Bases de datos en PHP

Bases de datos en PHP

- PHP soporta más de 15 sistemas gestores de bases de datos: SQLite, Oracle, SQL Server, PostgreSQL, IBM DB2, MySQL, etc.
- Hasta la versión 5 de PHP, el acceso a las bases de datos se hacía principalmente utilizando extensiones específicas para cada sistema gestor de base de datos (extensiones nativas).
- Si queríamos acceder a una base de datos de PostgreSQL, deberíamos instalar y utilizar la extensión de ese gestor en concreto. Las funciones y objetos a utilizar eran distintos para cada extensión.
- A partir de la versión 5 de PHP se introdujo en el lenguaje una extensión para acceder de una forma común a distintos sistemas gestores: PDO.

Bases de datos en PHP

- La gran ventaja de PDO está clara: podemos seguir utilizando una misma sintaxis aunque cambiemos el motor de nuestra base de datos.
- En algunas ocasiones preferiremos seguir usando extensiones nativas en nuestros programas. Mientras PDO ofrece un conjunto común de funciones, las extensiones nativas normalmente ofrecen más potencia (acceso a funciones específicas de cada gestor de base de datos) y en algunos casos también mayor velocidad.
- De los distintos SGBD existentes, vamos a utilizar MySQL/MaríaDB.
- Accederemos desde PHP a bases de datos MySQL utilizando tanto PDO como la extensión nativa MySQLi.

Bases de datos en PHP

- Para el acceso a las funcionalidades de ambas extensiones utilizaremos objetos.
- En PHP se utiliza la palabra new para crear un nuevo objeto instanciando una clase:

```
$a = new A();
```

- Y para acceder a los miembros de un objeto, debes utilizar el operador flecha ->:

```
$a->fecha();
```

Bases de datos en PHP

Acciones

- Establecer conexiones.
- Ejecutar sentencias SQL.
- Obtener los registros afectados o devueltos por una sentencia SQL.
- Emplear transacciones.
- Ejecutar procedimientos almacenados.
- Gestionar los errores que se produzcan durante la conexión o en el establecimiento de la misma.

Bases de datos en PHP

MySQLi

- Esta extensión está incluida en PHP desde la versión 5.
- Ofrece una interfaz de programación dual: se puede acceder a la extensión usando objetos o funciones de forma indiferente.

```php
// utilizando constructores y métodos de la programación orientada a objetos
$conexion = new mysqli('servidor', 'usuario', 'contraseña', 'base_de_datos');
print conexion->server_info;

// utilizando llamadas a funciones
$conexion = mysqli_connect('servidor', 'usuario', 'contraseña', 'base_de_datos');
print mysqli_get_server_info($conexion);
```

Bases de datos en PHP

MySQLi

- Parámetros de configuración propios de la extensión MySQLi en php.ini
    - **mysqli.allow_persistent.** Permite crear conexiones persistentes.
    - **mysqli.default_port.** Número de puerto TCP predeterminado a utilizar cuando se conecta al servidor de base de datos.
    - **mysqli.reconnect.** Indica si se debe volver a conectar automáticamente en caso de que se pierda la conexión.
    - **mysqli.default_host.** Host predeterminado a usar cuando se conecta al servidor de base de datos.
    - **mysqli.default_user.** Nombre de usuario predeterminado a usar cuando se conecta al servidor de base de datos.
    - **mysqli.default_pw.** Contraseña predeterminada a usar cuando se conecta al servidor de base de datos.

Bases de datos en PHP

MySQLi - Conexión

- Establecer una conexión con el servidor significa crear una instancia de la clase mysqli.
- El constructor de la clase puede recibir seis parámetros, todos opcionales, aunque lo más habitual es utilizar los cuatro primeros:
    - El nombre o dirección IP del servidor MySQL al que te quieres conectar.
    - Un nombre de usuario con permisos para establecer la conexión.
    - La contraseña del usuario.
    - El nombre de la base de datos a la que conectarse.

Bases de datos en PHP

MySQLi - Conexión

```php
// utilizando el constructor de la clase
$conProyecto = new mysqli('localhost', 'gestor', 'secreto', 'proyecto');

// utilizando el método connect
$conProyecto = new mysqli();
$conProyecto->connect('localhost', 'gestor', 'secreto', 'proyecto');
```

```php
// utilizando llamadas a funciones
$conProyecto = mysqli_connect('localhost', 'gestor', 'secreto', 'proyecto');
```

Bases de datos en PHP

MySQLi - Conexión

- Para comprobar el error, en caso de que se produzca, puedes usar las siguientes propiedades (o funciones equivalentes) de la clase mysqli:
  - connect_errno (o la función mysqli_connect_errno) devuelve el número de error o null si no se produce ningún error.
  - connect_error (o la función mysqli_connect_error) devuelve el mensaje de error o null si no se produce ningún error.

```php
@$conProyecto = new mysqli('localhost', 'gestor', 'secreto', 'proyecto');
$error = $conProyecto->connect_errno;
if ($error != null) {
    echo "<p>Error $error conectando a la base de datos: $conProyecto->connect_error</p>";
    die();
}
```

Bases de datos en PHP

MySQLi - Conexión

- Si una vez establecida la conexión, quieres cambiar la base de datos puedes usar el método "select_db" (o la función "mysqli_select_db" de forma equivalente) para indicar el nombre de la nueva, lógicamente el usuario con el que hemos iniciado la conexión debe tener permisos en la nueva.
- Una vez finalizadas las tareas con la base de datos, utiliza el método "close" (o la función "mysqli_close") para cerrar la conexión con la base de datos y liberar los recursos que utiliza.

Bases de datos en PHP

MySQLi - Ejecución de consultas

- Para ejecutar una consulta debemos utilizar el método query, equivalente a la función mysqli_query.
- Si se ejecuta una consulta de acción que no devuelve datos (como una sentencia SQL de tipo UPDATE, INSERT o DELETE), la llamada devuelve true si se ejecuta correctamente o false en caso contrario.
- El número de registros afectados se puede obtener con la propiedad affected_rows (o con la función mysqli_affected_rows).
- En el caso de ejecutar una sentencia SQL que sí devuelva datos (como un SELECT), éstos se devuelven en forma de un objeto resultado (de la clase "mysqli_result"). En el punto siguiente verás cómo se pueden manejar los resultados obtenidos.

Bases de datos en PHP

MySQLi - Ejecución de consultas

```php
@$conProyecto = new mysqli('localhost', 'gestor', 'secreto', 'proyecto');
$error = $conProyecto->connect_errno;
if ($error == null) {
    $resultado = $conProyecto->query('DELETE FROM stock WHERE unidades=0');
    if ($resultado) {
        echo "<p>Se han borrado $conProyecto->affected_rows registros.</p>";
    }
    $conProyecto->close(); //cerramos la conexion
}
```

- Es importante tener en cuenta que los resultados obtenidos se almacenarán en memoria mientras los estés usando. Cuando ya no los necesites, los puedes liberar con el método free de la clase mysqli_result (o con la función mysqli_free_result):

Bases de datos en PHP

MySQLi - Obtención y uso de los resultados

- Al ejecutar una consulta que devuelve datos obtienes un objeto de la clase mysqli_result.
- Esta clase sigue los criterios de ofrecer un interface de programación dual, es decir, una función por cada método con la misma funcionalidad que éste.
- Para trabajar con los datos obtenidos del servidor, tienes varias posibilidades:
    - fetch_array **(función mysql_fetch_array):** Obtiene un registro completo del conjunto de resultados y lo almacena en un array. Por defecto el array contiene tanto claves numéricas como asociativas. Por ejemplo, para acceder al primer campo devuelto, podemos utilizar como clave el número 0 o su nombre indistintamente.

Bases de datos en PHP

MySQLi - Obtención y uso de los resultados - fetch_array

```php
$resultado = $conProyecto->query('SELECT producto, unidades FROM stocks WHERE unidades<2');
$stock = $resultado->fetch_array();  // Obtenemos el primer registro
$producto = $stock['producto'];  // O también $stock[0];
$unidades = $stock['unidades'];  // O también $stock[1];
echo "<p>Producto $producto: $unidades unidades.</p>";
```

- Este comportamiento se puede modificar usando el parámetro opcional que puede tomar alguno de los siguientes valores:
    - **MYSQLI_NUM:** Devuelve un array con claves numéricas.
    - **MYSQLI_ASSOC:** Devuelve un array asociativo.
    - **MYSQLI_BOTH:** Es el comportamiento por defecto, en el que devuelve un array con claves numéricas y asociativas.

Bases de datos en PHP

MySQLi - Obtención y uso de los resultados

- El resto de posibilidades son las siguientes:
  - fetch_assoc **(función mysqli_fetch_assoc):** Idéntico a fetch_array pasando como parámetro MYSQLI_ASSOC.
  - fetch_row **(función mysqli_fetch_row):** Idéntico a fetch_array pasando como parámetro MYSQLI_NUM.
  - fetch_object **(función mysqli_fetch_object):** Similar a los métodos anteriores, pero devuelve un objeto en lugar de un array. Las propiedades del objeto devuelto se corresponden con cada uno de los campos del registro.

Bases de datos en PHP

MySQLi - Obtención y uso de los resultados

- Parar recorrer todos los registros de un array, puedes hacer un bucle teniendo en cuenta que cualquiera de los métodos o funciones anteriores devolverán null cuando no haya más registros en el conjunto de resultados:

```php
$resultado = $conProyecto->query('SELECT producto, unidades FROM stocks WHERE unidades<2');
$stock = $resultado->fetch_object();
while ($stock != null) {
        echo "<p>Producto $stock->producto: $stock->unidades unidades.</p>";
        $stock = $resultado->fetch_object();
}
```

Bases de datos en PHP

MySQLi - Transacciones

- Por defecto, en MySQL (mediante el motor InnoDB) cada consulta individual se incluye dentro de su propia transacción.
- Esto se puede deshabilitar mediante `$db->autocommit(false);`
- A partir de ese momento, las siguientes operaciones sobre la base de datos iniciarán una transacción que deberás finalizar utilizando:
  - commit (o la función mysqli_commit). Realizar una operación "commit" de la transacción actual, devolviendo true si se ha realizado correctamente o false en caso contrario.
  - rollback (o la función mysqli_rollback). Realizar una operación "rollback" de la transacción actual, devolviendo true si se ha realizado correctamente o false en caso contrario.

Bases de datos en PHP

MySQLi - Transacciones

- La tienda 1 (CENTRAL) tiene 2 unidades del producto de código 3DSNG y la tienda 3 (SUCURSAL2) ninguno.
- Utilizamos una transacción para mover una unidad de ese producto de la tienda 1 a la tienda 3.

```php
//Definimos una variable para comprobar que no tenemos errores
$todoBien=true;
//Iniciamos la transacción
$conProyecto->autocommit(false);
$update="update stocks set unidades=1 where producto=(select id from productos
where nombre_corto='3DSNG') AND tienda=1";
if(!$conProyecto->query($update)){
    $todoBien=false;
}
//fijate en este insert, el select devolverá el productos.id del producto de
nombre_corto='3DSNG' 3 y 1 es decir
// estamos haciendo un insert into stocks(producto, tienda, unidades) lo valores
1, 3, 1
$insert="insert into stocks(producto, tienda, unidades) select id, 3, 1 from
productos where nombre_corto='3DSNG'";
if(!$conProyecto->query($insert)){
    $todoBien=false;
}
//Si todo fue bien hacemos el commit si no el rollback
if($todoBien){
    $conProyecto->commit();
    echo "<p>Los cambios se han realizado correctamente.</p>";
}
else{
    $conProyecto->rollback();
    echo "<p>No se han podido realizar los cambios.</p>";
}
```

Bases de datos en PHP

PHP Data Objects (PDO)

- Si vamos a programar una aplicación que utilice como sistema gestor de bases de datos MySQL, la extensión MySQLi que acabamos de ver es una buena opción.
- Si en el futuro tenemos que cambiar el SGBD por otro distinto, tendríamos que volver a programar gran parte del código de la misma.
- El objetivo de PDO es que si llegado el momento necesitamos cambiar el servidor de base de datos, las modificaciones que tengamos que realizar en el código sean mínimas.
- PDO no abstrae de forma completa el sistema gestor que se utiliza. Por ejemplo, no modifica las sentencias SQL para adaptarlas a las características específicas de cada servidor.

Bases de datos en PHP

PDO

- La extensión PDO debe utilizar un driver o controlador específico para el tipo de base de datos que se utilice. Para consultar los controladores disponibles en tu instalación de PHP, puedes utilizar la información que proporciona la función phpinfo().
- PDO no ofrece un interface de programación dual. Únicamente se pueden utilizar objetos con sus métodos y propiedades.

Bases de datos en PHP

PDO - Conexión

- Para conectar con la base de datos hay que instanciar un objeto de la clase PDO pasándole los siguientes parámetros:
    - Origen de datos (DSN). Es una 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.

Bases de datos en PHP

PDO - Conexión

```php
$host = "localhost";
$db = "proyecto";
$user = "gestor";
$pass = "secreto";
$dsn = "mysql:host=$host;dbname=$db";
$conProyecto=new PDO($dsn, $user, $pass);
//se recomienda guardar los datos(host, user...) en variables porque si estos cambian
//solo tenemos que actualizar el valor de estas variables
```

Bases de datos en PHP

PDO - Conexión

- Para cerrar la conexión es necesario destruir el objeto asegurándose de que todas las referencias a él existentes sean eliminadas; esto se puede hacer asignando null a la variable que contiene el objeto.

```
$conProyecto = null;
```

- Una vez establecida la conexión, podemos utilizar el método getAttribute para obtener información del estado de la conexión y setAttribute para modificar algunos parámetros que afectan a la misma.

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

Bases de datos en PHP

PDO - Conexión

- Para controlar los errores tenemos el atributo: ATTR_ERRMODE con los posible valores:
    - **ERRMODE_SILENT:** El modo por defecto, no muestra errores (se recomienda en entornos en producción).
    - **ERRMODE_WARNING:** Además de establecer el código de error, emitirá un mensaje E_WARNING, es el modo empleado para depurar o hacer pruebas para ver errores sin interrumpir el flujo de la aplicación.
    - **ERRMODE_EXCEPTION:** Además de establecer el código de error, lanzará una PDOException que podemos capturar en un bloque try catch(). Lo veremos más adelante.

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

Bases de datos en PHP

PDO - Consultas

- Para ejecutar una consulta SQL utilizando PDO, debemos diferenciar aquellas sentencias SQL que no devuelven como resultado un conjunto de datos, de aquellas otras que sí lo devuelven.
- En el caso de las consultas de acción, como INSERT, DELETE o UPDATE, el método exec devuelve el número de registros afectados.

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

Bases de datos en PHP

PDO - Consultas

- Si la consulta genera un conjunto de datos, como es el caso de SELECT, hay que utilizar el método query, que devuelve un objeto de la clase PDOStatement.
- Hay varias posibilidades para tratar con el conjunto de resultados devuelto por el método query. La más utilizada es el método fetch de la clase PDOStatement. Este método devuelve un registro del conjunto de resultados, o false si ya no quedan registros por recorrer.

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

```
$conProyecto = new PDO(". . .");
$resultado = $conProyecto->query("SELECT producto, unidades FROM stocks");
while ($registro = $resultado->fetch(PDO::FETCH_OBJ)) {
   echo "Producto ".$registro->producto.": ".$registro->unidades."<br />";
}
```

Bases de datos en PHP

PDO - Consultas

- El método fetch genera y devuelve a partir de cada registro un array con claves numéricas y asociativas.
- Para cambiar su comportamiento, admite un parámetro opcional que puede tomar uno de los siguientes valores:
  - PDO::FETCH_ASSOC. Devuelve solo un array asociativo.
  - PDO::FETCH_NUM. Devuelve solo un array con claves numéricas.
  - PDO::FETCH_BOTH. Devuelve un array con claves numéricas y asociativas. Es el comportamiento por defecto.
  - PDO::FETCH_OBJ. Devuelve un objeto cuyas propiedades se corresponden con los campos del registro.
  - PDO::FETCH_LAZY. Devuelve tanto el objeto como el array con clave dual anterior.

Bases de datos en PHP

PDO - Consultas

- También podemos utilizar fecthAll() que te trae todos los datos de golpe, sin abrir ningún puntero, almacenándolos en un array. Se recomienda cuando no se esperan demasiados resultados.

```
$resultado = $stmt->fetchAll(PDO::FETCH_ASSOC);
foreach ($resultado as $row){
    echo $row["nombre"]." ".$row["apellido"];
}
```

Bases de datos en PHP

PDO - Transacciones

- Por defecto PDO trabaja en modo "autocommit", esto es, confirma de forma automática cada sentencia que ejecuta el servidor. Para trabajar con transacciones, PDO incorpora tres métodos:
  - beginTransaction. Deshabilita el modo "autocommit" y comienza una nueva transacción, que finalizará cuando ejecutes uno de los dos métodos siguientes.
  - commit. Confirma la transacción actual.
  - rollback. Revierte los cambios llevados a cabo en la transacción actual.
- Una vez ejecutado un commit o un rollback, se volverá al modo de confirmación automática.

Bases de datos en PHP

PDO - Transacciones

```
$ok = true;

$conProyecto->beginTransaction();

if(!$conProyecto->exec('DELETE …')) $ok = false;

if(!$conProyecto->exec('UPDATE …')) $ok = false;

…

if ($ok) $conProyecto->commit();  // Si todo fue bien confirma los cambios

else $dwes->rollback();   //  y si no, los revierte
```

Bases de datos en PHP

PDO - Transacciones - excepciones

```
try {
    // connection successful
    $db = new PDO("mysql:host=localhost;dbname=banking_sys", "petermac", "abc123");
} catch (Exception $error) {
    die("Connection failed: " . $error->getMessage());
}
```

```
try {
    $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $db->beginTransaction();
    $db->exec("insert into accounts (account_id, amount) values (23, '5000')" );
    $db->exec("insert into accounts (account_id, amount) values (27, '-5000')" );
    $db->commit();
} catch (Exception $error) {
    $db->rollback();
    echo "Transaction not completed: " . $error->getMessage();
}
```

Bases de datos en PHP

Gestión de errores

- PHP define una clasificación de los errores que se pueden producir en la ejecución de un programa y ofrece métodos para ajustar el tratamiento de los mismos.

- Para hacer referencia a cada uno de los niveles de error, PHP define una serie de constantes. Cada nivel se identifica por una constante. Por ejemplo, la constante E_NOTICE hace referencia a avisos que pueden indicar un error al ejecutar un script, y la constante E_ERROR engloba errores fatales que provocan que se interrumpa forzosamente la ejecución.

Bases de datos en PHP

Gestión de errores

- La configuración inicial de cómo se va a tratar cada error según su nivel se realiza en php.ini el fichero de configuración de PHP. Entre los principales parámetros que podemos ajustar están:
    - **error_reporting:** Indica qué tipos de errores se notificarán.Su valor predeterminado es E_ALL & ~E_NOTICE que indica que se notifiquen todos los errores (E_ALL) salvo los avisos en tiempo de ejecución (E_NOTICE).
    - **display_errors:** En su valor por defecto (On), hace que los mensajes se envíen a la salida estándar (y por lo tanto se muestren en el navegador). Se debe desactivar (Off) en los servidores que no se usan para desarrollo sino para producción.

Bases de datos en PHP

Gestión de errores

- Desde el código, podemos usar la función error_reporting con las constantes anteriores para establecer el nivel de notificación en un momento determinado.
- Por ejemplo, si en algún lugar del código figura una división en la que exista la posibilidad de que el divisor sea cero, cuando esto ocurra obtendremos un mensaje de error en el navegador. Para evitarlo, podemos desactivar la notificación de errores de nivel E_WARNING antes de la división y restaurarla a su valor normal a continuación:

```
error_reporting(E_ALL & ~E_NOTICE & ~E_WARNING);
$resultado = $dividendo / $divisor;
error_reporting(E_ALL & ~E_NOTICE);
```

Bases de datos en PHP

Gestión de errores

- Podemos programar una función para que sea la que se ejecuta cada vez que se produce un error.
- El nombre de esa función se indica utilizando set_error_handler y debe tener como mínimo dos parámetros obligatorios (el nivel del error y el mensaje descriptivo) y hasta otros tres opcionales con información adicional sobre el error (el nombre del fichero en que se produce, el número de línea, y un volcado del estado de las variables en ese momento).

Bases de datos en PHP

Gestión de errores

```
set_error_handler("miGestorDeErrores");
$resultado = $dividendo / $divisor;
restore_error_handler();
function miGestorDeErrores($nivel, $mensaje)
{
    switch($nivel) {
        case E_WARNING:
            echo "Error de tipo WARNING: $mensaje.<br />";
            break;
        default:
            echo "Error de tipo no especificado: $mensaje.<br />";
    }
}
```

La función restore_error_handler restaura el manejador de errores original de PHP (más concretamente, el que se estaba usando antes de la llamada a set_error_handler).

Bases de datos en PHP

Excepciones

- A partir de la versión 5 se introdujo en PHP un modelo de excepciones similar al existente en otros lenguajes de programación:
    - El código susceptible de producir algún error se introduce en un bloque try.
    - Cuando se produce algún error, se lanza una excepción utilizando la instrucción throw.
    - Después del bloque try debe haber como mínimo un bloque catch encargado de procesar el error.
    - Si una vez acabado el bloque try no se ha lanzado ninguna excepción, se continúa con la ejecución en la línea siguiente al bloque o bloques catch.

Bases de datos en PHP

Excepciones

- Por ejemplo, para lanzar una excepción cuando se produce una división por cero podríamos hacer:

```
try {
    if ($divisor == 0)
    throw new Exception("División por cero.");
    $resultado = $dividendo / $divisor;
}catch (Exception $e) {
    echo "Se ha producido el siguiente error: ".$e->getMessage();
}
```

Bases de datos en PHP

Excepciones

- PHP ofrece una clase base Exception para utilizar como manejador (handler) de excepciones. Para lanzar una excepción no es necesario indicar ningún parámetro, aunque de forma opcional se puede pasar un mensaje de error (como en el ejemplo anterior) y también un código de error.
- Entre los métodos que puedes usar con los objetos de la clase Exception están:
  - getMessage. Devuelve el mensaje, en caso de que se haya puesto alguno.
  - getCode. Devuelve el código de error si existe.

Bases de datos en PHP

Excepciones

- Las extensiones más modernas orientadas a objetos, como es el caso de PDO, utilizan este modelo de excepciones.
- En este caso, lo más común es que la extensión defina sus propios manejadores de errores heredando de la clase Exception.

Bases de datos en PHP

Excepciones

- La clase PDO permite definir la fórmula que usará cuando se produzca un error, utilizando el atributo PDO::ATTR_ERRMODE. Las posibilidades son:
  - **PDO::ERRMODE_SILENT.** El modo por defecto, no muestra errores (se recomienda en entornos en producción).
  - **PDO::ERRMODE_WARNING.** Además de establecer el código de error, emitirá un mensaje E_WARNING, es el modo empleado para depurar o hacer pruebas para ver errores sin interrumpir el flujo de la aplicación.
  - **PDO::ERRMODE_EXCEPTION.** Cuando se produce un error lanza una excepción utilizando el manejador propio PDOException.
- PDO::__construct() siempre lanzará una PDOException si la conexión falla, independientemente de que PDO::ATTR_ERRMODE esté esteblecido.
- Para utilizar excepciones con la extensión PDO, se debe configurar la conexión haciendo:

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

Bases de datos en PHP

Excepciones

```
$host = "localhost";
$db = "proyecto";
$user = "gestor";
$pass = "1234";
$dsn = "mysql:host=$host;dbname=$db";
try {
    $conProyecto = new PDO($dsn, $user, $pass);
    $conProyecto->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $ex) {
    die("Error en la conexion, mensaje de error:" . $ex->getMessage());
}
```

Captura la excepción que lanza PDO debido a que la contraseña era "secreto" y no "1234". El bloque catch muestra el siguiente mensaje:

```
Error en la conexion, mensaje de error: SQLSTATE[HY000] [1045] Access denied for user 'gestor'@'localhost' (using password: YES)
```