UT4: DESARROLLO DE APLICACIONES WEB CON ACCESO A BASES DE DATOS

Néstor Sabater

1. ACCESO A BASES DE DATOS DESDE PHP

- Una de las aplicaciones más frecuentes de PHP es generar una interface web para acceder y gestionar la información almacenada en una base de datos.

- Usando PHP podemos mostrar en una página web información extraída de la base de datos, o enviar sentencias al gestor de la base de datos paraque elimine o actualice algunos registros.

- PHP soporta más de 15 sistemas gestores de bases de datos: SQLite, Oracle, SQL Server, PostgreSQL, IBM DB2, MySQL, etc.

1. ACCESO A BASES DE DATOS DESDE PHP

- 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).

- Es decir, que 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 (PHP Data Object)**.

1. ACCESO A BASES DE DATOS DESDE PHP

- La gran ventaja de PDO está clara: podemos seguir utilizando una misma sintaxis aunque cambiemos el motor de nuestra base de datos.

- Por el contrario, 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.

1. ACCESO A BASES DE DATOS DESDE PHP

- De los distintos SGBD existentes, vamos a aprender a utilizar MySQL.
  - Es el gestor de bases de datos más empleado con el lenguaje PHP.
  - Es la letra "M" que figura en los acrónimos AMP y XAMPP.

- En PHP se utiliza la palabra **new** para crear un nuevo objeto instanciando una clase:

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

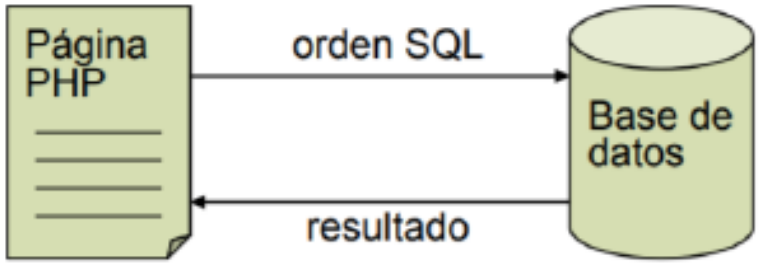
- Y para acceder a los miembros de un objeto, debes utilizar el operador flecha ->:

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

2. LENGUAJE SQL

- SQL (Structured Query Language) es el lenguaje que utilizaremos para comunicarnos con las bases de datos.

- Procedimiento de comunicación con la base de datos:



2. LENGUAJE SQL

- Las instrucciones más habituales son:
  - SELECT
  - INSERT
  - UPDATE
  - DELETE

2. LENGUAJE SQL

- **SENTENCIA SELECT**

  SELECT expresion **FROM** tabla

  [**WHERE** condicion]

  [**ORDER BY** {col_name|formula}[ASC|DESC]…]

  LIMIT [..]

  - Ejemplo: De la tabla de noticias(id, titulo, texto, categoría, fecha), obtener las noticias del día con un tope máximo de 10, ordenadas la más reciente a la más antigua

```
select *
from noticias
where fecha=CURDATE() LIMIT 10 ORDER BY fecha desc;
```

2. LENGUAJE SQL

- **SENTENCIA INSERT**

  INSERT [INTO] nombre_tabla [(nombre_columna,….]

  VALUES ((expresion | default)…)

  INSERT [INTO] nombre_tabla

  SET nombre_columna = (expresion | default), …

```
INSERT INTO noticias (id, titulo,texto,categoria,fecha)
VALUES (37,”Nueva promoción en Las Palmas”,”145 viviendas de lujo”,”promociones”,curdate());
```

2. LENGUAJE SQL

- **SENTENCIA UPDATE**

  UPDATE nombre_tabla [(nombre_columna,…]

  SET nombre_columna1=expres1 [,nombre_columna2=expresion2…]

  [**WHERE** condicion]

  [**ORDER BY**…]

  [**LIMIT** row_count]

  - Ejemplo: De la tabla noticias, modifica la categoría ofertas donde id=37.

```
UPDATE noticias set categoria=”ofertas” where id=37;
```

2. LENGUAJE SQL

- **SENTENCIA DELETE**

  DELETE FROM nombre_tabla

  [**WHERE** condicion]

  [**ORDER BY**…]

  [**LIMIT** row_count]

  - Ejemplo: Borra las noticias con más de 10 días de antigüedad.

```
DELETE FROM noticias WHERE fecha<CURDATE()-10;
```

3. CONEXIÓN A MYSQL

- MySQL dispone de tres APIs para PHP:

    1. La API clásica **mysql**
    2. La API mejorada **mysqli**
    3. La API **PDO** (PHP DATA OBJECT- Objetos de datos de PHP)

- Se consideran activas las dos últimas. Ambos mecanismos de conexión son bastante rápidos.

- Lo más habitual es elegir entre mysqli (extensión nativa) y PDO.

4. PHP DATA OBJECTS (PDO)

- Si vas a programar una aplicación que utilice como sistema gestor de bases de datos MySQL, la extensión MySQLi es una buena opción.

- Pero si en el futuro tienes que cambiar el SGBD por otro distinto, tendrás que volver a programar gran parte del código de la misma.

- El objetivo es que si llegado el momento necesitas cambiar el servidor de base de datos, las modificaciones que debas realizar en tu código sean mínimas.

4. PHP DATA OBJECTS (PDO)

- PDO se basa en las características de orientación a objetos de PHP pero, al contrario que la extensión MySQLi, no ofrece una interface de programación dual.

- Para acceder a las funcionalidades de la extensión tienes que emplear los objetos que ofrece, con sus métodos y propiedades. **No existen funciones alternativas**.

4. PHP DATA OBJECTS (PDO)

- Establecimiento de conexiones
    - Para establecer una conexión con una base de datos utilizando PDO, debes instanciar un objeto de la clase PDO pasándole los siguientes parámetros (solo el primero es obligatorio):
        - 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.
        - Opciones de conexión, almacenadas en forma de array.

4. PHP DATA OBJECTS (PDO)

- Establecimiento de conexiones
  - Por ejemplo, podemos establecer una conexión con la base de datos 'proyecto' creada anteriormente de la siguiente forma:

```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
```

4. PHP DATA OBJECTS (PDO)

- Establecimiento de conexiones

```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
```

  - Si como en el ejemplo, se utiliza el controlador para MySQL, los parámetros específicos para utilizar en la cadena DSN (separadas unas de otras por el carácter punto y coma) a continuación del prefijo "**mysql:**" son los siguientes:

    - **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.
    - **unix_socket**. Socket de MySQL en sistemas Unix.

4. PHP DATA OBJECTS (PDO)

- Establecimiento de conexiones

  - Si quisieras indicar al servidor MySQL que utilice codificación UTF-8 o UTF8mb4 (utf8 con soporte para "emojis“ muy recomendable) para los datos que se transmitan, aunque hay más formas de hacerlo la siguiente es la más sencilla.

```
$dsn = "mysql:host=$host;dbname=$db;charset=utf8mb4";
```

4. PHP DATA OBJECTS (PDO)

- Establecimiento de conexiones
  - Una vez establecida la conexión, puedes utilizar el método **getAttribute** para obtener información del estado de la conexión y **setAttribute** para modificar algunos parámetros que afectan a la misma.
  - Por ejemplo, para obtener la versión del servidor puedes hacer:

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

  - Y si quieres por ejemplo que te devuelva todos los nombres de columnas en mayúsculas:

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

4. PHP DATA OBJECTS (PDO)

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

- Establecimiento de conexiones
    - Y muy importante para controlar los errores tendremos el atributo: **ATTR_ERRMODE** con los posibles 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().

4. PHP DATA OBJECTS (PDO)

- Establecimiento de conexiones
    - Para cerrar la conexión hay que saber que la misma permanecerá activa durante el tiempo de vida del objeto PDO.
    - Para cerrarla, 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.

```php
$conProyecto = null;
```

4. PHP DATA OBJECTS (PDO)

- Ejecución de consultas
  - Para ejecutar una consulta SQL utilizando PDO, debes 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>";
```

4. PHP DATA OBJECTS (PDO)

- Ejecución de consultas
    - Si la consulta genera un conjunto de datos, como es el caso de SELECT, debes utilizar el método query, que devuelve un objeto de la clase PDOStatement.

```php
$host = "localhost";
$db = "proyecto";
$user = "gestor";
$pass = "secreto";
$dsn = "mysql:host=$host;dbname=$db";
$conProyecto=new PDO($dsn, $user, $pass);
$conProyecto->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING);
$resultado = $conProyecto->query("SELECT producto, unidades FROM stock");
```

4. PHP DATA OBJECTS (PDO)

- Ejecución de consultas
    - 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.

4. PHP DATA OBJECTS (PDO)

- Ejecución de consultas
  - Una vez ejecutado un commit o un rollback, se volverá al modo de confirmación automática.

```php
$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
```

4. PHP DATA OBJECTS (PDO)

- Ejecución de consultas
    - Ten en cuenta que no todos los motores no soportan transacciones.
    - Tal es el caso del motor MyISAM de MySQL. En este caso concreto, PDO ejecutará el método beginTransaction sin errores, pero naturalmente no será capaz de revertir los cambios si fuera necesario ejecutar un rollback.

4. PHP DATA OBJECTS (PDO)

```
. . .

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

- Obtención y utilización de conjuntos de resultados.
  - Al igual que con la extensión MySQLi, en PDO tienes 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.

4. PHP DATA OBJECTS (PDO)

```
. . .

$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 />";
}
```

- Obtención y utilización de conjuntos de resultados.
  - Por defecto, 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.
    - **PDO::FETCH_BOUND**. Devuelve true y asigna los valores del registro a variables, según se indique con el método bindColumn . Este método debe ser llamado una vez por cada columna, indicando en cada llamada el número de columna (empezando en 1) y la variable a asignar.

```
$conProyecto = new PDO(". . .");
$resultado = $conProyecto->query("SELECT producto, unidades FROM stocks");
$resultado->bindColumn(1, $producto);
$resultado->bindColumn(2, $unidades);
while ($registro = $resultado->fetch(PDO::FETCH_OBJ)) {
    echo "Producto ".$producto.": ".$unidades."<br />";
}
```

4. PHP DATA OBJECTS (PDO)

- Obtención y utilización de conjuntos de resultados.
    - 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, que podrían provocar problemas de memoria al querer guardar de golpe en un array muchas filas provenientes de una consulta.

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

4. PHP DATA OBJECTS (PDO)

```
. . .

$conProyecto = new PDO(". . .");
$stmt = $conProyecto->prepare('INSERT INTO familia (cod, nombre) VALUES (?, ?)');
```

- Consultas preparadas
  - Al igual que con MySQLi, también utilizando PDO podemos preparar consultas parametrizadas en el servidor para ejecutarlas de forma repetida.
  - El procedimiento es similar e incluso los métodos a ejecutar tienen prácticamente los mismos nombres.
  - Para preparar la consulta en el servidor MySQL, deberás utilizar el método prepare() de la clase PDO.
  - Este método devuelve un objeto de la clase PDOStatement. Los parámetros se pueden marcar utilizando signos de interrogación.

4. PHP DATA OBJECTS (PDO)

- Consultas preparadas
  - O también utilizando parámetros con nombre, precediéndolos por el símbolo de dos puntos.

```php
$stmt = $conProyecto->prepare('INSERT INTO familia (cod, nombre) VALUES (:cod, :nombre)');
```

  - Antes de ejecutar la consulta hay que asignar un valor a los parámetros utilizando el método bindParam de la clase PDOStatement.

  - Si utilizas signos de interrogación para marcar los parámetros, el procedimiento es equivalente al método bindColumn que acabamos de ver.

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

4. PHP DATA OBJECTS (PDO)

- Consultas preparadas
  - Si utilizas parámetros con nombre, debes indicar ese nombre en la llamada a bindParam.

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

  - Una vez preparada la consulta y enlazados los parámetros con sus valores, se ejecuta la consulta utilizando el método execute().

```
$stmt->execute();
```

4. PHP DATA OBJECTS (PDO)

- Consultas preparadas
    - También existe otra forma de pasar valores a los parámetros. Hay un método , que funciona pasando los valores mediante un array, al método execute().

```
$nombre="Monitores";
$codigo="MONI";
$stmt = $conProyecto->prepare('INSERT INTO familia (cod, nombre) VALUES (:cod, :nombre)');
$stmt->execute([ ':cod'=>$codigo, ':nombre'=>$nombre]);
]);
```

4. PHP DATA OBJECTS (PDO)

- Errores y manejo de excepciones
    - 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 estos.
    - Para hacer referencia a cada uno de los niveles de error, PHP define una serie de constantes.
    - Por ejemplo, la constante **E_NOTICE** hace referencia a avisos que pueden indicar un error al ejecutar el guión, y la constante **E_ERROR** engloba errores fatales que provocan que se interrumpa forzosamente la ejecución.

4. PHP DATA OBJECTS (PDO)

- Errores y manejo de excepciones
    - 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 puedes ajustar están:
        - **error_reporting**. Indica qué tipos de errores se notificarán. Su valor se forma utilizando los operadores a nivel de bit para combinar las constantes anteriores. 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.

4. PHP DATA OBJECTS (PDO)

- Errores y manejo de excepciones
  - Desde código, puedes 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 de tu código figura una división en la que exista la posibilidad de que el divisor sea cero, cuando esto ocurra obtendrás un mensaje de error en el navegador.
  - Para evitarlo, puedes 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);
```

4. PHP DATA OBJECTS (PDO)

```php
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 />";
    }
}
```

- Errores y manejo de excepciones
  - Al usar la función error_reporting solo controlas qué tipo de errores va a notificar PHP.
  - A veces puede ser suficiente, pero para obtener más control sobre el proceso existe también la posibilidad de reemplazar la gestión de los mismos por la que tú definas. Es decir, puedes 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).
  - 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).

4. PHP DATA OBJECTS (PDO)

- 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.

4. PHP DATA OBJECTS (PDO)

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

```php
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();
}
```

4. PHP DATA OBJECTS (PDO)

- 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.

4. PHP DATA OBJECTS (PDO)

- Excepciones
    - Las funciones internas de PHP y muchas extensiones como MySQLi usan el sistema de errores visto anteriormente.
    - Solo 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.

4. PHP DATA OBJECTS (PDO)

- Excepciones
  - Vimos en el apartado anterior que la clase PDO permitía definir la fórmula que usará cuando se produzca un error, utilizando el atributo PDO::ATTR_ERRMODE.
  - Las posibilidades eran:
    - PDO::ERRMODE_SILENT.
    - PDO::ERRMODE_WARNING.
    - PDO::ERRMODE_EXCEPTION. Cuando se produce un error lanza una excepción utilizando el manejador propio PDOException.

4. PHP DATA OBJECTS (PDO)

- Excepciones
  - Es decir, que si quieres utilizar excepciones con la extensión PDO, debes configurar la conexión haciendo:

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

  - Por ejemplo, el siguiente código:

```php
$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());
}
```

4. PHP DATA OBJECTS (PDO)

- Excepciones
  - 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)
```

5. RESUMEN

```php
// Variables
$hostDB = '127.0.0.1';
$nombreDB = 'ejemplo';
$usuarioDB = 'root';
$contrasenyaDB = '123';
// Conecta con base de datos
$hostPDO = "mysql:host=$hostDB;dbname=$nombreDB;";
$miPDO = new PDO($hostPDO, $usuarioDB, $contrasenyaDB);
// Prepara SELECT
$miConsulta = $miPDO->prepare('SELECT * FROM Escuelas;');
// Ejecuta consulta
$miConsulta->execute();
// Imprimo
$resultados = $miConsulta->fetchAll();
foreach ($resultados as $posicion => $columna) {
    echo $columna['nombre'];
}
```

5. RESUMEN

- Nos devolverá un array por cada fila, duplicando los datos para poder obtenerlos por posición o por el nombre de la columna.

```
Array
(
    [id] => 1
    [0] => 1
    [school] => Oxford
    [1] => Oxford
    [my_population] => 12345
    [2] => 12345
    [created_at] => 2018-07-31 11:04:04
    [3] => 2018-07-31 11:04:04
)
Array
(
    [id] => 2
    [0] => 2
    [school] => London
    [1] => London
    [my_population] => 76543
    [2] => 76543
    [created_at] => 2018-08-31 10:04:04
    [3] => 2018-08-31 10:04:04
)
...
```

5. RESUMEN

- Los datos se pasan al ejecutar la orden (execute), sustituyendo las claves por los elementos con dos puntos delante.

- Por ejemplo, tenemos :nombre que será sustituido por beethoven.

```php
// Variables
$hostDB = '127.0.0.1';
$nombreDB = 'ejemplo';
$usuarioDB = 'root';
$contrasenyaDB = '123';
// Conecta con base de datos
$hostPDO = "mysql:host=$hostDB;dbname=$nombreDB;";
$miPDO = new PDO($hostPDO, $usuarioDB, $contrasenyaDB);
// Prepara INSERT
$miInsert = $miPDO->prepare('INSERT INTO alumnos (nombre, email, codigo_postal) VALUES (:nombre, :
email, :codigo_postal)');
// Ejecuta INSERT con los datos
$miInsert->execute(
    array(
        'nombre' => 'beethoven',
        'email' => 'beethoven@cuatroestaciones.com',
        'codigo_postal' => '1234'
    )
);
```

5. RESUMEN

- Crear un registro

```html
<form method="post">
    <p>
        <label for="titulo">Titulo</label>
        <input id="titulo" type="text" name="titulo">
    </p>
    <p>
        <label for="autor">Autor</label>
        <input id="autor" type="text" name="autor">
    </p>
    <p>
        <div>¿Disponible?</div>
        <input id="si-disponible" type="radio" name="disponible" value="1" checked> <label for="si-disponible">Si</label>
        <input id="no-disponible" type="radio" name="disponible" value="0"> <label for="no-disponible">No</label>
    </p>
    <p>
        <input type="submit" value="Guardar">
    </p>
</form>
```

5. RESUMEN

- Crear un registro

```php
// Comprobamso si recibimos datos por POST
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    // Recogemos variables
    $titulo = isset($_REQUEST['titulo']) ? $_REQUEST['titulo'] : null;
    $autor = isset($_REQUEST['autor']) ? $_REQUEST['autor'] : null;
    $disponible = isset($_REQUEST['disponible']) ? $_REQUEST['disponible'] : null;
    // Variables
    $hostDB = '127.0.0.1';
    $nombreDB = 'ejemplo';
    $usuarioDB = 'root';
    $contrasenyaDB = '';
    // Conecta con base de datos
    $hostPDO = "mysql:host=$hostDB;dbname=$nombreDB;";
    $miPDO = new PDO($hostPDO, $usuarioDB, $contrasenyaDB);
    // Prepara INSERT
    $miInsert = $miPDO->prepare('INSERT INTO libros (titulo, autor, disponible) VALUES (:titulo, :
autor, :disponible)');
    // Ejecuta INSERT con los datos
    $miInsert->execute(
        array(
            'titulo' => $titulo,
            'autor' => $autor,
            'disponible' => $disponible
        )
    );
    // Redireccionamos a Leer
    header('Location: leer.php');
}
```

5. RESUMEN

- Crear un registro
  - Todo unido…

php

```php
<?php
// Comprobamso si recibimos datos por POST
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    // Recogemos variables
    $titulo = isset($_REQUEST['titulo']) ? $_REQUEST['titulo'] : null;
    $autor = isset($_REQUEST['autor']) ? $_REQUEST['autor'] : null;
    $disponible = isset($_REQUEST['disponible']) ? $_REQUEST['disponible'] : null;
    // Variables
    $hostDB = '127.0.0.1';
    $nombreDB = 'ejemplo';
    $usuarioDB = 'root';
    $contrasenyaDB = '';
    // Conecta con base de datos
    $hostPDO = "mysql:host=$hostDB;dbname=$nombreDB;";
    $miPDO = new PDO($hostPDO, $usuarioDB, $contrasenyaDB);
    // Prepara INSERT
    $miInsert = $miPDO->prepare('INSERT INTO libros (titulo, autor, disponible) VALUES (:titulo, :
autor, :disponible)');
    // Ejecuta INSERT con los datos
    $miInsert->execute(
        array(
            'titulo' => $titulo,
            'autor' => $autor,
            'disponible' => $disponible
        )
    );
    // Redireccionamos a Leer
    header('Location: leer.php');
}
?>
```

5. RESUMEN

- Crear un registro
  - Todo unido…

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Crear - CRUD PHP</title>
</head>
<body>
    <form action="" method="post">
        <p>
            <label for="titulo">Titulo</label>
            <input id="titulo" type="text" name="titulo">
        </p>
        <p>
            <label for="autor">Autor</label>
            <input id="autor" type="text" name="autor">
        </p>
        <p>
            <div>¿Disponible?</div>
            <input id="si-disponible" type="radio" name="disponible" value="1" checked> <label for
="si-disponible">Si</label>
            <input id="no-disponible" type="radio" name="disponible" value="0"> <label for="no-dis
ponible">No</label>
        </p>
        <p>
            <input type="submit" value="Guardar">
        </p>
    </form>
</body>
</html>
```