

# Desarrollo Web en Entorno Servidor

7.- PHP – Bases de Datos

IES Severo Ochoa

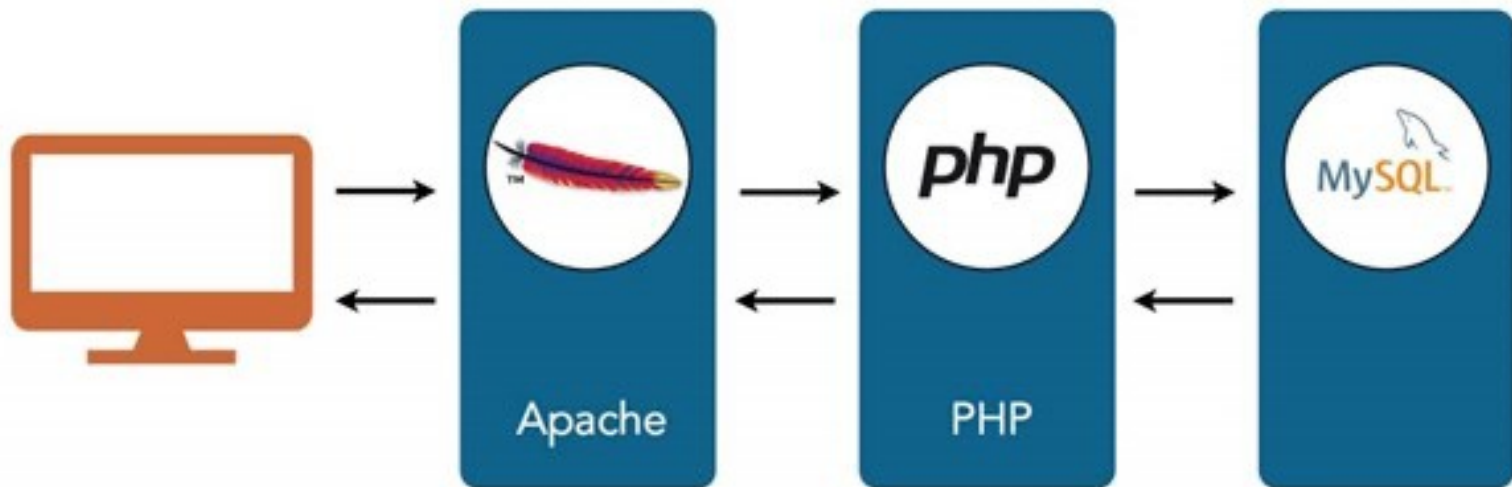


# Índice

- *mysqli*
- PDO
  - Sentencias preparadas
  - Recogiendo datos
- Patrones
- Transacciones
- Autenticación de usuarios

# Arquitectura de 3 niveles

## Handling a Web Request



# MySQL / MariaDB

- SGBD software libre más implantado.
- Forma parte de XAMPP
- Herramienta de administración → *PhpMyAdmin*
- SQL
  - *DDL: create database, create table*
    - *show datases, show tables, describe*
  - *DML: insert, update, delete / select*
- Cliente MariaDB (root/root):
  - > `mariadb -u root -p`

# PHP y las BBDD

- <https://www.php.net/manual/es/refs.database.vendors.php>
- PHP ofrece dos formas de acceder a MySQL.
  - *mysqli*: interfaz procedural (*legacy*)
    - Sólo para MySQL / MariaDB
  - *PDO*: orientado a objetos
    - Abstrae el SGBD
- Otra alternativa es utilizar un *framework* ORM (*Object Relational Mapping*)
  - *Doctrine*
  - *Eloquent* (Laravel)

- <https://www.php.net/manual/es/book.mysqli.php>
- Extensión mejorada
- Programación dual (procedural y objetos)
  - Por ejemplo, para crear una conexión:

```
// utilizando constructores y métodos de la P00
$conexion1 = new mysqli('localhost', 'dwes', 'abc123.', 'dwes');
echo $conexion1->server_info;

echo "<br>";

// utilizando llamadas a funciones
$conexion2 = mysqli_connect('localhost', 'dwes', 'abc123.', 'dwes');
echo mysqli_get_server_info($conexion2);
```

# mysqli: operación DML

```
$conexion = mysqli_connect('localhost', 'dwes', 'abc123.', 'dwes');
$error = $conexion->connect_errno;
if ($error == null) {
    $sql='DELETE FROM stock WHERE unidades=0';
    $resultado = $conexion->query($sql);
    if ($resultado) {
        echo "Se ha borrado $conexion->affected_rows registros"; }
    }
$conexion->close();
```

# mysqli: consultas I

- Recupera los registros como array:

```
$conexion = mysqli_connect('localhost', 'dwes', 'abc123.', 'dwes');
$error = $conexion->connect_errno;
if ($error == null) {
    $resultado = $conexion->query('SELECT producto, unidades
                                   FROM stock WHERE unidades<2');

    $stock = $resultado->fetch_array();
    // Obtenemos el primer registro
    $producto = $stock['producto']; // 0 también $stock[0];
    $unidades = $stock['unidades']; // 0 también $stock[1];
    echo "<p>Producto $producto: $unidades unidad/es.</p>";
}
$conexion->close();
```



# mysqli: consultas II

- Recupera los registros como objeto:

```
$conexion = mysqli_connect('localhost', 'dwes', 'abc123.', 'dwes');
$error = $conexion->connect_errno;
if ($error == null) {
    $resultado = $conexion->query('SELECT producto, unidades
                                   FROM stock WHERE unidades<2');

    //Recorre los resultados
    while ($stock = $resultado->fetch_object()) {
        print "<p>Producto $stock->producto: $stock->unidades u.</p>";
    }
}
$conexion->close();
```

# PDO



- PHP Data Objects
- <https://www.php.net/manual/es/book.pdo.php>
- Interfaz Orientada a objetos
  - Cada SGBD implementa el interfaz con un driver.
- Desacopla las operaciones de DML del SGDB
- La conexión se realiza mediante una fuente de datos (*datasource*)
  - Cadena con toda la información de conexión (controlador: parámetros)
  - ```
$conexion = new PDO('mysql:host=localhost;  
dbname=dwes', 'dwes', 'abc123.');
```

# Excepciones en PDO

- PDO puede utilizar las excepciones para gestionar los errores → encapsular código PDO entre `try/catch`
- Modos disponibles:
  - `PDO::ERRMODE_SILENT` -> Es el modo por defecto. Aquí tendremos que chequear los errores usando `->errorCode()` y `->errorInfo()`.
  - `PDO::ERRMODE_WARNING` -> Genera errores *warning* PHP pero permitiría la ejecución normal de la aplicación.
  - `PDO::ERRMODE_EXCEPTION` -> Recomendada. Dispara una excepción permitiéndonos gestionar el error de forma amigable.
    - Cualquier fallo con la BBDD lanza **`PDOException`**

# Creando la conexión y excepciones

```
<?php
const DSN = "mysql:host=localhost;dbname=dwes";
const USUARIO = "dwes";
const PASSWORD = "abc123.";

$conexion = null;

// 1.- Abrimos la conexión
try {
    $conexion = new PDO(DSN, USUARIO, PASSWORD);
    // Forzamos a gestionar los errores como excepciones
    $conexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // 2.- Operamos con la base de datos
    $version = $conexion->getAttribute(PDO::ATTR_SERVER_VERSION);
    echo $version;
} catch (PDOException $e){
    echo $e->getMessage();
}

// 3.- Liberamos la conexión
$conexion = null;
```

# Fichero de configuración I

- Es conveniente dejar los datos de conexión en un archivo separado
  - No se sube a *Git*
- Método I: Colocar los datos como constantes
- `config/configuracion.inc.php`

```
<?php
const DSN = "mysql:host=localhost;dbname=dwes";
const USUARIO = "dwes";
const PASSWORD = "abc123.";
```

# PDO: Borrado

```
<?php
include "config/database.inc.php";

$conexion = null;

try {
    $conexion = new PDO(DSN, USUARIO, PASSWORD);
    $conexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // exec devuelve la cantidad de tuplas afectadas
    $cantidadAfectada =
        $conexion->exec("DELETE FROM stock WHERE unidades=0");
    echo $cantidadAfectada;
} catch (PDOException $e){
    echo $e->getMessage();
}

$conexion = null;
```

# Sentencias preparadas

- Siempre hemos de utilizar sentencias preparadas
  - Evita *SQL Injection* y mejoran el rendimiento

1) *[prepare]* Prepara la sentencia, obteniendo una `PDOStatement`

- `$sentencia = $conexion -> prepare($sql)`

2) *[bind]* Pasar parámetros

- Mediante ?
- Mediante `:param`
  - Uso de array asociativo
  - Uso de `bindParam (referencia) / bindValue (valor)`



3) *[execute]* Ejecutar la sentencia

- `$sentencia->execute([datos]);`
- `$sentencia->execute();`

# PDO: Borrado con parámetros I

```
<?php
include "config/database.inc.php";

$conexion = null;

try {
    $cantidad = $_GET["cantidad"];

    $conexion = new PDO(DSN, USUARIO, PASSWORD);
    $conexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $sql = "DELETE FROM stock WHERE unidades = " + $cantidad;
    $sentencia = $conexion->prepare($sql);
    $isOk = $sentencia->execute();

    $cantidadAfectada = $sentencia->rowCount();
    echo $cantidadAfectada;
} catch (PDOException $e){
    echo $e->getMessage();
}

$conexion = null;
```





# PDO: Borrado con parámetros II

```
<?php
include "config/database.inc.php";

$conexion = null;

try {
    $cantidad = $_GET["cantidad"];

    $conexion = new PDO(DSN, USUARIO, PASSWORD);
    $conexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // preparamos la consulta: los parámetros se indican con ?
    $sql = "DELETE FROM stock WHERE unidades = ?";
    $sentencia = $conexion->prepare($sql);
    $isOk = $sentencia->execute( [$cantidad] ); // le pasamos los parámetros

    $cantidadAfectada = $sentencia->rowCount();
    echo $cantidadAfectada;
} catch (PDOException $e){
    echo $e->getMessage();
}

$conexion = null;
```



# PDO: Borrado con parámetros II

```
<?php
include "config/database.inc.php";

$conexion = null;

try {
    $cantidad = $_GET["cantidad"] ?? 0;

    $conexion = new PDO(DSN, USUARIO, PASSWORD);
    $conexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // se asignan nombre a los parámetros
    $sql = "DELETE FROM stock WHERE unidades = :cant";
    $sentencia = $conexion->prepare($sql);
    // le pasamos un array asociativo
    $isOk = $sentencia->execute(["cant" => $cantidad]);

    $cantidadAfectada = $sentencia->rowCount();
    echo $cantidadAfectada;
} catch (PDOException $e){
    echo $e->getMessage();
}

$conexion = null;
```



# PDO: Borrado con bindParam

```
<?php
include "config/database.inc.php";

$conexion = null;
try {
    $cantidad = $_GET["cantidad"] ?? 0;

    $conexion = new PDO(DSN, USUARIO, PASSWORD);
    $conexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // se asignan nombre a los parámetros
    $sql = "DELETE FROM stock WHERE unidades = :cant";
    $sentencia = $conexion->prepare($sql);
    // le asociamos los datos
    $sentencia->bindParam(":cant", $cantidad);
    $isOk = $sentencia->execute();

    $cantidadAfectada = $sentencia->rowCount();
    echo $cantidadAfectada;
} catch (PDOException $e){
    echo $e->getMessage();
}

$conexion = null;
```



# bindParam VS bindValue

```
// se asignan nombre a los parámetros
$sql = "DELETE FROM stock WHERE unidades = :cant";
$sentencia = $conexion->prepare($sql);
// bindParam enlaza por referencia
$cantidad = 0;
$sentencia->bindParam(":cant", $cantidad);
$cantidad = 1;
$isOk = $sentencia->execute(); // se eliminan con cant = 1
// bindValue enlaza por valor
$cantidad = 0;
$sentencia->bindValue(":cant", $cantidad);
$cantidad = 1;
$isOk = $sentencia->execute(); // se eliminan con cant = 0
```

- `bindValue()` → cuando se tienen que insertar datos sólo una vez
- `bindParam()` → cuando se tienen que pasar datos múltiples (desde un array por ejemplo).
- Además aceptan un tercer parámetro con el tipo de dato `PDO::PARAM_X`:  
<https://www.php.net/manual/es/pdo.constants.php>

# Inserción de registros

- Si tenemos un campo autoincrementable, dejamos el campo vacío, y tras la inserción:
  - `$conexion->lastInsertId()` ;

```
$nombre = $_GET["nombre"] ?? "SUCURSAL X";
```

```
$telefono = $_GET["telefono"] ?? "636123456";
```

```
$sql = "INSERT INTO tienda(nombre, tlf) VALUES (:nombre, :telefono)";
```

```
$sentencia = $conexion->prepare($sql);
```

```
$sentencia->bindParam(":nombre", $nombre);
```

```
$sentencia->bindParam(":telefono", $telefono);
```

```
$isOk = $sentencia->execute();
```

```
$idGenerado = $conexion->lastInsertId();
```

```
echo $idGenerado;
```

# Consulta de registros

- **`PDOStatement::fetch`** → obtiene la siguiente fila de un conjunto de resultados.
- Antes de llamar a *fetch* (o durante) hay que especificar cómo se quieren devolver los datos:
  - **`PDO::FETCH_ASSOC`**: array indexado cuyos keys son el nombre de las columnas.
  - **`PDO::FETCH_NUM`**: array indexado cuyos keys son números.
  - **`PDO::FETCH_BOTH`**: valor por defecto. Devuelve un array indexado cuyos keys son tanto el nombre de las columnas como números.

# Diferencias entre FETCH

|                               |                                |                 |               |
|-------------------------------|--------------------------------|-----------------|---------------|
| <b>Consulta</b>               | <b>SELECT * FROM artículos</b> |                 |               |
| <b>Columnas</b>               | <b>id</b>                      | <b>articulo</b> | <b>precio</b> |
| <b>1a línea del resultado</b> | 1                              | Albaricoques    | 35.5          |

Resultado de un fetch (lectura con diferentes funciones)

| <b>PDO::FETCH_NUM</b> |              | <b>PDO::FETCH_ASSOC:</b> |              | <b>PDO::FETCH_BOTH</b> |              |
|-----------------------|--------------|--------------------------|--------------|------------------------|--------------|
| <b>Clave</b>          | <b>Valor</b> | <b>Clave</b>             | <b>Valor</b> | <b>Clave</b>           | <b>Valor</b> |
| <b>0</b>              | 1            | <b>id</b>                | 1            | <b>id</b>              | 1            |
| <b>1</b>              | Albaricoques | <b>articulo</b>          | Albaricoques | <b>0</b>               | 1            |
| <b>2</b>              | 35.5         | <b>precio</b>            | 35.5         | <b>texto</b>           | Albaricoques |
|                       |              |                          |              | <b>1</b>               | Albaricoques |
|                       |              |                          |              | <b>precio</b>          | 35.5         |
|                       |              |                          |              | <b>2</b>               | 35.5         |

# Consulta con array asociativo

```
<?php
include "config/database.inc.php";

$conexion = null;

try {
    $conexion = new PDO(DSN, USUARIO, PASSWORD);
    $conexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $sql = "select * from tienda";
    $sentencia = $conexion->prepare($sql);
    $sentencia->setFetchMode(PDO::FETCH_ASSOC);
    $sentencia->execute();

    while ($fila = $sentencia->fetch()) {
        echo "Codigo:". $fila["cod"]. "<br />";
        echo "Nombre:". $fila["nombre"]. "<br />";
        echo "Teléfono:". $fila["tlf"]. "<br />";
    }
} catch (PDOException $e){
    echo $e->getMessage();
}

$conexion = null;
```



# Consulta con fetchAll

- En vez de ir fila a fila, podemos recuperar una matriz con el resultado
- `PDOStatement::fetchAll()`

```
$sql = "select * from tienda";  
$sentencia = $conexion->prepare($sql);  
$sentencia->setFetchMode(PDO::FETCH_ASSOC);  
$sentencia->execute();  
  
$tiendas = $sentencia->fetchAll();  
  
foreach ($tiendas as $tienda) {  
    echo "Codigo:". $tienda["cod"]. "<br />";  
    echo "Nombre:". $tienda["nombre"]. "<br />";  
}
```

# Consultas con objetos

- Si queremos leer los datos con formato de objeto  
→ `PDO::FETCH_OBJ`
- Se crea un objeto con propiedades públicas nombradas igual que las columnas

```
$sql = "select * from tienda";  
$sentencia = $conexion->prepare($sql);  
$sentencia->setFetchMode(PDO::FETCH_OBJ);  
$sentencia->execute();  
  
while ($t = $sentencia->fetch()) {  
    echo "Codigo:". $t->cod. "<br />";  
    echo "Nombre:". $t->nombre. "<br />";  
    echo "Teléfono:". $t->tlf. "<br />";  
}
```

# Consultas con modelos I

- Si queremos usar una de nuestras clases, ya sean del modelo o un objeto de transferencia (*transfer object*) → `PDO::FETCH_CLASS`
- Los nombres de los atributos privados deben coincidir con los nombres de las columnas.

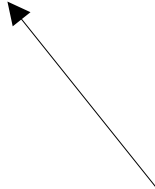
```
class Tienda {  
    private int $cod;  
    private string $nombre;  
    private ?string $tlf;  
  
    public function getCodigo() : int {  
        return $this->cod;  
    }  
  
    public function getNombre() : string {  
        return $this->nombre;  
    }  
  
    public function getTelefono() : ?string {  
        return $this->tlf;  
    }  
}
```

# Consultas con modelos II

```
$sql = "select * from tienda";  
$sentencia = $conexion->prepare($sql);  
$sentencia->setFetchMode(PDO::FETCH_CLASS, "Tienda");  
$sentencia->execute();
```

```
while ($t = $sentencia->fetch()) {  
    echo "Codigo:". $t->getCodigo(). "<br />";  
    echo "Nombre:". $t->getNombre(). "<br />";  
    echo "Teléfono:". $t->getTelefono(). "<br />";  
    var_dump($t);  
}
```

Nombre de  
la clase a  
mapear



- Más ejemplos con el uso de consultas y objetos en: <https://phpdelusions.net/pdo/objects>

# Consultas con modelos III

- Si nuestras clases tienen constructores (deberían), al asignar el método fetch hay que indicarle que rellene las propiedades después de llamar al constructor:

`PDO::FETCH_PROPS_LATE`

```
$sql = "select * from tienda";  
$sentencia = $conexion->prepare($sql);  
$sentencia->setFetchMode(PDO::FETCH_CLASS |  
                        PDO::FETCH_PROPS_LATE, Tienda::class);  
$sentencia->execute();  
$tiendas = $sentencia->fetchAll();
```

# Consultas con LIKE

- Los comodines se ponen en el dato a asociar, no en la propia consulta.

```
$sql = "select * from tienda where nombre like :nombre or tlf like :tlf";  
$sentencia = $conexion->prepare($sql);  
$sentencia->setFetchMode(PDO::FETCH_CLASS | PDO::FETCH_PROPS_LATE,  
                           Tienda::class);  
$cadBuscar = "%".$busqueda."%";  
$sentencia->execute(["nombre" => $cadBuscar, "tlf" => $cadBuscar]);  
  
$result = $sentencia->fetchAll();
```

# Depurando consultas

- Es normal que no estemos seguro de qué consulta se esté enviando al SGBD.
- Tras ejecutar la consulta, a partir de la sentencia:  
`$sentencia->debugDumpParams ()`

```
SQL: [63] select * from tienda where nombre like :nombre  
or tlf like :tlf
```

```
Sent SQL: [68] select * from tienda where nombre like  
'%UCUR%' or tlf like '%UCUR%'
```

```
Params: 2
```

```
Key: ...
```

- El acceso a datos es una de la áreas donde más se utilizan los patrones de diseño.
  - *DAO (Data Access Object)* – abstrae los datos de la capa de negocio / un DAO por entidad
  - *Repository*: similar a DAO, pero se plantea a menor granularidad / por funcionalidad
  - *Factory (method)*: aísla la creación de conexiones/elementos en una clase/método
  - *Entity*: agrupa los atributos que se mapean con la base de datos
  - *Transfer Object / Value Object*: Almacén de datos, sin lógica, se envían desde los datos a la vista



# Fichero de configuración II

- Metemos los datos en un *array* y los devolvemos
- Se puede asignar una variable a un include
- `config/configuracion.php`

```
<?php
return [
    'db' => [
        'name' => 'dwes',
        'username' => 'dwes',
        'password' => 'abc123.',
        'connection' => 'mysql:host=localhost',
        'dsn' => 'mysql:host=localhost;dbname=dwes',
        'options' => [
            PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8",
            PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
            PDO::ATTR_PERSISTENT => true
        ]
    ]
];
```

# Factoría PDO

- Colocamos todo el código de creación en un método estático:

```
<?php

class PDOFactory {

    public static function getConexion() : PDO {

        $config = include "config/configuracion.php";

        $conexion = new PDO($config["db"]["dsn"],
            $config["db"]["username"],
            $config["db"]["password"],
            $config["db"]["options"]);

        return $conexion;
    }
}
```

- Interfaz con las operaciones comunes

```
interface TiendaRepository {  
    public function getAll(): ?array;  
    public function getTiendaPorCodigo(int $cod): ?Tienda;  
    public function insert(string $nombre, string $telefono): Tienda;  
    public function delete(int $cod);  
}
```

```
class MDBTiendaRepository implements TiendaRepository {  
    private $log;  
  
    public function __construct(LoggerInterface $logger) {  
        $this->log = $logger;  
    }  
  
    public function getAll(): ?array {  
        $conexion = null;  
        $result = null;  
  
        try {  
            $conexion = PDOFactory::getConnection();  
        }  
    }  
}
```

# Usando el *Repository*

- Este código se colocaría en el *Controller / Action* (próxima unidad)
  - Ahora lo dejaremos dentro de nuestras operaciones de negocio en nuestro modelo.

```
$tienda = null;  
try {  
    $repositorio = new MDBTiendaRepository(LogFactory::getLogger());  
    $tienda = $repositorio->insert($nombre, $telefono);  
    $mensaje = "La operación se ha realizado con éxito";  
} catch (TiendaException) {  
    $mensaje = "Ha ocurrido un error al crear la tienda";  
}
```

- Hasta ahora teníamos en nuestras clases de modelo tanto los atributos de las clases/entidades como sus operaciones.
- Vamos a colocar los atributos y sus *getter/setter* en clases de tipo *Entity*
  - Por ejemplo `Tienda`
- Y sus operaciones (*alta, baja, prestar, alquilar*) de momento las colocaremos en un servicio
  - Por ejemplo, `TiendaService`

# TiendaService

```
class TiendaService {  
  
    private TiendaRepository $rep;  
  
    public function __construct()  
    {  
        $repositorio = new MDBTiendaRepository(LogFactory::getLogger());  
        $this->rep = $repositorio;  
    }  
  
    public function crearTienda(string $nombre, string $telefono) : Tienda {  
        return $this->rep->insert($nombre, $telefono);  
    }  
  
    public function buscar(string $cadena) {  
        return $this->rep->search($cadena);  
    }  
}
```

- Por defecto PDO trabaja en modo *autocommit*
  - 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

# Ejemplo transacciones

```
<?php
$conexion = null;

try {
    $conexion = PDOFactory::getConexion();

    $conexion->beginTransaction();

    $sql = "DELETE FROM stock WHERE unidades=0";
    $sentencia = $conexion->prepare($sql);
    $sentencia->execute();

    $sql = "UPDATE stock SET unidades=unidades+1 WHERE tienda=1";
    $sentencia = $conexion->prepare($sql);
    $sentencia->execute();

    $conexion->commit();

} catch (PDOException $e){
    echo $e->getMessage();
    $conexion->rollBack();
}

$conexion = null;
```

Si hay algún fallo, se deshacen las operaciones realizadas



# Login/Password

- Para almacenar las contraseñas: `password_hash()`
- Al autenticar al usuario verificar la contraseña con `password_verify()`
  - Recuperamos el usuario por su *login*
  - Comprobamos que coincidan las *hash* de las contraseñas

```
$usu = $_POST["login"] ?? "";
```

```
$sql = "select * from usuarios where usuario = ?";
```

```
$sentencia = $conexion->prepare($sql);
```

```
$sentencia->execute( [$usu] );
```

```
$usuario = $sentencia->fetch();
```

```
if ( $usuario && password_verify($_POST['pass'], $usuario['password']) ) {  
    echo "OK!";  
} else {  
    echo "KO";  
}
```

# Almacenando usuarios

- `password_hash` permite diferentes algoritmos de HASH:
  - `PASSWORD_DEFAULT`: algoritmo `bcrypt`
  - `PASSWORD_BCRYPT`: algoritmo `CRYPT_BLOWFISH`, compatible con `crypt()`

```
$usu = $_GET["usuario"];  
$pas = $_GET["password"];
```

```
$sql = "INSERT INTO usuarios(usuario, password)  
VALUES (:usuario, :password)";  
$sentencia = $conexion->prepare($sql);  
$isOk = $sentencia->execute([  
    "usuario" => $usu,  
    "password" => password_hash($pas, PASSWORD_DEFAULT)]);
```

¿Alguna pregunta?