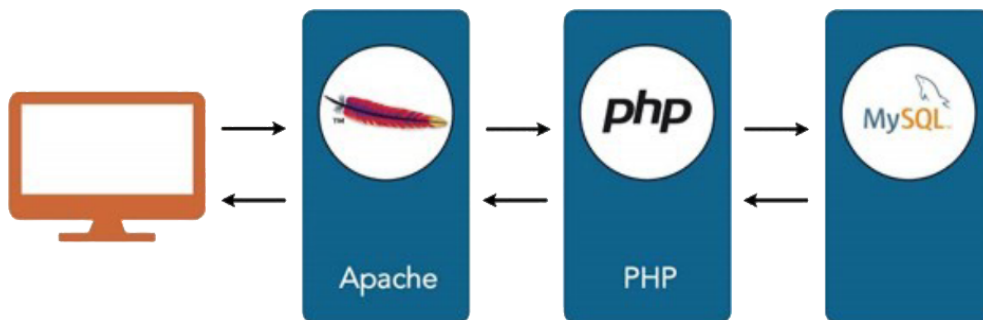


unidad didáctica 6

Acceso a datos



1. **duración y criterios de evaluación**
2. **instalación**
3. **estructura de una base de datos**
4. **CholloSevero**
5. **SQL**
6. **phpMyAdmin**
 6. 1. [Creando una base de datos dentro de phpMyAdmin](#)
 6. 2. [opciones en phpMyAdmin](#)
7. **MySQLi**
 7. 1. [recuperando datos de una BD](#)
8. **PHP Data Objects :: PDO**
 8. 1. [fichero de configuración de la BD](#)
 8. 2. [sentencias preparadas](#)
 8. 3. [ejemplo parámetros](#)
 8. 4. [ejemplo bindParam](#)
 8. 5. [bindValue VS bindParam](#)
 8. 6. [insertando registros](#)
 8. 7. [consultando registros](#)
 8. 8. [consultas con modelos](#)
 8. 9. [consultas con LIKE](#)
9. **login & password**
10. **acceso a ficheros**
 10. 1. [modos de apertura de ficheros](#)
 10. 2. [operaciones con archivos](#)
 10. 3. [información de un fichero](#)
 10. 4. [archivos PDF](#)
11. **referencias**

1. duración y criterios de evaluación

En esta unidad vamos a aprender a acceder a datos que se encuentran en un servidor; recuperando, editando y creando dichos datos a través de una base de datos.

A través de las distintas capas o niveles, de las cuales 2 de ellas ya conocemos (*Apache, PHP*) y *MySQL* la que vamos a estudiar en este tema.

Duración estimada: 12 sesiones

Resultado de aprendizaje y criterios de evaluación:

RA6. Desarrolla aplicaciones de acceso a almacenes de datos, aplicando medidas para mantener la seguridad y la integridad de la información.

a) Se han analizado las tecnologías que permiten el acceso mediante programación a la información disponible en almacenes de datos.

b) Se han creado aplicaciones que establezcan conexiones con bases de datos.

c) Se ha recuperado información almacenada en bases de datos.

d) Se ha publicado en aplicaciones web la información recuperada.

e) Se han utilizado conjuntos de datos para almacenar la información.

f) Se han creado aplicaciones web que permitan la actualización y la eliminación de información disponible en una base de datos.

g) Se han probado y documentado las aplicaciones web.

2. instalación

XAMPP: simplemente nos descargaríamos el programa y lo activaríamos. Para descargar XAMPP [pulsa aquí](#).

Docker: utilizando el software de contenedores nos descargaremos [esta imagen de docker](#) y lanzamos:

```
1 | docker-compose up -d
```

Si todo ha salido bien y el contenedor está en marcha, podremos visitar la página de phpMyAdmin de la siguiente manera

```
1 | http://localhost:8000
```



Para acceder debemos utilizar las siguientes credenciales que vienen configuradas en el archivo `docker-compose.yml`:

```
1 | usuario: root
2 | contraseña: 1234
```

3. estructura de una base de datos

Echando un vistazo al módulo de primer curso *Bases de Datos* sabemos que una base de datos tiene muchos campos con sus nombres y valores; pero además sabemos que la base de datos debe tener un nombre. Por tanto, tendríamos la siguiente estructura para una base de datos:

```

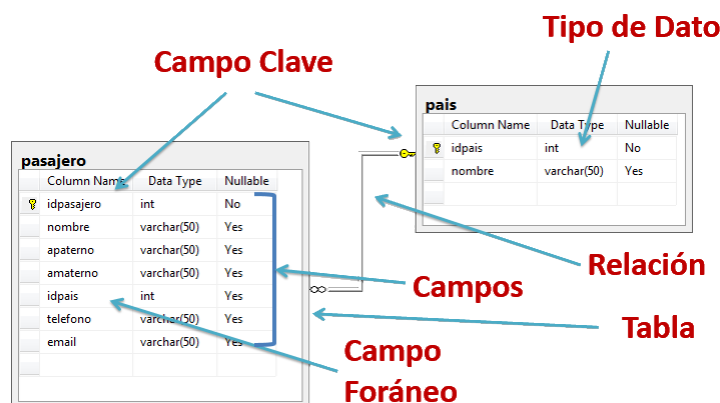
1  NombreBaseDeDatos
2      |__Tabla-#1
3      |          |__DatosTabla-#1
4      |
5      |__Tabla-#2
6      |          |__DatosTabla-#2
7      |
8      |__Tabla-#3
9      |          |__DatosTabla-#3
10     [...]
    
```

Veámoslo en un ejemplo real:

```


1  Ryanair
2      |__pasajero
3      |      |__id[*]
4      |      |__nombre
5      |      |__apellidos
6      |      |__edad
7      |      |__id_vuelo[^]
8      |
9      |__vuelo
10     |      |__id[*]
11     |      |__n_plazas
12     |      |__disponible
13     |      |__id_pais[^]
14     |
15     |__pais
16         |__id[*]
17         |__nombre
    
```


[*] Clave primaria [^] Clave Foránea






4. CholloSevero


A lo largo de esta unidad vamos a trabajar con una base de datos que iremos confeccionando conforme avancemos; donde almacenaremos la información relacionada con ofertas que publiquen los usuarios y los listaremos en función de varios filtros, nuevos, más votados, más vistos, más comentados entre otros, al más puro estilo [Chollometro](#).



Portátil LENOVO IdeaPad 3 15.6" - Intel Core i7-1165G7 - 8 GB RAM - 512 GB SSD PCIe - Intel Iris Xe Graphics
579€ 699€ (-17%)  Envío gratuito | Ofertas MediaMarkt


Buen precio en Mediamarkt, precio final al tramitar el pedido con el descuento directo de 100€. Portátil - Lenovo IdeaPad 3 15ITL6, 15.6" FHD, Intel® Core™ i7-1165G7, 8 GB RAM, 512 GB SSD PCIe, Windows 11... [Leer más](#)


 Destrangis   23 [Ir al chollo](#)




Peluches solidarios de Carrefour (para enfermedades infantiles) por solo 3'50€
3,50€ | Ofertas Carrefour



Peluches solidarios de Carrefour, para combatir enfermedades raras infantiles. Recordar el chollo es ser solidario <3

 Samarkando   36 [Ir al chollo](#)




Barra de sonido LG SN7Y, Subwoofer inalámbrico, 380 W, Dolby Atmos, Meridian Sound, ASC, Bass Blast+
234€ 399€ (-41%)  Envío gratuito | Ofertas MediaMarkt

Descuento de 25€ al añadir al carrito. Con la barra de sonido LG SN7Y de color negro escucharás tus canciones favoritas en una calidad de audio impresionante gracias a sus tecnologías. [Leer más](#)

 TheGamerKap   26 [Ir al chollo](#)

hace 2 h, 19 m

hace 2 h, 51 m



¡CELEBRA LA NAVIDAD CON HASTA UN 50% DTO. EN HUAWEI!*

Categorías populares

- Portátiles
- Consola PS5
- Consola Xbox S...
- Televisores
- Smartphones y ...

Tiendas populares

5. SQL

Utilizaremos el lenguaje de consulta estructurada (*Structured Query Language*) SQL para realizar las consultas a nuestras bases de datos y para mostrar el contenido en las distintas interfaces web que creemos a lo largo de la unidad. Si quieres saber más detalles visita [Wiki SQL](#).

Ejemplo de una sentencia SQL donde seleccionamos todas las filas y columnas de nuestra tabla llamada `pais`:

```
1 | SELECT * FROM pais
```

Estas sentencias pueden invocarse desde la consola de comandos mediante el intérprete `mysql*` (previamente instalado en el sistema) o a través de la herramienta `phpMyAdmin`.

Las sentencias SQL también las podemos usar dentro de nuestro código php; de tal manera que cuando se cargue nuestra interfaz web, lance una sentencia SQL para mostrar los datos que queramos.

```
1 | <?php
2 |     // Listado de clientes, ordenados por DNI de manera ASCendente
3 |     $clientesOrdenadosPorDNI = "SELECT * FROM `pasajero` ORDER BY `dni` ASC";
4 | ?>
```

6. phpMyAdmin

Este software funciona bajo Apache y PHP y es una interfaz web para gestionar las bases de datos que tengamos disponibles en nuestro servidor local. Muchos *hostings* ofrecen esta herramienta por defecto para poder gestionar las BBDD que tengamos configuradas bajo nuestra cuenta.



6.1. Creando una base de datos dentro de phpMyAdmin

1. Para crear una nueva base de datos debemos entrar en phpMyAdmin como **usuario root** y pinchar en la opción **Nueva** del menú de la izquierda.
2. En la nueva ventana de creación pondremos un **nombre** a nuestra bd.
 1. También estableceremos el **cotejamiento** `utf8mb4_unicode_ci` para que nuestra bd soporte todo tipo de caracteres (como los asiáticos e incluso emojis 😊).
3. Le damos al botón de **Crear** para crear la bd y comenzar a crear las distintas tablas que vayamos a introducir en ella.



El sistema generará el código SQL para crear todo lo que le hemos puesto y creará la base de datos con las tablas que le hayamos insertado.

```
1 CREATE TABLE `persona` ( `id` INT NOT NULL AUTO_INCREMENT , `nombre` TINYTEXT NOT NULL , `apellidos` TEXT NOT NULL , `telefono` TINYTEXT NOT NULL , PRIMARY KEY (`id`)) ENGINE = InnoDB;
```

6.2. opciones en phpMyAdmin

Cuando seleccionamos una base de datos de la lista, el sistema nos muestra varias pestañas con las cuales interactuar con la base de datos en cuestión:

- **Estructura** : podemos ver las distintas tablas que consolidan nuestra base de datos.
- **SQL** : si queremos inyectar código SQL para que el sistema lo interprete.
- **Buscar** : para buscar por términos, en nuestra base de datos, aplicando distintos filtros de búsqueda.

- **Generar consulta** : parecido a SQL pero de forma gráfica, sin tener que saber nada del lenguaje.
- **Exportar** e **importar** : como su nombre indica, para hacer cualquiera de las 2 operaciones sobre la base de datos.
- **Operaciones** : distintas opciones avanzadas para realizar en nuestra base de datos, de la cual destacaremos la opción **Cotejamiento** donde podremos cambiar el cotejamiento de nuestra tabla .

OJO CON ÉSTO porque podemos eliminar datos sin querer, ya que al cambiar el cotejamiento podemos suprimir caracteres no soportados por el nuevo cotejamiento.

No vamos a profundizar en el resto de opciones, pero, en la pestaña **Más**, existe la opción **Diseñador** para poder editar las relaciones entre tablas de una manera gráfica (pinchando y arrastrando) que veremos más adelante.

7. MySQLi

PHP hace uso de esta extensión mejorada para poder comunicarse con las bases de datos, ya sean MySQL (4.1 o superior) o MariaDB.

Cualquier consulta que queramos hacer a una base de datos necesitaremos hacer uso de la extensión `mysqli()`.

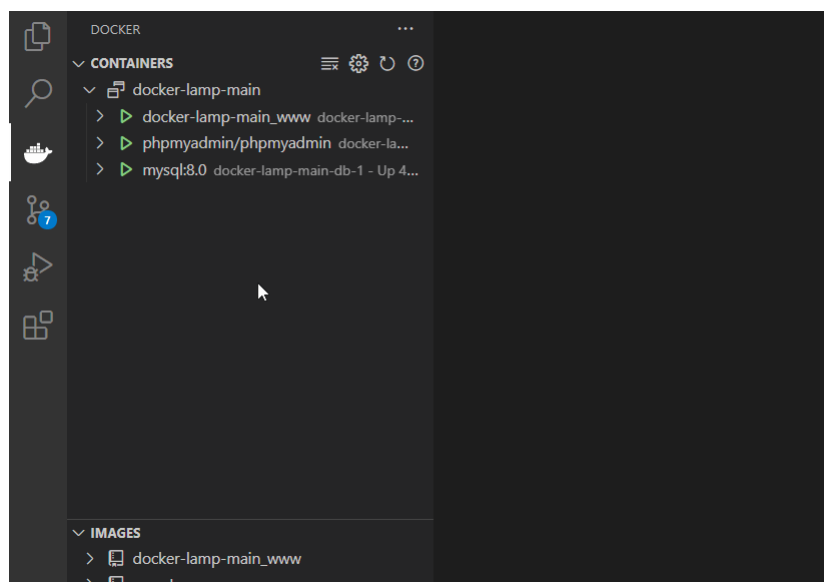
Veamos cómo conectarnos con una base de datos a través del código PHP:

```

1  <?php
2      // pruebas.php
3
4      // "SERVIDOR", "USUARIO", "CONTRASEÑA", "BASE DE DATOS"
5      $conexion =
mysqli_connect("d939ebf6a741", "tuUsuario", "tuContraseña", "tuBaseDeDatos");
6
7      // COMPROBAMOS LA CONEXIÓN
8      if(mysqli_connect_errno()) {
9          echo "Failed to connect to MySQL: " . mysqli_connect_error();
10         exit();
11     }
12
13     echo "<h1>Bienvenid@ a MySQL !!</h1>";
14 ?>

```

- `servidor`: el servidor donde se encuentra la base de datos que queremos usar suele ser **localhost**, pero en nuestro caso, al utilizar Docker será el nombre de la imagen **mysql:8.0** que aparece al dejar el ratón encima en el Visual Studio Code.



- `usuarioDB`: el usuario de la base de datos.
- `passwordDB`: la contraseña para ese usuario de la base de datos.
- `baseDeDatos`: nombre de la base de datos que queramos usar.

Si todo ha salido bien habréis visto un mensaje diciendo **Bienvenid@ a MySQL !!**

7.1. recuperando datos de una BD

Ahora que ya sabemos cómo conectarnos a una base de datos alojada en nuestro servidor, lo que necesitamos saber es cómo recuperar datos almacenados en la base de datos.

Durante la instalación de la imagen de Docker, se ha creado una tabla llamada **Pruebas** que contiene varios registros de distintas personas.

Vamos a recuperar esos datos para ver de qué forma se hace con PHP.

```

1  <?php
2      // pruebas.php
3
4      $conexion = mysqli_connect("d939ebf6a741", "usuario", "1234", "pruebadb");
5
6      // COMPROBAMOS LA CONEXIÓN
7      if (mysqli_connect_errno()) {
8          echo "Failed to connect to MySQL: " . mysqli_connect_error();
9          exit();
10     }
11
12     // CONSULTA A LA BASE DE DATOS
13     $consulta = "SELECT * FROM `productos`";
14     $listaproductos = mysqli_query($conexion, $consulta);
15
16     // COMPROBAMOS SI EL SERVIDOR NOS HA DEVUELTO RESULTADOS
17     if($listaproductos) {
18         // RECORREMOS CADA RESULTADO QUE NOS DEVUELVE EL SERVIDOR
19         foreach ($listaproductos as $elemento) {
20             echo "$elemento[id] -
21                 $elemento[descripcion] -
22                 $elemento[stock]
23                 <br>
24                 ";
25         }
26     }
27     ?>

```

Si todo ha salido bien, por pantalla verás el siguiente listado:

```

1  http://localhost/pruebas.php
2
3  1 leche 25
4  2 pan 12
5  3 galletas 5
6  4 gominolas 120
7  5 monster 2
8  6 kit kat 17
9  7 patatas fritas 7
10 8 donetes 5

```

8. PHP Data Objects :: PDO

De la misma manera que hemos visto con `mysqli`, PHP Data Objects (o `PDO`) es un driver de PHP que se utiliza para trabajar bajo una interfaz de objetos con la base de datos. A día de hoy, es lo que más se utiliza para manejar información desde una base de datos, ya sea relacional o no relacional.

De igual manera que pasaba con los objetos en PHP nativos, en la interfaz de MySQL el momento de la conexión es distinta:

```
1 <?php
2     $conexion = new PDO('mysql:host=localhost; dbname=pruebadb', 'usuario',
    '1234');
```

en docker

Recordar cambiar *localhost* por ID del contenedor `mysql:8.0`.

Además, con PDO podemos usar las excepciones con `try/catch` para gestionar los errores que se produzcan en nuestra aplicación. Para ello, como hacíamos antes, debemos encapsular el código entre bloques `try/catch`:

```
1 <?php
2     $dsn = 'mysql:dbname=pruebadb; host=127.0.0.1'; // 127.0.01 -> ID
    contenedor mysql:8.0
3     $usuario = 'usuario';
4     $contrasena = '1234';
5
6     try {
7         $mbd = new PDO($dsn, $usuario, $contrasena);
8         $mbd->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
9     } catch (PDOException $e) {
10         echo 'Fallo en la conexión: ' . $e->getMessage();
11     }
```

Los pasos a seguir han sido:

1º) Creamos la conexión con la base de datos a través del constructor PDO pasándole la información de la base de datos.

2º) Establecemos los parámetros para manejar las excepciones, en este caso hemos utilizado:

- `PDO::ATTR_ERRMODE` indicándole a PHP que queremos un reporte de errores.
- `PDO::ERRMODE_EXCEPTION` con este atributo obligamos a que lance excepciones, además de ser la opción más humana y legible que hay a la hora de controlar errores.

3ª) Cualquier error que se lance a través de PDO, el sistema lanzará una **PDOException**.

8.1. fichero de configuración de la BD

De la misma manera que creamos nuestro archivo de funciones `funciones.php` y albergamos todas las funciones que se usan de manera global en la aplicación, podemos establecer un archivo de constantes donde definamos los parámetros de conexión con la base de datos.

```

1  <?php
2
3      // conexion.php
4      const DSN = "mysql:host=localhost; dbname=pruebadb"; // localhost -> ID
      contenedor mysql:8.0
5      const USUARIO = "usuario";
6      const PASSWORD = "1234";
7
8      /*
9       * NO SUBAS ESTE ARCHIVO A git
10      */

```

OJO Este archivo contiene información **muy sensible** así que no es recomendable que subas este archivo a git.

8.2. sentencias preparadas

Se trata de sentencias que se establecen como si fueran plantillas de la SQL que vamos a lanzar, aceptando parámetros que son establecidos a posteriori de la declaración de la sentencia preparada.

Las sentencias preparadas evitan la **inyección de SQL** (SQL Injection) y mejoran el rendimiento de nuestras aplicaciones o páginas web.

```

1  <?php
2      $sql = "INSERT INTO productos VALUES (?, ?, ?)";

```

Cada interrogante es un parámetro que estableceremos después unas cuantas líneas más abajo.

Una vez tenemos la plantilla de nuestra consulta, debemos seguir con la preparación junto con 3 métodos más de PHP para su completa ejecución:

1º) `prepare`: prepara la sentencia antes de ser ejecutada.

2º) `bind`: el tipo de unión (*bind*) de dato. Puede ser mediante `?` o `:parametro`.

3º) `execute`: se ejecuta la consulta uniendo la plantilla con las variables o parámetros que hemos establecido.

8.3. ejemplo parámetros

```

1  <?php
2      // Borrando con parámetros
3      include "config/database.inc.php";
4
5      $conexion = null;
6

```

```

7      try {
8          $cantidad = $_GET["cantidad"];
9
10         $conexion = new PDO(DSN, USUARIO, PASSWORD);
11         $conexion -> setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
12
13         $sql = "DELETE FROM productos WHERE stock = ?";
14         $sentencia = $conexion -> prepare($sql);
15
16         $isOk = $sentencia -> execute([$cantidad]);
17         $cantidadAfectada = $sentencia -> rowCount();
18
19         echo $cantidadAfectada . " fila(s) afectada(s).";
20
21     } catch (PDOException $e) {
22         echo $e -> getMessage();
23     }
24
25     $conexion = null;

```

8.4. ejemplo bindParam

Muy parecido a utilizar parámetros pero esta vez la variable está dentro de la sentencia SQL, en este caso la hemos llamado `:cant`

```

1  <?php
2      include "config/database.inc.php";
3
4      $conexion=null;
5
6      try {
7          $cantidad = $_GET["cantidad"] ?? 0;
8
9          $conexion = new PDO(DSN, USUARIO, PASSWORD);
10         $conexion -> setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
11
12         $sql = "DELETE FROM productos WHERE stock = :cant";
13
14         $sentencia = $conexion -> prepare($sql);
15         $sentencia -> bindParam(":cant", $cantidad);
16
17         $isOk = $sentencia -> execute();
18
19         $cantidadAfectada = $sentencia -> rowCount();
20
21         echo $cantidadAfectada . " fila(s) afectada(s).";
22
23     } catch (PDOException $e) {
24         echo $e -> getMessage();
25     }
26
27     $conexion = null;

```

8.5. bindValue VS bindParam

Utilizaremos `bindValue()` cuando tengamos que insertar datos sólo una vez. En cambio, deberemos usar `bindParam()` cuando tengamos que pasar datos múltiples, como por ejemplo, un array.

```

1  <?php
2      // se asignan nombre a los parametros
3      $sql = "DELETE FROM productos WHERE stock = :cant";
4      $sentencia = $conexion -> prepare($sql);
5
6      // bindValue enlaza por VALOR
7      $cantidad = 0;
8      $sentencia -> bindValue(":cant", $cantidad);
9      $cantidad = 1;
10     // se eliminan con cant = 0
11     $isOk = $sentencia->execute();
12
13     // bindParam enlaza por REFERENCIA
14     $cantidad = 0;
15     $sentencia -> bindParam(":cant", $cantidad);
16     $cantidad = 1;
17     // se eliminan con cant = 1
18     $isOk = $sentencia -> execute();

```

Para más información y uso de las variables PDO [consulta el manual de PHP](#).

8.6. insertando registros

A la hora de insertar registros en una base de datos, debemos tener en cuenta que en la tabla puede haber valores autoincrementados. Para salvaguardar esto, lo que debemos hacer es dejar vacío ese campo autoincrementado; pero a la hora de hacer la conexión, debemos recuperarlo con el método `lastInsertId()`.

```

1  <?php
2      include "config/database.inc.php";
3
4      try{
5          $descripcion = $_GET["descripcion"] ?? "PRODUCTO X";
6          $stock = $_GET["stock"] ?? 0;
7
8          $conexion = null;
9
10         $conexion = new PDO(DSN, USUARIO, PASSWORD);
11         $conexion -> setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
12
13         $sql="insert into productos (descripcion, stock) values (:desc,
14         :stock)";
15
16         $sentencia = $conexion -> prepare($sql);
17         $sentencia -> bindParam(":desc", $descripcion);
18         $sentencia -> bindParam(":stock", $stock);
19
20         $isOk = $sentencia -> execute();

```

```

20         $idGenerado = $conexion -> lastInsertId();
21
22     } catch(PDOException $e){
23         echo $e->getMessage();
24     }

```

8.7. consultando registros

A la hora de recuperar los resultados de una consulta, bastará con invocar al método `PDOStatement::fetch` para listar las filas generadas por la consulta.

Pero debemos elegir el tipo de dato que queremos recibir entre los 3 que hay disponibles:

- `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.

Consulta	SELECT * FROM artículos		
Columnas	id	articulo	precio
1a línea del resultado	1	Albaricoques	35.5

Resultado de un fetch (lectura con diferentes funciones)

PDO::FETCH_NUM		PDO::FETCH_ASSOC:		PDO::FETCH_BOTH	
Clave	Valor	Clave	Valor	Clave	Valor
0	1	id	1	id	1
1	Albaricoques	articulo	Albaricoques	0	1
2	35.5	precio	35.5	texto	Albaricoques
				1	Albaricoques
				precio	35.5
				2	35.5

```

1  <?php
2      // consulta con array asociativo.php
3      include "config/database.inc.php";
4
5      $conexion = null;
6
7      try{
8          $conexion = new PDO(DSN, USUARIO, PASSWORD);
9          $conexion -> setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
10
11         $sql = "select * from productos";
12
13         $sentencia = $conexion -> prepare($sql);
14         $sentencia -> setFetchMode(PDO::FETCH_ASSOC);
15         $sentencia -> execute();
16
17         while($fila = $sentencia -> fetch()){
18             echo "Id:" . $fila["id"] . "<br />";
19             echo "Descripción:" . $fila["descripcion"] . "<br />";
20             echo "Stock:" . $fila["stock"] . "<br /><br />";
21         }
22

```



```

23     }catch(PDOException $e) {
24         echo $e -> getMessage();
25     }
26
27     $conexion = null;

```

Recuperando datos con una **matriz** como resultado de nuestra consulta:

```

1  <?php
2      // consulta con array asociativo
3      include "config/database.inc.php";
4
5      $conexion = null;
6
7      try{
8          $conexion = new PDO(DSN, USUARIO, PASSWORD);
9          $conexion -> setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
10
11          $sql="SELECT * FROM productos";
12
13          $sentencia = $conexion -> prepare($sql);
14          $sentencia -> setFetchMode(PDO::FETCH_ASSOC);
15          $sentencia -> execute();
16          // recuperado en la matriz $productos
17          $productos = $sentencia -> fetchAll();
18
19          foreach($productos as $elemento) {
20              echo "Id:" . $elemento["id"] . "<br />";
21              echo "Descripción:" . $elemento["descripcion"] . "<br />";
22              echo "Stock:" . $elemento["stock"] . "<br /><br />";
23          }
24
25      }catch(PDOException $e) {
26          echo $e -> getMessage();
27      }
28
29      $conexion = null;

```

Pero si lo que queremos es leer datos con forma de objeto utilizando `PDO::FETCH_OBJ`, debemos crear un objeto con propiedades públicas con el mismo nombre que las columnas de la tabla que vayamos a consultar.

```

1  <?php
2      // consulta con formato de objeto
3      include "config/database.inc.php";
4
5      $conexion = null;
6
7      try{
8          $conexion = new PDO(DSN, USUARIO, PASSWORD);
9          $conexion -> setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
10
11          $sql="SELECT * FROM productos";
12

```

```

13     $sentencia = $conexion -> prepare($sql);
14     $sentencia -> setFetchMode(PDO::FETCH_OBJ);
15     $sentencia -> execute();
16
17     while($p = $sentencia -> fetch()) {
18         echo "Id:" . $p -> id . "<br />";
19         echo "Descripción:" . $p -> descripcion . "<br />";
20         echo "Stock:" . $p -> stock . "<br />";
21     }
22
23     }catch(PDOException $e) {
24         echo $e -> getMessage();
25     }
26
27     $conexion = null;

```

8.8. consultas con modelos

Llevamos tiempo creando clases en PHP y las consultas también admiten este tipo de datos mediante el uso de `PDO::FETCH_CLASS`.

Si usamos este método, debemos tener en cuenta que los nombres de los atributos privados deben coincidir con los nombres de las columnas de la tabla que vayamos a manejar.

Así pues, si por lo que sea cambiamos la estructura de la tabla **DEBEMOS CAMBIAR** nuestra clase para que todo siga funcionando.

```

1  <?php
2      // clase Producto
3      class Producto {
4          private int $id;
5          private string $descripcion;
6          private ? int $stock;
7
8          public function getId() : int {
9              return $this -> id;
10         }
11
12         public function getDescripcion() : string {
13             return $this -> descripcion;
14         }
15
16         public function getStock() : ?int {
17             return $this -> stock;
18         }
19     }
20     ?>
21     <?php
22     // Consultando a través de la clase Producto
23     $sql = "SELECT * FROM productos";
24     $sentencia = $conexion -> prepare($sql);
25
26     // Aquí 'Tienda' es el nombre de nuestra clase
27     $sentencia -> setFetchMode(PDO::FETCH_CLASS, "Producto");
28     $sentencia -> execute();

```

```

29
30     while($p = $sentencia -> fetch()) {
31         echo "Id: " . $p -> getId() . "<br />";
32         echo "Descripcion: " . $p -> getDescripcion() . "<br />";
33         echo "Stock: " . $p -> getStock() . "<br /><br />";
34
35         //var_dump($p);
36     }

```

Pero ¿qué pasa si nuestras clases tienen constructor? pues que debemos indicarle, al método FETCH, que rellene las propiedades después de llamar al constructor y para ello hacemos uso de `PDO::FETCH_PROPS_LATE`.

```

1  <?php
2      // clase Producto
3      class Producto {
4          public function __construct (public int $id=1,
5                                      public string $descripcion="x",
6                                      public int $stock=1 ){ }
7
8          public function getId() : int {
9              return $this -> id;
10         }
11         public function getDescripcion() : string {
12             return $this -> descripcion;
13         }
14         public function getStock() : int {
15             return $this -> stock;
16         }
17     }
18     ?>
19     <?php
20         // consulta con array asociativo.php
21         include "config/database.inc.php";
22         $conexion = null;
23         try{
24             $conexion = new PDO(DSN, USUARIO, PASSWORD);
25             $conexion -> setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
26             $sql = "SELECT * FROM productos";
27
28             $sentencia = $conexion -> prepare($sql);
29             $sentencia -> setFetchMode(PDO::FETCH_CLASS|PDO::FETCH_PROPS_LATE,
30 "Producto");
31
32             $sentencia -> execute();
33
34             // $productos = $sentencia -> fetchAll();
35             while($p = $sentencia -> fetch()) {
36                 echo "Id: " . $p -> id . "<br />";
37                 echo "Descripción: " . $p -> descripcion . "<br />";
38                 echo "Stock: " . $p -> stock . "<br /><br />";
39             }
40         }catch(PDOException $e) {
41             echo $e -> getMessage();
42         }

```

```

42
43     $conexion = null;

```

8.9. consultas con LIKE

Para utilizar el comodín LIKE u otros comodines, debemos asociarlo al dato y NUNCA en la propia consulta.

```

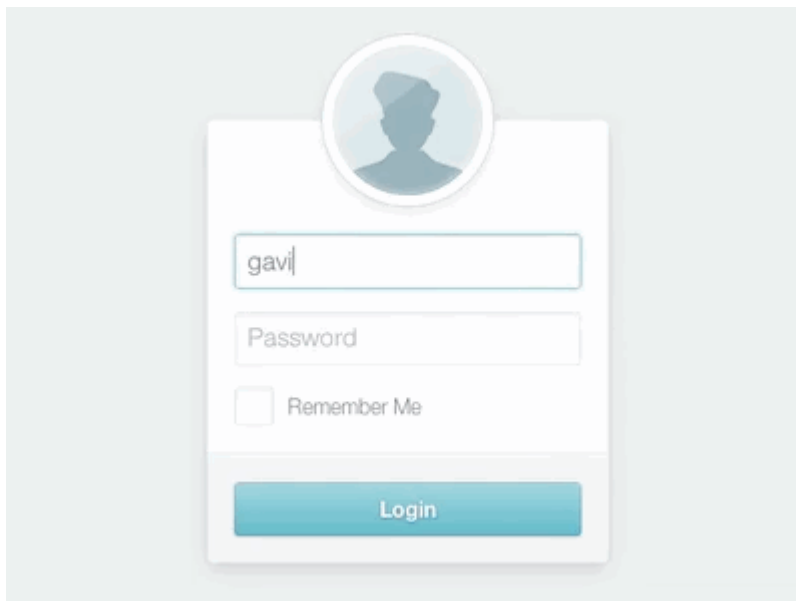
1  <?php
2      //      clase Producto
3      class Producto {
4          public function __construct (public int $id=1,
5                                      public string $descripcion="x",
6                                      public int $stock=1 ){ }
7
8          public function getId() : int {
9              return $this -> id;
10         }
11         public function getDescripcion() : string {
12             return $this -> descripcion;
13         }
14         public function getStock() : int {
15             return $this -> stock;
16         }
17     }
18     ?>
19     <?php
20         $busqueda1 = $_POST["cadenaDescripcion"];
21         $busqueda2 = $_POST["numStock"];
22
23         //      Utilizando comodines :: LIKE
24         include "config/database.inc.php";
25
26         $conexion = null;
27         try{
28             $conexion = new PDO(DSN, USUARIO, PASSWORD);
29             $conexion -> setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
30             $sql = "SELECT * FROM productos where descripcion like :desc and
31 stock = :stk";
32
33             $sentencia = $conexion -> prepare($sql);
34             $sentencia -> setFetchMode(PDO::FETCH_CLASS | PDO::FETCH_PROPS_LATE,
35 Producto::class);
36
37             $cadBuscar1 = "%" . $busqueda1 . "%";
38
39             $sentencia -> execute(["desc" => $cadBuscar1, "stk" => $cadBuscar2]);
40
41             while($p = $sentencia -> fetch()) {
42                 echo "Id:" . $p -> id . "<br />";
43                 echo "Descripción:" . $p -> descripcion . "<br />";
44                 echo "Stock:" . $p -> stock . "<br /><br />";
45             }
46         }catch(PDOException $e) {

```

```
46         echo $e -> getMessage();  
47     }  
48  
49     $conexion = null;
```

Tenéis una lista de ejemplos muy completa en la [documentación oficial](#).

9. login & password



Para manejar un sistema completo de login y password con contraseñas cifradas, necesitamos un método que cifre esos strings que el usuario introduce como contraseña; tanto en el formulario de registro como en el del login, ya que al codificar una contraseña, después tenemos que decodificarla para comprobar que ambas contraseñas (la que introduce el usuario en el login y la que tenemos en la base de datos) coincidan.

Necesitamos pues:

1º) `password_hash()` para almacenar la contraseña en la base de datos a la hora de hacer el INSERT.

- `PASSWORD_DEFAULT` almacenamos la contraseña usando el método de encriptación *bcrypt*.
- `PASSWORD_BCRYPT` almacenamos la contraseña usando el algoritmo `CRYPT_BLOWFISH` compatible con `crypt()`.

2º) `password_verify()` para verificar el usuario y la contraseña.

Ejemplo:

```

1  <form action="loginCrear.php" method="post">
2      <label for="usuario">usuario: </label>
3      <input type="text" name="usuario">
4      <label for="password">password: </label>
5      <input type="password" name="password">
6
7      <input type="submit" name="enviar" value="enviar">
8  </form>

```

```

1  <?php
2      // loginCrear.php
3      include "config/database.inc.php";
4
5      $conexion = null;

```

```

6      try{
7          $conexion = new PDO(DSN, USUARIO, PASSWORD);
8          $conexion -> setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
9
10         $usu = $_POST["usuario"];
11         $pas = $_POST["password"];
12         $fechaActual = date("Y-m-d H:i:s");
13         $cre = $fechaActual;
14         $acc = $fechaActual;
15
16         $sql = "INSERT INTO usuarios (usuario, password, created, access)
17                 VALUES (:usu, :pas, :cre, :acc)";
18
19         $sentencia = $conexion -> prepare($sql);
20         $isOk = $sentencia -> execute([
21             "usu" => $usu,
22             "pas" => password_hash($pas, PASSWORD_DEFAULT),
23             "cre" => $cre,
24             "acc" => $acc
25         ]);
26         // $idGenerado = $conexion -> lastInsertId();
27
28     } catch (PDOException $e) {
29         echo $e -> getMessage();
30     }
31
32     $conexion = null;

```

Ahora que tenemos el usuario codificado y guardado en la base de datos, vamos a recuperarlo para poder logearlo correctamente.

```

1  <?php
2      // Recuperando usuario y password en BD
3      include "config/database.inc.php";
4      $conexion = null;
5      try{
6          $conexion = new PDO(DSN, USUARIO, PASSWORD);
7          $conexion -> setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
8
9          $usu = $_POST["usuario"] ?? "";
10         $sql = "select * from usuarios where usuario = ?";
11
12         $sentencia = $conexion -> prepare($sql);
13         $sentencia -> execute([$usu]);
14
15         $usuario = $sentencia -> fetch();
16
17         if($usuario && password_verify($_POST['password'],
18 $usuario['password'])) {
19             echo "OK!";
20         } else {
21             echo "KO";
22         }
23     } catch (PDOException $e){
24         echo $e->getMessage();

```

```
24     }  
25     $conexion = null;
```


10. acceso a ficheros

Gracias a la función `fopen()` desde PHP podemos abrir archivos que se encuentren en nuestros servidor o una URL.

A esta función hay que pasarle 2 parámetros; el **nombre del archivo** que queremos abrir **y** el **modo** en el que se abrirá:

```
1 $fp = fopen("miarchivo.txt", "r");
```

Muchas veces no podemos abrir el archivo porque éste no se encuentra o no tenemos acceso a él; por eso es recomendable comprobar que podemos hacerlo:

```
1 if (!$fp = fopen("miarchivo.txt", "r")){
2     echo "No se ha podido abrir el archivo";
3 }
```

10.1. modos de apertura de ficheros

- `r` : Modo lectura. Puntero al principio del archivo.
- `r+` : Apertura para lectura y escritura. Puntero al principio del archivo.
- `w` : Apertura para escritura. Puntero al principio del archivo y lo sobrescribe. Si no existe se intenta crear.
- `w+` : Apertura para lectura y escritura. Puntero al principio del archivo y lo sobrescribe. Si no existe se intenta crear.
- `a` : Apertura para escritura. Puntero al final del archivo. Si no existe se intenta crear.
- `a+` : Apertura para lectura y escritura. Puntero al final del archivo. Si no existe se intenta crear.
- `x` : Creación y apertura para sólo escritura. Puntero al principio del archivo. Si el archivo ya existe dará error E_WARNING. Si no existe se intenta crear.
- `x+` : Creación y apertura para lectura y escritura. Mismo comportamiento que x.
- `c` : Apertura para escritura. Si no existe se crea. Si existe no se sobrescribe ni da ningún error. Puntero al principio del archivo.
- `c+` : Apertura para lectura y escritura. Mismo comportamiento que C.
- `b` : Cuando se trabaja con archivos binarios como jpg, pdf, png y demás. Se suele colocar al final del modo, es decir rb, r+b, x+b, wb...

10.2. operaciones con archivos

Para poder **leer** un archivo necesitamos usar la función `fread()` de PHP:

```

1 // Abriendo un archivo y leyendo su contenido
2 $file = "miarchivo.txt";
3 $fp = fopen($file, "r");
4
5 // $contents guardará el contenido
6 // filesize() nos devuelve el tamaño del archivo en cuestión
7 $contents = fread($fp, filesize($file));
8 print $contents;
9
10 // Cerramos la conexión con el archivo
11 fclose($fp)

```

Si lo que queremos es **escribir** en un archivo, deberemos hacer uso de la función `fwrite()` :

```

1 // Escribiendo en un archivo
2 $file = "miarchivo.txt";
3 $texto = "Hola qué tal";
4
5 $fp = fopen($file, "w");
6
7 fwrite($fp, $texto);
8 fclose($fp);

```

10.3. información de un fichero

Con PHP y su método `stat()` podemos obtener información sobre los archivos que le indiquemos. Este método devuelve hasta un total de 12 elementos con información acerca de nuestro archivo.

1	0	dev	número de dispositivo
2	1	ino	número de i-nodo
3	2	mode	modo de protección del i-nodo
4	3	nlink	número de enlaces
5	4	uid	ID de usuario del propietario
6	5	gid	ID de grupo del propietario
7	6	rdev	tipo de dispositivo, si es un dispositivo i-nodo
8	7	size	tamaño en bytes
9	8	atime	momento del último acceso (tiempo Unix)
10	9	mtime	momento de la última modificación (tiempo Unix)
11	10	ctime	momento de la última modificación del i-nodo (tiempo Unix)
12	11	blksize	tamaño del bloque E/S del sistema de ficheros
13	12	blocks	número de bloques de 512 bytes asignados

Unos ejemplos...

```

1 <?php
2 // Información del archivo
3
4 $file = "miarchivo.txt";
5 $texto = "Todos somos muy ignorantes, lo que ocurre es que no todos
    ignoramos las mismas cosas.";
6
7 $fp = fopen($file, "w");

```

```

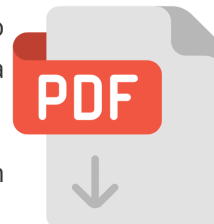
8     fwrite($fp, $texto);
9
10    $datos = stat($file);
11
12    echo $datos[3] . "<br>"; // Número de enlaces, 1
13    echo $datos[7] . "<br>"; // Tamaño en bytes, 85
14    echo $datos[8] . "<br>"; // Momento de último acceso, 1444138104
15    echo $datos[9] . "<br>"; // Momento de última modificación, 1444138251
16    ?>

```

Echa un vistazo a [las funciones de directorios](#) que tiene PHP, es muy interesante.

10.4. archivos PDF

Con PHP podemos manejar todo tipo de archivos como ya hemos visto pero, ¿qué pasa si queremos generar ficheros PDF con datos sacados de una base de datos?



Gracias a una clase escrita en PHP, podemos generar archivos PDF sin necesidad de instalar librerías adicionales en nuestro servidor.

Para ello, como tenemos composer dentro de nuestra imagen de Docker, usaremos **composer** para instalar esta dependencia.

Acuérdate que debemos haber hecho `composer init` para empezar un proyecto con composer, de lo contrario no podrás añadir ningún paquete.

Veamos un ejemplo de `Hello World` convertido a PDF:

```

1  <?php
2      ob_end_clean();
3      require('fpdf/fpdf.php');
4
5      // Instanciamos la clase
6      // P = Portrait | mm = unidades en milímetros | A4 = formato
7      $pdf = new FPDF('P', 'mm', 'A4');
8
9      // Añadimos una página
10     $pdf->AddPage();
11
12     // Establecemos la fuente y el tamaño de letra
13     $pdf->SetFont('Arial', 'B', 18);
14
15     // Imprimimos una celda con el texto que nosotros queramos
16     $pdf->Cell(60,20,'Hello World!');
17
18     // Terminamos el PDF
19     $pdf->Output();
20     ?>

```

Hay muchos ejemplos y tutoriales, así como documentación de la clase FPDF en la página oficial.

Visita [la sección de tutoriales y el manual](#) para sacar mayor partido a esta clase.

```

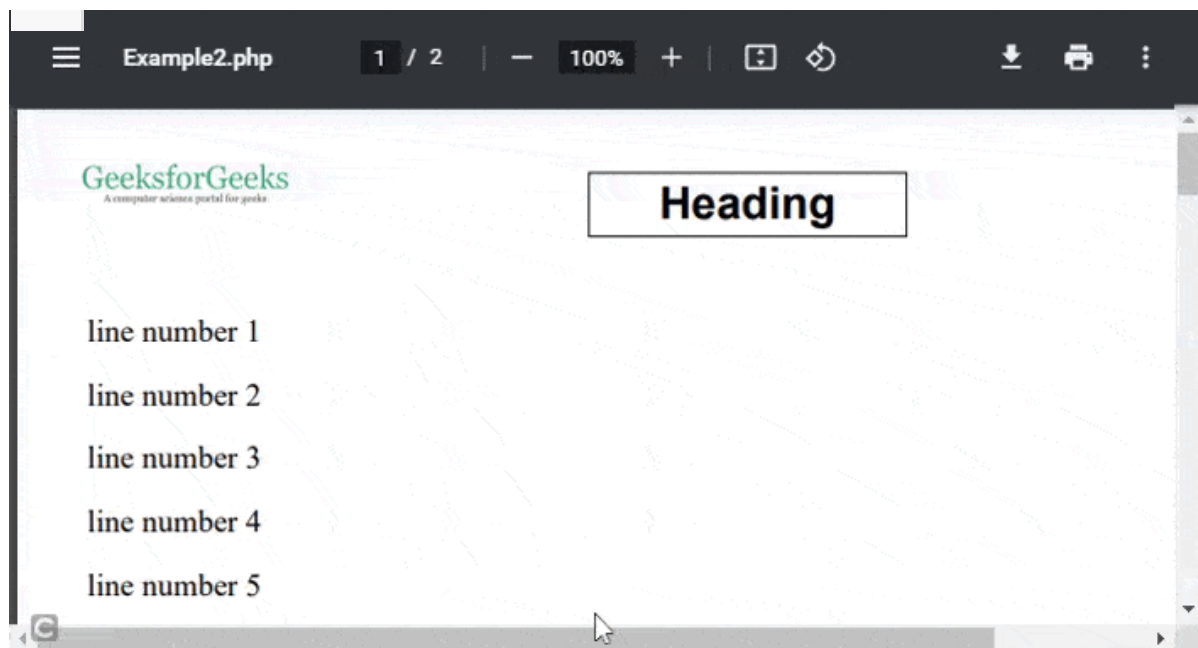
1  <?php

```

```

2
3 require('fpdf/fpdf.php');
4
5 class PDF extends FPDF {
6
7     // Cabecera
8     function Header() {
9
10         // Añadimos un logotipo
11         $this->Image('logo.png',10,8,33);
12
13         // establecemos la fuente y el tamaño
14         $this->SetFont('Arial','B',20);
15
16         // Movemos el contenido un poco a la derecha
17         $this->Cell(80);
18
19         // Pintamos la celda
20         $this->Cell(50,10,'Cabecera',1,0,'C');
21
22         // Pasamos a la siguiente línea
23         $this->Ln(20);
24     }
25
26     // Pie de página
27     function Footer() {
28
29         // Nos posicionamos a 1.5 cm desde abajo del todo de la página
30         $this->SetY(-15);
31
32         // Arial italic 8
33         $this->SetFont('Arial','I',8);
34
35         // Número de página
36         $this->Cell(0,10,'Página ' .
37             $this->PageNo() . '/{nb}',0,0,'C');
38     }
39 }
40
41 // Instanciamos la clase
42 $pdf = new PDF();
43
44 // Definimos un alias para la numeración de páginas
45 $pdf->AliasNbPages();
46
47 $pdf->AddPage();
48 $pdf->SetFont('Times','',14);
49
50 for($i = 1; $i <= 30; $i++)
51     $pdf->Cell(0, 10, 'Número de línea '
52         . $i, 0, 1);
53 $pdf->Output();
54
55 ?>

```



11. referencias

- [Tutorial de Composer](#)
- [Web Scraping with PHP – How to Crawl Web Pages Using Open Source Tools](#)
- [PHP Monolog](#)
- [Unit Testing con PHPUnit — Parte 1.](#), de Emiliano Zublena.