los valores de $valor0, $valor1 y $valor2 y las
ejecuciones (execute)
    foreach ($usuarios as $valor){
        $valor0= $valor[0];
        $valor1= $valor[1];
        $valor2= $valor[2];
        // Excecute
        $stmt->execute();
```

### Asignación en execute utilizando un array:

```
include ('../conexion.php');
$usuarios = [
    "Lucía",
```

```
    date('y-m-d'),
    2500
];
try {
    // Preparamos la consulta
    $stmt = $pdo->prepare("INSERT INTO usuario (nombre, fAlta, salario) values (?, ?,
?)");

    // Vinculamos al ejecutar utilizando el array

    if ($stmt->execute($usuarios))
        echo "El id del último usuario dado de alta es: " . $pdo->lastInsertId();
```

## Asignación (bind) con parámetros con nombre

```
$stmt = $pdo->prepare("INSERT INTO usuario (nombre, fAlta, salario) values
(:nombre,:fAlta,:salario)");

    // Bind

    $stmt->bindParam(':nombre', $nombre);
    $stmt->bindParam(':fAlta', $fAlta);
    $stmt->bindParam(':salario', $salario);
    // Excecute
    if ($stmt->execute()) {

        echo "El id del último usuario dado de alta es: " . $pdo->lastInsertId();
    } else
        echo "No se ha insertado ningún registro";
```

### Asignación en execute utilizando un array asociativo

```
//En este caso el array tiene que ser asociativo y coincidir el nombre de los
parámetros con los índices
$usuarios = [
   'nombre'=> "Lucía",
    'fAlta'=>date('y-m-d'),
    'salario'=>2500
];
try {
    // Preparamos la consulta
    $stmt = $pdo->prepare("INSERT INTO usuario (nombre, fAlta, salario) values
(:nombre,:fAlta,:salario)");

    // Vinculamos al ejecutar utilizando el array asociativo

    if ($stmt->execute($usuarios))
        echo "El id del último usuario dado de alta es: " . $pdo->lastInsertId();
```

## Ejemplo select con varias opciones de parámetros y recorrido

```
include ('../conexion.php');
$usuario = 'Juana';
$consulta = "select * FROM usuario
    WHERE nombre=:usuario";
try {
```

```php
    $resultado = $pdo->prepare($consulta);
    if ($resultado->execute(array(
        ":usuario" => $usuario
    ))) {
        // Recorremos el objeto con foreach aunque no es un array. Interfaz
Traversable
        foreach ($resultado as $row) {

            echo $row['id'] . "<br> ";
            echo $row['nombre'] . "<br>";
            echo $row['fAlta'] . "<br>";
            echo $row['activo'] . "<br>";
        }
    }
    // Con fetchAll(PDO::FETCH_ASSOC) convertimos el resultado en array asociativo
    // Utilizamos parámetros sin nominar

    echo "Ahora muestro los resultados con fetchAll() y parámetros anónimos</br>";
    $consulta = "select * FROM usuario WHERE nombre=?";
    $resultado = $pdo->prepare($consulta);
    $resultado->bindParam(1, $usuario);
    if ($resultado->execute()) {
        $arrayResultado = $resultado->fetchAll(PDO::FETCH_ASSOC);
        foreach ($arrayResultado as $row) {
            echo $row['id'] . "<br> ";
            echo $row['nombre'] . "<br>";
            echo $row['fAlta'] . "<br>";
            echo $row['activo'] . "<br>";
        }
    }
    // Con fetch(PDO::FETCH_ASSOC) recorremos todas las filas del resultado. Cada una
como array asociativo
    // Utilizamos parámetros con nombre
    echo "Ahora muestro los resultados con fetch() y parámetros con nombre</br>";
    $consulta = "select * FROM usuario WHERE nombre=:usuario";
    $resultado = $pdo->prepare($consulta);
    $resultado->bindParam(":usuario", $usuario);
    if ($resultado->execute()) {
        // $stmt->setFetchMode(PDO::FETCH_ASSOC);

        # Mostramos los resultados.
        # Fijaros que se devuelve un objeto cada vez que se lee una fila del
recordset.
        while ($row = $resultado->fetch(PDO::FETCH_ASSOC)) {

            echo $row['id'] . "<br> ";
            echo $row['nombre'] . "<br>";
            echo $row['fAlta'] . "<br>";
            echo $row['activo'] . "<br>";
        }
    }
} catch (PDOException $e) {

    // En este caso guardamos los errores en un archivo de errores log
    error_log($e->getMessage() . "##Código: " . $e->getCode() . "  " . microtime() .
PHP_EOL, 3, "../logBD.txt");
    // guardamos en ·errores el error que queremos mostrar a los usuarios
    $errores['datos'] = "Ha habido un error <br>";
}

?>
```