

# Unidad 3- Trabajar con base de datos en PHP

---

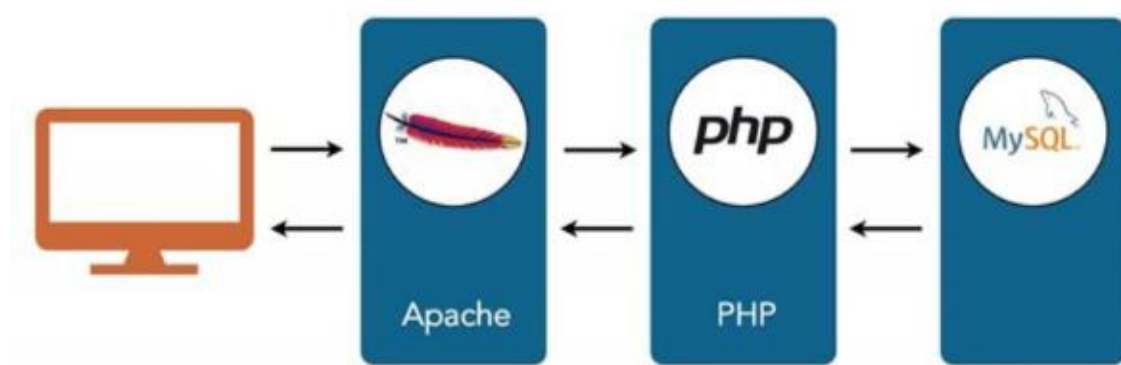
## ÍNDICE

<b>1. Acceso a datos .....</b>	<b>2</b>
1.1 Estructura de una base de datos.....	2
<b>2. SQL .....</b>	<b>3</b>
<b>3. phpMyAdmin .....</b>	<b>4</b>
3.1 Creando una base de datos dentro de phpMyAdmin .....	4
3.2 Opciones de phpMyAdmin.....	7
3.3 Creando una base de datos con comandos .....	7
<b>4 Extensión MySQLi.....</b>	<b>9</b>
4.1 Establecimiento de conexiones.....	9
4.2 Ejecución de consultas .....	10
4.3 Transacciones.....	11
4.4 Obtención y utilización de conjuntos de resultados .....	12
4.5 Consultas preparadas.....	13
<b>5. PHP Data Objects (PDO).....</b>	<b>16</b>
5.1 Establecimiento de conexiones.....	16
5.2 Ejecución de consultas .....	18
5.3 Obtención y utilización de conjuntos de resultados .....	19
5.4 Consultas preparadas.....	20
<b>6. Errores y manejo de excepciones.....</b>	<b>22</b>
6.1 Excepciones .....	23

## 1. Acceso a datos

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.

Lo haremos a través de las distintas capas o niveles: Apache, PHP y MySQL



### 1.1 Estructura de una base de datos

La base de datos tendrá la siguiente estructura:

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

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

## 2. SQL

El lenguaje de consulta estructurada (Structured Query Language) es el que vamos a utilizar para realizar consultas a nuestras bases de datos para mostrar el contenido en las distintas interfaces web que creemos a lo largo de la unidad.

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

```
SELECT * FROM país
```

Las sentencias SQL 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.

```

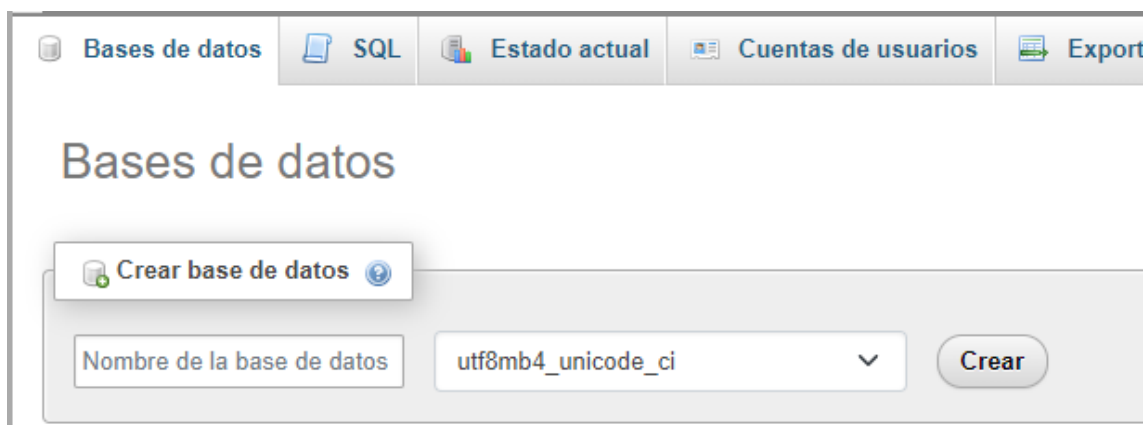
1  <?php
2      // Listado de clientes, ordenados por DNI de manera ASCendente
3      $clientesOrdenadosPorDNI = "SELECT * FROM `pasajero` ORDER BY `dni`" ASC;
4  ?>
```

### 3. phpMyAdmin

Es un software que funciona bajo Apache y PHP, es una interfaz web para gestionar las bases de datos que tengamos disponibles en nuestro servidor local.

#### 3.1 Creando una base de datos dentro de phpMyAdmin

- Abrimos XAMMP
- Localhost/phpmyadmin



Bases de datos

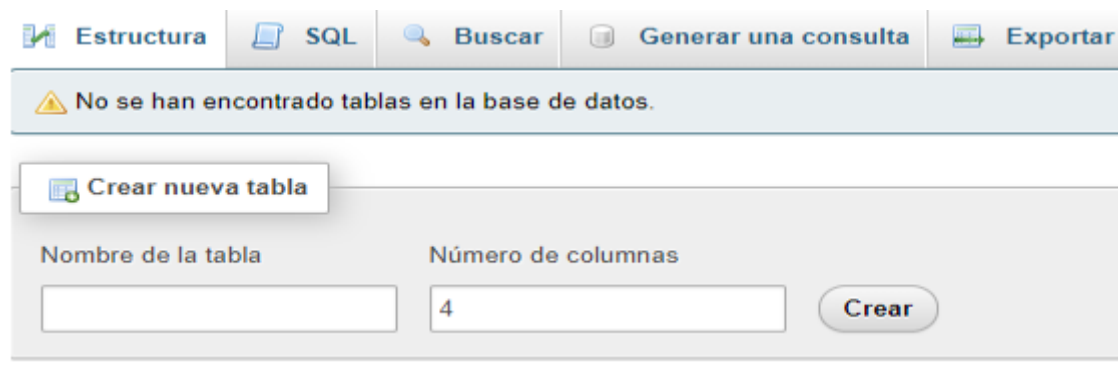
Crear base de datos

Nombre de la base de datos

utf8mb4\_unicode\_ci

Crear

- En la ventana de creación se pone un nombre a nuestra BBDD. En mi caso se llama **ejemplo**
- Estableceremos el cotejamiento **utf8mb4\_unicode\_ci** para que nuestra BBDD soporte todo tipo de caracteres.
- Le damos al botón Crear y ya se podrá empezar a escribir las distintas tablas que vayamos a introducir en ella



Estructura

SQL

Buscar

Generar una consulta

Exportar

No se han encontrado tablas en la base de datos.

Crear nueva tabla

Nombre de la tabla

Número de columnas

4

Crear

## Creemos una tabla

Vamos a crear una pequeña tabla donde aparezca el id y el nombre de usuario.

Entonces, en nombre le ponemos, por ejemplo: Person.

Y como queremos 2 datos, en número de campos ponemos 2.

Crear nueva tabla

Nombre de la tabla

Person

Número de columnas

2

Crear

Nombre	Tipo	Longitud/Valores	Predeterminado	Cotejamiento
<div>id</div> <div>Seleccionar desde las columnas centrales</div>	INT	100	Ninguno	
<div>Nombre</div> <div>Seleccionar desde las columnas centrales</div>	VARCHAR	100	Ninguno	
Nulo	Índice	A_I	Comentarios	Vir
<div></div>	<div><input type="checkbox"/></div>	<div>PRIMARY</div> <div>PRIMARY</div>	<div><input checked="" type="checkbox"/></div>	<div></div>
<div></div>	<div><input type="checkbox"/></div>	<div>---</div>	<div><input type="checkbox"/></div>	<div></div>

A\_I es autoincrementable.

Una vez creada la tabla, vamos a **meter datos en ella.**

Para ello, nos vamos a insertar y rellenamos.



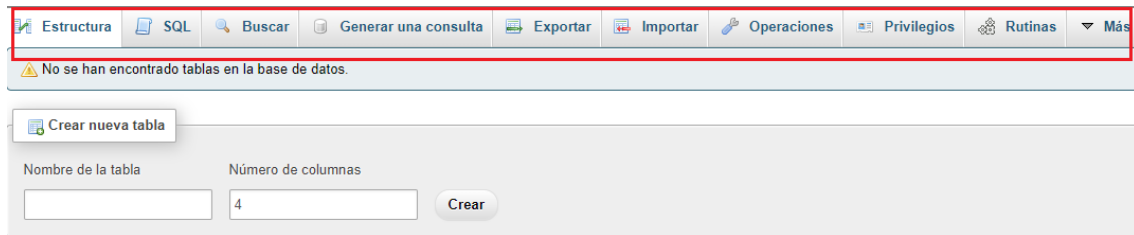
The screenshot shows a database management interface with a top navigation bar containing 'Examinar', 'Estructura', 'SQL', 'Buscar', and 'Insertar'. The 'Insertar' tab is active. Below the navigation bar, a green status bar indicates 'Mostrando filas 0 - 3 (total de 4, La consulta tardó 0,0004 segundos.)'. The SQL editor shows the query 'SELECT \* FROM `person`'. Below the editor, there are links for 'Perfilando', 'Editar en línea', 'Editar', 'Explicar SQL', 'Crear código PHP', and 'A'. A control bar shows 'Mostrar todo', 'Número de filas: 25', and 'Filtrar filas: Buscar en'. An 'Opciones extra' button is also present. The main area displays a table with 4 rows and 2 columns: 'Id' and 'Nombre'. Each row has a checkbox, an 'Editar' button, a 'Copiar' button, and a 'Borrar' button.

	Id	Nombre
<input type="checkbox"/>	1	Miguel
<input type="checkbox"/>	2	Antonio
<input type="checkbox"/>	3	Lucía
<input type="checkbox"/>	4	Juana

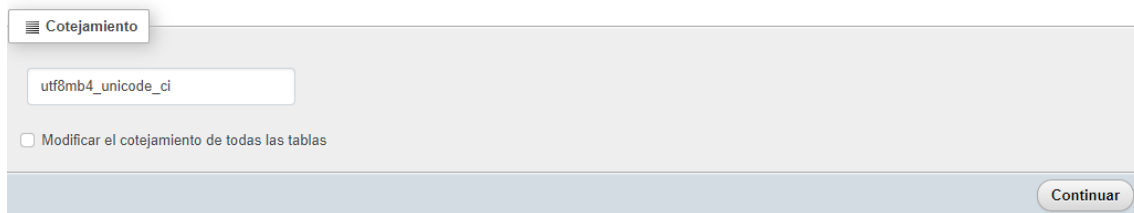
Dejamos esta pequeña base de datos reservada para futuros ejemplos.

### 3.2 Opciones de phpMyAdmin

Cuando seleccionamos una base de datos de la lista (la lista aparece en la izquierda), el sistema nos muestra varias pestañas:



- **Estructura:** podemos ver las distintas tablas que tenemos en nuestra base de datos
- **SQL:** por si queremos inyectar código SQL para que el sistema lo interprete
- **Buscar:** sirve para buscar los términos, en nuestra base de datos, aplicando distintos filtros de búsqueda
- **Generar una consulta:** parecido a SQL, pero de una manera más gráfica, sin tener que saber nada del lenguaje
- **Exportar e importar:** cualquiera de las 2 operaciones sobre la base de datos.
- **Operaciones:** distintas operaciones avanzadas para realizar en nuestra base de datos. Destacaremos la opción de Cotejamiento



Se puede cambiar el cotejamiento de nuestra tabla. Pero **CUIDADO** porque se pueden eliminar datos sin querer, ya que, al cambiar el cotejamiento podemos suprimir caracteres no soportados por el nuevo cotejamiento.

### 3.3 Creando una base de datos con comandos

- **Crear base de datos:**

```
CREATE DATABASE Prueba;
```

- **Crear tablas en nuestra base de datos:**

```
CREATE TABLE nombretabla  
(  
    nombrecolumna1 tipodato1,  
    nombrecolumna2 tipodato2,  
    nombrecolumna3 tipodato3,  
    ..  
);
```

## PROPIEDADES DE LAS COLUMNAS DE LAS TABLAS

**Permitir valores NULL:** Por defecto si no se especifica NOT NULL, la columna podrá soportar nulos.

**Primary Key:** Indica si la columna será la llave primaria de la tabla.

**Unique:** Indica que la columna permitirá valores únicos, no se puede repetir un mismo valor en varias filas (row).

**Default:** Se indica el valor default que tendrá la columna cuando en la inserción no este contemplado este campo.

**Identity:** La columna será auto incrementable por defecto inicia en 1.

**Description:** Una breve descripción a la columna.

**Ejemplo:**

```
1 CREATE TABLE Alumnos (  
2 Id INT PRIMARY KEY NOT AUTO_INCREMENT NOT NULL,  
3 Nombre VARCHAR(40) NOT NULL,  
4 Apellido VARCHAR(40) NOT NULL,  
5 Domicilio TEXT,  
6 Fecha_Nacimiento DATETIME  
7 ); |
```

- **Insertar valores**

```
INSERT INTO nombretabla (nombrecolumna1, nombrecolumna2,  
etc)
```

```
VALUES ('Dato1', 'Dato2', etc);
```

**Ejemplo:**

```
1 INSERT INTO alumnos (Nombre, Apellido, Domicilio, Fecha_Nacimiento)  
2 VALUES ('Nombre1', 'Apellido 1', 'Domicilio 1', '01/12/1999'),  
3 ('Nombre2', 'Apellido 2', 'Domicilio 2', '02/12/1999'); |  
4
```



## 4 Extensión MySQLi

PHP hace uso de esta extensión mejorada para poder comunicarse con las bases de datos.

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:

### 4.1 Establecimiento de conexiones

Para poder comunicarte desde un programa PHP con un servidor MySQL, el primer paso es establecer una conexión.

**Establecer una conexión** con el servidor en MySQLi significa crear una instancia de la clase `mysqli`. El constructor de la clase puede recibir 6 parámetros, todos opcionales. Lo más habitual es utilizar los cuatro primeros:

1. El nombre o dirección IP del servidor MySQL al que te quieres conectar.
2. Un nombre de usuario con permisos para establecer la conexión.
3. La contraseña del usuario.
4. El nombre de la base de datos a la que conectarse.
5. El número del puerto en que se ejecuta el servidor MySQL.
6. El socket o la tubería con nombre (named pipe) a usar.

Además, es importante verificar que la conexión se ha establecido correctamente. 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

Si utilizar el constructor de la clase, para conectarte a la base de datos **ejemplo** sería:

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

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.

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 se utiliza.

## 4.2 Ejecución de consultas

Sabemos cómo conectarnos a una base de datos alojada en nuestro servidor, ahora vamos a ver cómo **recuperar datos almacenados** en la base de datos.

La forma más inmediata de ejecutar una consulta, si utilizas esta extensión, es el método `query` o 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.

En el caso de ejecutar una sentencia SQL que sí devuelva datos (como un SELECT), éstos se devuelven en forma de un objeto.

Veamos cómo hacerlo: En el punto 3.1 del tema, hemos creado una tabla llamada Person. Para acceder a ella:

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

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

```
1 Miguel
2 Antonio
3 Lucía
4 Juana
```

### 4.3 Transacciones

Si necesitas utilizar transacciones deberás asegurarte de que estén soportadas por el motor de almacenamiento que gestiona tus tablas en MySQL. Si utilizas InnoDB, por defecto cada consulta individual se incluye dentro de su propia transacción. Puedes gestionar este comportamiento con el método `autocommit` (función `mysqli_autocommit`).

```
1 <?php
2 $dsn = 'mysql:localhost;dbname=ejemplo';
3 $usuario = 'root';
4 $contraseña = '';
5 $conexion = mysqli_connect($dsn, $usuario, $contraseña );
6 |
7
8 $conexion->autocommit(false); // deshabilitamos el modo transaccional automático
```

Al deshabilitar las transacciones automáticas, 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.

```
1 <?php
2 $dsn = 'mysql:localhost;dbname=ejemplo';
3 $usuario = 'root';
4 $contraseña = '';
5 $conexion = mysqli_connect($dsn, $usuario, $contraseña );
6
7 $conexion->query('DELETE FROM stock WHERE unidades=0'); // Inicia una transacción
8 $conexion->query('UPDATE stock SET unidades=3 WHERE producto="STYLUSSX515W"');
9 $conexion->commit(); // Confirma los cambios
```

#### 4.4 Obtención y utilización de conjuntos de resultados

Ya hemos visto que para ejecutar una consulta que devuelve datos obtienes un conjunto de la clase `mysqli_result`.

Para trabajar con los datos obtenidos en el servidor, hay varias posibilidades:

1. `fetch_array` (función `mysqli_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.

```
1 <?php
2 $dsn = 'mysql:localhost;dbname=ejemplo';
3 $usuario = 'root';
4 $contraseña = '';
5 $conexion = mysqli_connect($dsn, $usuario, $contraseña );
6
7 $resultado = $conexion->query('SELECT producto, unidades FROM stock WHERE unidades<2');
8 $stock = $resultado->fetch_array(); // Obtenemos el primer registro
9 $producto = $stock['producto']; // O también $stock[0];
10 $unidades = $stock['unidades']; // O también $stock[1];
11 print "<p>Producto $producto: $unidades unidades.</p>";
```

Este comportamiento por defecto se puede modificar utilizando un parámetro opcional, que puede tomar 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.
2. `fetch_assoc` (función `mysqli_fetch_assoc`). Idéntico a `fetch_array` pasando como parámetro **MYSQLI\_ASSOC**.
  3. `fetch_row` (función `mysqli_fetch_row`). Idéntico a `fetch_array` pasando como parámetro **MYSQLI\_NUM**.
  4. `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.

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

```
1 <?php
2 $dsn = 'mysql:localhost;dbname=ejemplo';
3 $usuario = 'root';
4 $contraseña = '';
5 $conexion = mysqli_connect($dsn, $usuario, $contraseña );
6
7 $resultado = $conexion->query('SELECT producto, unidades FROM stock WHERE unidades<2');
8 $stock = $resultado->fetch_object();
9 while ($stock != null) {
10     print "<p>Producto $stock->producto: $stock->unidades unidades.</p>";
11     $stock = $resultado->fetch_object();
12 }
```

## 4.5 Consultas preparadas

Cada vez que se envía una consulta al servidor, este debe analizarla antes de ejecutarla.

Algunas sentencias SQL, como las que insertan valores en una tabla, deben repetirse de forma habitual en un programa.

Para acelerar este proceso, MySQL admite consultas preparadas. Estas consultas se almacenan en el servidor, listas para ser ejecutadas cuando sea necesario.

Para trabajar con consultas preparadas con la extensión MySQLi de PHP, debes utilizar la clase `mysqli_stmt`. Utilizando el método `stmt_init` de la clase `mysqli` (o la función `mysqli_stmt_init`) obtienes un objeto de dicha clase.

```
1  <?php
2  $dsn = 'mysql:localhost;dbname=ejemplo';
3  $usuario = 'root';
4  $contraseña = '';
5  $conexion = mysqli_connect($dsn, $usuario, $contraseña );
6
7  $consulta = $conexion->stmt_init();
```

Pasos a seguir para ejecutar una consulta preparada son:

1. Preparar la consulta en el servidor MySQL utilizando el método `prepare` (función `mysqli_stmt_prepare`).
2. Ejecutar la consulta, tantas veces como sea necesario, con el método `execute` (función `mysqli_stmt_execute`).
3. Una vez que ya no se necesita más, se debe ejecutar el método `close` (función `mysqli_stmt_close`).

Por ejemplo, para preparar y ejecutar una consulta que inserta un nuevo registro en la familia:

```
1  <?php
2  $dsn = 'mysql:localhost;dbname=ejemplo';
3  $usuario = 'root';
4  $contraseña = '';
5  $conexion = mysqli_connect($dsn, $usuario, $contraseña );
6
7  $consulta = $conexion->stmt_init();
8  $consulta->prepare('INSERT INTO familia (cod, nombre) VALUES ("TABLET", "Tablet PC")');
9  $consulta->execute();
10 $consulta->close();
11 $conexion->close();
```

**El problema** es que de poco sirve preparar una consulta de inserción de datos como la anterior, si los valores que inserta son siempre los mismos. Por este motivo, las consultas preparadas admiten parámetros.

Para preparar una consulta con parámetros, en lugar de poner los valores, debes indicar con un signo de interrogación su posición dentro de la sentencia SQL.

```
9 $consulta->prepare('INSERT INTO familia (cod, nombre) VALUES (?, ?)');
```

Antes de ejecutar la consulta tienes que utilizar el método `bind_param` (o la función `mysqli_stmt_bind_param`) para sustituir cada parámetro por su valor. El primer parámetro del método `bind_param` es una cadena de texto en la que cada carácter indica el tipo de un parámetro, según la siguiente tabla.

Carácter	Tipo del parámetro
I.	Número entero
D.	Número real
S.	Cadena de texto
B.	Contenido en formato binario

En el caso anterior, si almacenas los valores a insertar en sendas variables, puedes hacer:

```
1 <?php
2 $dsn = 'mysql:localhost;dbname=ejemplo';
3 $usuario = 'root';
4 $contraseña = '';
5 $conexion = mysqli_connect($dsn, $usuario, $contraseña );
6
7 $consulta = $conexion->stmt_init();
8 $consulta->prepare('INSERT INTO familia (cod, nombre) VALUES (?, ?)');
9 $cod_producto = "TABLET";
10 $nombre_producto = "Tablet PC";
11 $consulta->bind_param('ss', $cod_producto, $nombre_producto);
12 $consulta->execute();
13 $consulta->close();
14 $conexion->close();
```

**Nota:** cuando uses `bind_param` para enlazar los parámetros de una consulta preparada con sus respectivos valores, deberás usar siempre variables como en el ejemplo anterior. Si intentas utilizar literales, por ejemplo:

```
7
8 $consulta->bind_param('ss', 'TABLET', 'Tablet PC'); // Genera un error
```

Obtendrás un error. El motivo es que los parámetros del método `bind_param` se pasan por referencia.

El método `bind_param` permite tener una consulta preparada en el servidor MySQL y ejecutarla tantas veces como quieras cambiando ciertos valores cada vez. Además, en el caso de las consultas que devuelven valores, se puede utilizar el método `bind_result` (función `mysqli_stmt_bind_result`) para asignar a variables los campos que se obtienen tras la ejecución. Utilizando el método `fetch` (`mysqli_stmt_fetch`) se recorren los registros devueltos. Observa el siguiente código:

```
1  <?php
2  $dsn = 'mysql:localhost;dbname=ejemplo';
3  $usuario = 'root';
4  $contraseña = '';
5  $conexion = mysqli_connect($dsn, $usuario, $contraseña );
6
7  $consulta = $conexion->stmt_init();
8  $consulta->prepare('SELECT producto, unidades FROM stock WHERE unidades<2');
9  $consulta->execute();
10 $consulta->bind_result($producto, $unidades);
11 while($consulta->fetch()) {
12     print "<p>Producto $producto: $unidades unidades.</p>";
13 }
14 $consulta->close();
15 $conexion->close();
```

#### ACTIVIDADES PROPUESTAS – Classroom: Unidad 3 - Tarea 1

*Importante: Antes de hacer las actividades, en la carpeta DWEEES de htdocs, crea una carpeta que se llame U2Tarea1*

## 5. PHP Data Objects (PDO)

De la misma manera que hemos visto con mysqli, PHP Data Objects (PDO) es un driver de PHP que se utiliza para trabajar bajo un 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.

PDO se basa en las características de orientación a objetos de PHP (que desarrollaremos en temas posteriores) 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 ofrece funciones alternativas.

### 5.1 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):

1. 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.
2. Nombre de usuario con permisos para establecer la conexión.
3. Contraseña del usuario.
4. Opciones de conexión, almacenadas en forma de array.

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

```
<?php
$conexion = new PDO('mysql:host=localhost; dbname=ejemplo', 'root', '');
// (DNS, Nombre base datos, Usuario, Contraseña)
```

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 ; ) a continuación del prefijo mysql: son las 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.

Si quisieras indicar al servidor MySQL utilice codificación UTF-8 para los datos que se transmitan, puedes usar una opción específica de la conexión:



```

1  <?php
2  $dsn = 'mysql:localhost;dbname=ejemplo';
3  $usuario = 'root';
4  $contraseña = '';
5  $opciones = array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8");
6  $conexion = new PDO($dsn, $usuario, $contraseña, $opciones);
7  ?>

```

Una vez establecida la conexión, puedes utilizar los siguientes métodos:

- `getAttribute` para obtener la información del estado de la conexión
- `setAttribute` para modificar algunos parámetros que afectan a la misma

En ejemplo, se usa `getAttribute` para obtener la versión del servidor y `setAttribute` para que te devuelva todos los nombres de columnas en mayúscula:

```

1  <?php
2  $dsn = 'mysql:localhost;dbname=ejemplo';
3  $usuario = 'root';
4  $contraseña = '';
5  $opciones = array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8");
6  $conexion = new PDO($dsn, $usuario, $contraseña, $opciones);
7
8  |
9  // Obtener la versión del servidor
10 $version = $conexion->getAttribute(PDO::ATTR_SERVER_VERSION);
11 print "Version: $version"
12
13 // Hacer que te devuelva todos los nombres de columnas en mayúscula
14
15 $version = $conexion->setAttribute(PDO::ATTR_CASE, PDO::CASE_UPPER);
16
17
18 ?>

```

En el manual de PHP, las páginas de las funciones `getAttribute` y `setAttribute` te permiten consultar los posibles parámetros que se aplican a cada una:

<https://www.php.net/manual/es/pdo.getattribute.php>

<https://www.php.net/manual/es/pdo.setattribute.php>

## 5.2 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 consultas de acción como INSERT, DELETE o UPDATE, el método `exec` devuelve el número de registros afectados.

```
<?php
$dsn = 'mysql:localhost;dbname=ejemplo';
$usuario = 'root';
$contraseña = '';
$opciones = array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8");
$conexion = new PDO($dsn, $usuario, $contraseña, $opciones);

$registros = $conexion->exec('DELETE FROM stock WHERE unidades=0');
print "<p>Se han borrado $registros registros.</p>";

?>
```

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`

```
6  $conexion = new PDO($dsn, $usuario, $contraseña, $opciones);
7
8
9  $resultado = $conexion->query("SELECT producto, unidades FROM stock");
10
11 ?>
```

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 3 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.

```

1  <?php
2  $dsn = 'mysql:localhost;dbname=ejemplo';
3  $usuario = 'root';
4  $contraseña = '';
5  $opciones = array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8");
6  $conexion = new PDO($dsn, $usuario, $contraseña, $opciones);
7
8
9  $ok = true;
10 $conexion->beginTransaction();
11 if($conexion->exec('DELETE ...') == 0) $ok = false;
12 if($conexion->exec('UPDATE ...') == 0) $ok = false;
13 ...
14 if ($ok) $conexion->commit(); // Si todo fue bien confirma los cambios
15 else $conexion->rollback(); // y si no, los revierte
16
17 ?>

```

### 5.3 Obtención y utilización de conjuntos de resultados

Al igual que con la extensión MySQLi, en PDO hay varias posibilidades para tratar el conjunto de resultados devuelto por el método query visto anteriormente.

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.

```

1  <?php
2  $dsn = 'mysql:localhost;dbname=ejemplo';
3  $usuario = 'root';
4  $contraseña = '';
5  $conexion = new PDO($dsn, $usuario, $contraseña);
6
7  |
8  $resultado = $conexion->query("SELECT producto, unidades FROM stock");
9  while ($registro = $resultado->fetch()) {
10     echo "Producto ".$registro['producto'].": ".$registro['unidades']."<br />";
11 }

```

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.

```

1  <?php
2  $dsn = 'mysql:localhost;dbname=ejemplo';
3  $usuario = 'root';
4  $contraseña = '';
5  $conexion = new PDO($dsn, $usuario, $contraseña);
6
7
8  $resultado = $conexion->query("SELECT producto, unidades FROM stock");
9  while ($registro = $resultado->fetch(PDO::FETCH_OBJ)) {
10     echo "Producto ".$registro->producto." : ".$registro->unidades."<br />";
11 }
12

```

- **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.

```

1  <?php
2  $dsn = 'mysql:localhost;dbname=ejemplo';
3  $usuario = 'root';
4  $contraseña = '';
5  $conexion = new PDO($dsn, $usuario, $contraseña);
6
7
8  $resultado = $conexion->query("SELECT producto, unidades FROM stock");
9  $resultado->bindColumn(1, $producto);
10 $resultado->bindColumn(2, $unidades);
11 while ($registro = $resultado->fetch(PDO::FETCH_OBJ)) {
12     echo "Producto ".$producto." : ".$unidades."<br />";
13 }

```

#### 5.4 Consultas preparadas

Al igual que con MySQLi, utilizando PDO también 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.

Se utiliza 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 como en el caso anterior

```

1  <?php
2  $dsn = 'mysql:localhost;dbname=ejemplo';
3  $usuario = 'root';
4  $contraseña = '';
5  $conexion = new PDO($dsn, $usuario, $contraseña);
6
7  $consulta = $conexion->prepare('INSERT INTO familia (cod, nombre) VALUES (?, ?)');

```

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

```

1  <?php
2  $dsn = 'mysql:localhost;dbname=ejemplo';
3  $usuario = 'root';
4  $contraseña = '';
5  $conexion = new PDO($dsn, $usuario, $contraseña);
6
7  $consulta = $conexion->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 `bindParam` que acabamos de ver.

```

7  $cod_producto = "TABLET";
8  $nombre_producto = "Tablet PC";
9  $consulta->bindParam(1, $cod_producto);
10 $consulta->bindParam(2, $nombre_producto);

```

Si utilizas parámetros con nombre, debes indicar ese nombre en la llamada a `bindParam`.

```

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

```

**Nota:** Tal y como sucedía con la extensión MySQLi, cuando uses `bindParam` para asignar los parámetros de una consulta preparada, deberás usar siempre variables como en ejemplo anterior.

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

```

6
7  $consulta->execute();

```

Alternativamente, es posible asignar los valores de los parámetros en el momento de ejecutar la consulta, utilizando un array (asociativo o con claves numéricas dependiendo de la forma en que hayas indicado los parámetros) en la llamada a `execute`.

```

7  $parametros = array(":cod" => "TABLET", ":nombre" => "Tablet PC");
8  $consulta->execute($parametros);

```

## ACTIVIDADES PROPUESTAS – Classroom: Unidad 3 - Tarea 2

*Importante: Antes de hacer las actividades, en la carpeta DWEEES de htdocs, crea una carpeta que se llame U3Tarea2*

## 6. 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 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 el guion, y la constante **E\_ERROR** engloba errores fatales que provocan que se interrumpa forzosamente la ejecución.

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.

**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:

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

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 ejecuta PHP 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).

Por ejemplo:

```
 9  set_error_handler("miGestorDeErrores");
10  $resultado = $dividendo / $divisor;
11  restore_error_handler();
12
13  function miGestorDeErrores($nivel, $mensaje)
14  {
15      switch($nivel) {
16          case E_WARNING:
17              echo "Error de tipo WARNING: $mensaje.<br />";
18              break;
19          default:
20              echo "Error de tipo no especificado: $mensaje.<br />";
21          }
22  }
```

## 6.1 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.

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

```
 8  try {
 9      if ($divisor == 0)
10          throw new Exception("División por cero.");
11      $resultado = $dividendo / $divisor;
12  }
13  catch (Exception $e) {
14      echo "Se ha producido el siguiente error: ".$e->getMessage();
15  }
```

PHP ofrece una clase base `Exception` para utilizar como manejador 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.

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`.

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`**. No se hace nada cuando ocurre un error. Es el comportamiento por defecto.
- **`PDO::ERRMODE_WARNING`**. Genera un error de tipo `E_WARNING` cuando se produce un error.
- **`PDO::ERRMODE_EXCEPTION`**. Cuando se produce un error lanza una excepción utilizando el manejador propio `PDOException`.

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

```
1 <?php
2 $dsn = 'mysql:localhost;dbname=ejemplo';
3 $usuario = 'root';
4 $contraseña = '';
5 $conexion = new PDO($dsn, $usuario, $contraseña);
6
7 $conexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

Por ejemplo, el siguiente código:



```

1  <?php
2  $dsn = 'mysql:localhost;dbname=ejemplo';
3  $usuario = 'root';
4  $contraseña = '';
5  $conexion = new PDO($dsn, $usuario, $contraseña);
6
7  $conexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
8  try {
9      $sql = "SELECT * FROM stox";
10     $result = $conexion->query($sql);
11
12 }
13 catch (PDOException $p) {
14     echo "Error ".$p->getMessage()."<br />";
15 }

```

Captura la excepción que lanza PDO debido a que la tabla no existe. El bloque catch muestra el siguiente mensaje:

```

Error SQLSTATE[42S02]: Base table or view not found: 1146 Table 'conexion.stox' doesn't exist

```