UNIDAD 3

# Trabajar con bases de datos en PHP.

En esta unidad utilizarás esos conocimientos para realizar programas sencillos que utilicen información almacenada en bases de datos. El principal sistema gestor de bases de datos que se utiliza junto al lenguaje PHP es MySQL. A continuación, aprenderás a acceder desde PHP a bases de datos MySQL utilizando las extensiones MySQLi y PDO.

# 1.- Acceso a bases de datos desde PHP.

Una de las aplicaciones más frecuentes de PHP es generar una interfaz web para acceder y gestionar la información almacenada en una base de datos. Usando PHP podemos mostrar en una página web información extraída de la base de datos, o enviar sentencias al gestor de la base de datos para que elimine o actualice algunos registros.

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

A partir de la versión 5 de PHP se introdujo en el lenguaje una extensión para acceder de una forma común a distintos sistemas gestores: **PDO**. La gran ventaja de **PDO está clara**: podemos seguir utilizando **una misma sintaxis**, aunque cambiemos el motor de nuestra base de datos. Por el contrario, en algunas ocasiones preferiremos seguir usando extensiones nativas en nuestros programas. Mientras PDO ofrece un conjunto común de funciones, **las extensiones nativas** normalmente **ofrecen** más **potencia** (acceso a funciones específicas de cada gestor de base de datos) y en algunos casos también **mayor velocidad.**

De los distintos **SGBD** existentes, vas a aprender a utilizar **MySQL**. MySQL es un gestor de bases de datos **relacionales** de **código abierto bajo licencia GNU GPL**. Es el gestor de bases de datos más empleado con el lenguaje PHP. Como ya vimos, es la letra "M" que figura en los acrónimos AMP y XAMPP.

Veremos cómo acceder desde PHP a bases de datos MySQL utilizando tanto **PDO** como la extensión nativa **MySQLi**. Previamente verás una pequeña introducción al manejo de MySQL, aunque para el seguimiento de esta unidad se supone que conoces el lenguaje SQL utilizado en la gestión de bases de datos relacionales.

Además, para el acceso a las funcionalidades de ambas extensiones deberás utilizar objetos. Aunque más adelante verás todas las características que nos ofrece PHP para crear programas orientados a objetos, debemos suponer también en este punto un cierto conocimiento de programación orientada a objetos. Básicamente, debes saber cómo crear y utilizar objetos.

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

$a = new A();

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

$a->fecha();

# 2.- MySQL.

MySQL es un sistema gestor de bases de datos (**SGBD**) **relacionales**. Es un programa de código abierto que se ofrece bajo licencia GNU GPL, aunque también ofrece una licencia comercial en caso de que quieras utilizarlo para desarrollar aplicaciones de código propietario. En las últimas versiones (a partir de la 5.1), se ofrecen, de hecho, varios productos distintos: uno de código libre **(Community Edition),** y otro u otros comerciales **(Standard Edition, Enterprise Edition).**

Incorpora múltiples motores de almacenamiento, cada uno con características propias: unos son más veloces, otros, aportan mayor seguridad o mejores capacidades de búsqueda. Cuando crees una base de datos, puedes elegir el motor en función de las características propias de la aplicación. Si no lo cambias, el motor que se utiliza por defecto se llama **MyISAM**, que es **muy rápido,** pero a cambio **no contempla integridad referencial ni tablas transaccionales**. El motor **InnoDB** es un poco **más lento,** pero sí soporta tanto **integridad referencial como tablas transaccionales**.

**MySQL** se emplea en múltiples aplicaciones web, ligado en la mayor parte de los casos al lenguaje PHP y al servidor web Apache. Utiliza SQL para la gestión, consulta y modificación de la información almacenada. Soporta la mayor parte de las características de **ANSI SQL 99,** y añade además algunas extensiones propias.

En el tema anterior instalamos **Xampp** en **Windows**, el SGBD que viene con **Xampp** es **MariaDB**que es un sistema de gestión de bases de datos derivado de**MySQL**con licencia GPL. A todos los efectos, a lo largo de este tema y del módulo, dará igual utilizar uno u otro, de hecho, **MariaDB** está desarrollado por parte de la comunidad de **MySQL**, antes de que el proyecto fuese adquirido por Oracle y, salvo ligeros cambios, ambos son altamente compatibles.

Guía de MySQL: <https://downloads.mysql.com/docs/refman-5.0-es.pdf>

## 2.1.- Base de datos proyecto

La base de datos **proyecto**. En ella meteremos las tablas del siguiente esquema, fíjate en los campos y las relaciones entre ellas. De igual manera en dicho archivo crearemos el usuario "**gestor**" con contraseña "**secreto**" y le daremos todos los permisos en la base de datos "**proyecto**".

A continuación, se muestra el código SQL para la creación de la base de datos proyecto, junto con sus tablas y los datos. Copia el código a un fichero llamado proyecto.sql para llevar a cabo la creación de la base de datos siguiendo las indicaciones de la sección siguiente.

## 2.2.- Herramientas de administración.

Existen muchas herramientas que permiten establecer una conexión con un servidor MySQL para realizar tareas de administración. Algunas herramientas se ejecutan en la línea de comandos, otras presentan una interfaz gráfica basado en web o propio del sistema operativo en que se ejecuten. Unas se incluyen con el propio servidor, y otras es necesario obtenerlas e instalarlas de forma independiente. Las hay que están orientadas a algún propósito concreto y también que permiten realizar varias funciones de administración.

Con el servidor MySQL se incluyen algunas **herramientas de administración** en línea de comandos, entre las que debes conocer:

* **mysql**. Permite conectarse a un servidor MySQL para ejecutar sentencias.
* **mysqladmin**. Es un cliente específico para tareas de administración.
* **mysqlcheck**. Permite correr acciones de mantenimiento de las tablas de la base de datos. Comprueba, repara, optimiza y analiza tablas.
* **mysqldump**. Es una utilidad de cliente para realizar copias de seguridad de las bases de datos.
* **mysqlimport**. Proporciona una línea de comandos para importar los contenidos de un fichero.
* **mysqlshow**. Muestra información sobre bases de datos y tablas.

Estas herramientas comparten unas cuantas opciones relativas al establecimiento de la conexión con el servidor. Muchas de estas opciones tienen también una forma abreviada:

**--user=nombre\_usuario (-u nombre\_usuario).** Indica un nombre de usuario con permisos para establecer la conexión. Si no se especifica se usará el nombre de usuario actual del sistema operativo.

**--password=contraseña (-pcontraseña).** Contraseña asociada al nombre de usuario anterior. Si se utiliza la opción abreviada, debe figurar justo a continuación de la letra p, sin espacios intermedios. Si es necesario introducir una contraseña y no se indica ninguna, se pedirá para establecer la conexión.

**--host=equipo\_servidor (-h equipo\_servidor).** Nombre del equipo con el que se establecerá la conexión. Si no se indica nada, se usará "localhost".

Para establecer una conexión al servidor local con la herramienta mysql, podemos hacer:

El ejecutable "mysql" está en "c:\xampp\mysql\bin\" con una consola cmd o powershell abierta en esta ubicación (si no la hemos metido en la ruta (path) de los ejecutable del sistema) basta con **teclear "mysql -u root"** .

Con cualquier otro usuario: **mysql -u usuario -p baseDeDatos**. Por ejemplo: "mysql -u juan -p proyecto".

**mysql**

La forma más habitual de utilizar la herramienta **mysql** es en modo interactivo. Una vez te conectas al servidor **MySQL**, te presenta una línea de órdenes. En esa línea de órdenes puedes introducir sentencias SQL, que se ejecutarán sobre la base de datos seleccionada, y algunos comandos especiales. Las sentencias SQL deben terminar en el carácter ";". Entre los comandos especiales que puedes usar están:

* **connect**. Establece una conexión con un servidor **MySQL**.
* **use**. Permite seleccionar una base de datos.
* **exit** o **quit**. Termina la sesión interactiva con **MySQL**.
* **help**. Muestra una pantalla de ayuda con la lista de comandos disponibles.

Por ejemplo, si cuando estás utilizando la herramienta quieres seleccionar la base de datos "dwes", debes hacer: **mysql>use dwes;**

Las sentencias que teclees a partir de ese instante se ejecutarán sobre la base de datos "dwes".

También puedes usar el comando **mysql** para que ejecute todas las sentencias de un archivo de procesamiento por lotes (normalmente con extensión .**sql**).

mysql -u root -p < proyecto.sql

**Debes conocer**

De entre el resto de las herramientas de administración independientes que podemos utilizar con MySQL, podemos destacar dos:

**MySQL** **Workbench** es una herramienta genérica con interfaz gráfico nativo que permite administrar tanto el servidor como las bases de datos que éste gestiona. Ha sido desarrollada por los creadores de MySQL y se ofrece en dos ediciones, una de ellas de código abierto bajo licencia GPL.

**phpMyAdmin** es una aplicación web muy popular para la administración de servidores MySQL. Presenta una interfaz web de administración programado en PHP bajo licencia GPL. Su objetivo principal es la administración de las bases de datos y la gestión de la información que maneja el servidor.

**Recomendación**

¿Podemos o no almacenar "emojis" en **MySQL**?, la respuesta es si, pero no utilizando el "**utf-8**" que normalmente se usaba.

En 2010 (Con su versión **5.5.3**) **MySQL** agrega una variante a **"utf-8"** llamada "**utf8mb4"**. Con este nuevo tipo de codificación, cada carácter puede ser representado hasta 4 bytes, lo que nos permitirá guardar "emojis" en nuestras tablas algo cada vez más necesario.

mysql>create database mi\_base\_de\_datos CHARACTER SET = utf8mb4 COLLATE = utf8mb4\_unicode\_ci;

## 2.2.2.- phpMyAdmin.

XAMPP incluye la herramienta **phpMyAdmin** con lo que no hace falta ninguna instalación adicional. Para ejecutarlo pichamos en el botón **Admin** asociado a MySQL en el panel de control de XAMPP.

Por otro lado, si el entorno se ha creado instalando servidor MySQL, debes instalarlo de forma individual ya que no viene incluido en la distribución del servidor.

En el caso de Ubuntu lo podemos hacer de dos formas:

* La más sencilla es usar el gestor de paquetes, por ejemplo, tecleando desde un terminal: **sudo apt install phpmyadmin**
* El proceso de instalación es sencillo. Simplemente te pregunta por el servidor web a utilizar (escoger apache2), y después debes dejar que configure una nueva base de datos propia en el servidor. Una vez instalada la aplicación, podrás acceder vía web con un navegador utilizando la URL "**http://localhost/phpmyadmin/**".
* La otra es descargarnos directamente el paquete de la web de **phpMyadmin** ([**https://www.phpmyadmin.net/**)](https://www.phpmyadmin.net/)) y descomprimirlo en "**/var/www/html**"  (el directorio raíz de Apache en Ubuntu). Lo más sencillo es que renombremos la carpeta con el nombre **phpmyadmin**.

La interfaz de la aplicación se compone de un panel de navegación a la izquierda, donde se muestran las bases de datos, y un panel principal con un menú en la parte superior y una serie de acciones e información en la parte central. Si seleccionas la base de datos "**proyecto**", la información en pantalla cambia.

Utilizando los menús de la parte superior, puedes:

* Ver y modificar la **estructura** de la base de datos.
* Ejecutar sentencias SQL.
* **Buscar** información en toda la base de datos o en parte de esta.
* **Generar una consulta** utilizando un asistente.
* **Exportar** e **importar** información, tanto de la estructura como de los datos.
* **Diseñar** las relaciones existentes entre las tablas.
* Otras **operaciones**, como hacer una copia de la base de datos.

Si seleccionas una tabla en lugar de la base de datos, podrás efectuar a ese nivel operaciones similares a las anteriores.

Para crear la base de datos "**proyecto**" y el usuario "gestor" selecciona la pestaña Importar y pincha en el botón Seleccionar archivo, selecciona el archivo donde reside el esquema y los datos de la base de datos y finalmente pincha el botón Continuar en la parte inferior de la página.

Web phpmyadmin: <https://docs.phpmyadmin.net/en/latest/>

## 2.2.1.- mysqladmin.

La forma más habitual de utilizar la herramienta **mysql** es en modo interactivo. Una vez te conectas al servidor **MySQL**, te presenta una línea de órdenes. En esa línea de órdenes puedes introducir sentencias SQL, que se ejecutarán sobre la base de datos seleccionada, y algunos comandos especiales. Las sentencias SQL deben terminar en el carácter ";". Entre los comandos especiales que puedes usar están:

* **connect**. Establece una conexión con un servidor **MySQL**.
* **use**. Permite seleccionar una base de datos.
* **exit** o **quit**. Termina la sesión interactiva con **MySQL**.
* **help**. Muestra una pantalla de ayuda con la lista de comandos disponibles.

Por ejemplo, si cuando estás utilizando la herramienta quieres seleccionar la base de datos "dwes", debes hacer: **mysql>use dwes;**

Las sentencias que teclees a partir de ese instante se ejecutarán sobre la base de datos "dwes".

**mysqladmin** es una herramienta no interactiva orientada a tareas de administración del propio servidor. Las tareas concretas de administración a llevar a cabo se indican mediante parámetros en la línea de comandos. Entre las tareas que puedes llevar a cabo con esta utilidad se encuentran:

* Crear y eliminar bases de datos.
* Mostrar la configuración y el estado del servidor.
* Cambiar contraseñas.
* Detener un servidor.

Por ejemplo, si quieres mostrar información sobre el estado actual del servidor local, puedes utilizar el comando **"status"** y para ver la versión instalada **"version"**, puedes ver los comandos disponibles usando **"mysqladmin --help"**:

mysqladmin --help

sudo mysqladmin status

sudo mysqladmin version

Fíjate que los comandos anteriores no necesitan especificar usuario porque son solamente de información. Si quisiésemos, por ejemplo, crear una base de datos, borrarla, reiniciar el servidor... con **mysqladmin,** deberíamos hacerlo con **sudo** y especificando el usuario **root**. Por ejemplo, para crear una base de datos de nombre **miBase**:

sudo mysqladmin -u root create miBase

# 3.- Utilización de bases de datos MySQL en PHP.

Existen dos formas de comunicarse con una base de datos desde PHP: utilizar una extensión nativa programada para un **SGBD** concreto, o utilizar una extensión que soporte varios tipos de bases de datos. Tradicionalmente las conexiones se establecían utilizando la extensión nativa **mysql**. Esta extensión se mantiene en la actualidad para dar soporte a las aplicaciones ya existentes que la utilizan, pero no se recomienda utilizarla para desarrollar nuevos programas. Lo más habitual es elegir entre **mysqli** (extensión nativa) y **PDO**.

Con cualquiera de ambas extensiones, podrás realizar acciones sobre las bases de datos como:

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

**PDO y mysqli** (y también la antigua extensión **mysql**) utilizan un **driver de bajo nivel** para comunicarse con el servidor **MySQL**. Hasta hace poco el único controlador disponible para realizar esta función era **libmysql**, que no estaba optimizado para ser utilizado desde PHP. A partir de la versión **5.3**, viene preparado para utilizar también un nuevo driver mejorado para realizar esta función, el driver nativo de MySQL, **mysqlnd**.

## 3.1.- Extensión MySQLi.

Esta extensión se desarrolló para aprovechar las ventajas que ofrecen las versiones **4.1.3** y posteriores de **MySQL**, y viene incluida con PHP a partir de la versión **5**. Ofrece una interface de programación dual, pudiendo accederse a las funcionalidades de la extensión utilizando objetos o funciones de forma indiferente. Por ejemplo, para establecer una conexión con un servidor **MySQL** y consultar su versión, podemos utilizar cualquiera de las siguientes formas:

* new mysqli();
* mysqli\_connect();

En ambos casos, la variable **$conexion** es de tipo objeto. La utilización de los métodos y propiedades que aporta la clase **mysqli** normalmente produce un código más corto y legible que si utilizas llamadas a funciones.

Extensiones: <https://www.php.net/manual/es/book.mysqli.php>

Entre las mejoras que aporta respecto a la antigua extensión **mysql**, figuran:

* Interfaz orientada a objetos.
* Soporte para transacciones.
* Soporte para consultas preparadas.
* Mejores opciones de depuración y seguridad.

Las opciones de configuración se almacenan en el fichero "**php.ini"**. En este fichero hay una sección específica para las opciones de configuración propias de cada extensión. Entre las opciones que puedes configurar para la extensión **MySQLi**están:

* **mysqli.allow\_persistent.** Permite crear conexiones persistentes.
* **mysqli.default\_port**. Número de puerto TCP predeterminado a utilizar cuando se conecta al servidor de base de datos.
* **mysqli.reconnect**. Indica si se debe volver a conectar automáticamente en caso de que se pierda la conexión.
* **mysqli.default\_host**. Host predeterminado a usar cuando se conecta al servidor de base de datos.
* **mysqli.default\_user**. Nombre de usuario predeterminado a usar cuando se conecta al servidor de base de datos.
* **mysqli.default\_pw**. Contraseña predeterminada a usar cuando se conecta al servidor de base de datos.

En la documentación de PHP se incluye una lista completa de las **directivas** relacionadas con la extensión **MySQLi** que se pueden utilizar en "**php.ini"**. <https://www.php.net/manual/es/mysqli.configuration.php>

## 3.1.1.- Establecimiento de conexiones.

Para poder comunicarte desde un programa PHP con un servidor **MySQL**, el primer paso es establecer una conexión. Toda comunicación posterior que tenga lugar se hará utilizando esa conexión.

Si utilizas la extensión **MySQLi**, establecer una conexión con el servidor significa crear una instancia de la **clase mysqli**. El constructor de la clase puede recibir seis parámetros, todos opcionales, aunque lo más habitual es utilizar los cuatro primeros:

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

Si utilizas el constructor de la clase, para conectarte a la base de datos "**proyecto**" puedes hacer:

$conProyecto = new mysqli('localhost', 'gestor', 'secreto', 'proyecto');

Aunque también tienes la opción de primero crear la instancia, y después utilizar el método "**connect"** para establecer la conexión con el servidor:

$conProyecto = new mysqli();

$conProyecto->connect('localhost', 'gestor', 'secreto', 'proyecto');

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 0 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.

Por ejemplo, el siguiente código comprueba el establecimiento de una conexión con la base de datos "**proyecto**" y finaliza la ejecución si se produce algún error:

$conProyecto = new mysqli('localhost', 'gestor', 'secreto', 'proyecto');

if ($conProyecto->connect\_error) {

die('Error de Conexión (' . $conProyecto->connect\_errno . ') ' . $conProyecto->connect\_error);

//die() detiene la ejecución del script php

}

Por otro lado, es preferible utilizar **excepciones** para realizar el control de errores en PHP tal y como se hace por defecto a partir de la versión de PHP 8.1. Si estoy trabajando con una versión anterior de PHP y queremos activar la generación de excepciones durante el uso de la librería deberemos establecerlo mediante los **flags** de informe de errores **MYSQLI\_REPORT\_ERROR | MYSQLI\_REPORT\_STRICT**

Si quisiéramos realizar control de errores mediante el tratamiento de excepciones podríamos codificarlo de la siguiente manera:

$controlador = new mysqli\_driver();

$controlador->report\_mode = MYSQLI\_REPORT\_ERROR | MYSQLI\_REPORT\_STRICT;

try {

$conProyecto = new mysqli('localhost', 'gestor', 'secreto', 'proyecto');

} catch (mysqli\_sql\_exception $e) {

die ("Error en la conexión a la base de datos: " . $e->getMessage());

}

*Observa que, como veremos posteriormente con más detalle, puedes anteponer a cualquier expresión el operador de* ***control de errores*** *"****@"****para que se ignore cualquier posible error que pueda producirse al ejecutarla.* [*https://www.php.net/manual/es/language.operators.errorcontrol.php*](https://www.php.net/manual/es/language.operators.errorcontrol.php)

Si 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 ddbb, lógicamente el usuario con el que hemos iniciado la conexión debe tener permisos en la nueva.

$conProyecto->select\_db('otra\_bd');

Una vez finalizadas las tareas con la base de datos, utiliza el método **"close"** (o la función "**mysqli\_close"**) para cerrar la conexión con la base de datos y liberar los recursos que utiliza.

$conProyecto->close();

## 3.1.2.- Ejecución de consultas.

La forma más inmediata de ejecutar una consulta, si utilizas esta extensión, es el método **query**, equivalente a la función **mysqli\_query**. Si se ejecuta una consulta de acción que no devuelve datos (como una sentencia SQL de tipo **UPDATE**, **INSERT** o **DELETE**), la llamada devuelve **true** si se ejecuta correctamente o **false** en caso contrario. El número de registros afectados se puede obtener con la propiedad **affected\_rows** (o con la función **mysqli\_affected\_rows**).

Al ejecutar una sentencia SQL que sí devuelva datos (como un **SELECT**), éstos se devuelven en forma de un objeto resultado (de la clase **"mysqli\_result**"). En el punto siguiente verás cómo se pueden manejar los resultados obtenidos.

El método **"query()"** tiene un parámetro opcional que afecta a cómo se obtienen internamente los resultados, pero no a la forma de utilizarlos posteriormente. En la opción por defecto, **MYSQLI\_STORE\_RESULT**, los resultados se recuperan todos juntos de la base de datos y se almacenan de forma local. Si cambiamos esta opción por el valor **MYSQLI\_USE\_RESULT**, los datos se van recuperando del servidor según se vayan necesitando.

$resultado = $conProyecto->query('SELECT producto, unidades FROM stocks', MYSQLI\_USE\_RESULT);

*Otra forma que puedes utilizar para ejecutar una consulta es el método****real\_query****(o la función****mysqli\_real\_query****), que siempre devuelve true o false según se haya ejecutado correctamente o no. Si la consulta devuelve un conjunto de resultados, se podrán recuperar de forma completa utilizando el método****store\_result****, o según vaya siendo necesario gracias al método****use\_result****.* [*https://www.php.net/manual/es/mysqli.real-query.php*](https://www.php.net/manual/es/mysqli.real-query.php)

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

$resultado->free();

## 3.1.3.- 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**).

$conProyecto->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.

...

$conProyecto->query('DELETE FROM stocks WHERE tienda=2'); // Inicia una transacción

$conProyecto->query('UPDATE stocks SET unidades=3 WHERE producto=24');

. . .

$conProyecto->commit(); // Confirma los cambios

Una vez finalizada esa transacción, comenzará otra de forma automática.

## 3.1.4.- Obtención y utilización de conjuntos de resultados.

Ya sabes que al ejecutar una consulta que devuelve datos obtienes un objeto de la clase **mysqli\_result**. Esta clase sigue los criterios de ofrecer una interfaz de programación dual, es decir, una función por cada método con la misma funcionalidad que éste.

Para trabajar con los datos obtenidos del servidor, tienes varias posibilidades:

* **fetch\_array** (función **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.

Este comportamiento por defecto se puede modificar utilizando un parámetro opcional, que puede tomar los siguientes valores:

1. **MYSQLI\_NUM**. Devuelve un array con claves numéricas.
2. **MYSQLI\_ASSOC**. Devuelve un array asociativo.
3. **MYSQLI\_BOTH**. Es el comportamiento por defecto, en el que devuelve un array con claves numéricas y asociativas.

* **fetch\_assoc** (función **mysqli\_fetch\_assoc**). Idéntico a **fetch\_array** pasando como parámetro **MYSQLI\_ASSOC**.
* **fetch\_row** (función **mysqli\_fetch\_row**). Idéntico a **fetch\_array** pasando como parámetro **MYSQLI\_NUM**.
* **fetch\_object** (función **mysqli\_fetch\_object**). Similar a los métodos anteriores, pero devuelve un objeto en lugar de un array. Las propiedades del objeto devuelto se corresponden con cada uno de los campos del registro.

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

Clase mysqli\_result: <https://www.php.net/manual/es/class.mysqli-result.php>

Bootstrap: <https://getbootstrap.com/docs/4.1/getting-started/introduction/>

## 3.1.5.- Consultas preparadas.

Cada vez que se envía una consulta al servidor, éste 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.

Por otra parte, existe un riesgo de seguridad muy importante al usar formularios para **insertar, consultar, modificar, borrar** datos en una base de datos, la "**inyección SQL**”. Unos de los métodos que se recomiendan para evitar este tipo de ataques es precisamente usar consultas parametrizadas ya que los valores de los parámetros son transmitidos después, usando un protocolo diferente y no necesitan ser escapados.

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.

$conProyecto = new mysqli('localhost', 'gestor', 'secreto', 'proyecto');

$stmt = $conProyecto->stmt\_init();

Los pasos que debes seguir para ejecutar una consulta preparada son:

* Preparar la consulta en el servidor **MySQL** utilizando el método **prepare** (función **mysqli\_stmt\_prepare**).
* Ejecutar la consulta, tantas veces como sea necesario, con el método **execute** (función **mysqli\_stmt\_execute**).
* 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 tabla familia:

try {

$stmt->prepare(INSERT INTO familias (cod, nombre) VALUES ("TABLET", "Tablet PC")');

$stmt->execute();

} catch (mysqli\_sql\_exception $e) {

die("Error en la preparación o ejecución: " . $e->getMessage());

}

$stmt->close();

$conProyecto->close();

El problema observado 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.

$stmt->prepare('INSERT INTO familias (cod, nombre) VALUES (?, ?)');

Y 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.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

*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:*

$stmt->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.

Consultas preparadas y la clase **mysqli\_stmt**: <https://www.php.net/manual/es/class.mysqli-stmt.php>

## 3.2.- PHP Data Objects (PDO).

**Si vas a programar una aplicación que utilice como sistema gestor de bases de datos MySQL**, la extensión **MySQLi** que acabas de ver es una buena opción. Ofrece acceso a todas las características del motor de base de datos, a la vez que reduce los tiempos de espera en la ejecución de sentencias.

Sin embargo, **si en el futuro tienes que cambiar el**SGBD por otro distinto, tendrás que volver a programar gran parte del código de esta. Por eso, antes de comenzar el desarrollo, es muy importante revisar las características específicas del proyecto. En el caso de que exista la posibilidad, presente o futura, de utilizar otro servidor como almacenamiento, deberás adoptar una capa de abstracción para el acceso a los datos. Existen varias alternativas como **ODBC**, pero sin duda la opción más recomendable en la actualidad es **PDO**.

El objetivo es que si llegado el momento necesitas cambiar el servidor de base de datos, las modificaciones que debas realizar en tu código sean mínimas. Incluso es posible desarrollar aplicaciones preparadas para utilizar un almacenamiento u otro según se indique en el momento de la ejecución, pero éste no es el objetivo principal de PDO. **PDO** no **abstrae** de forma completa el sistema gestor que se utiliza. Por ejemplo, no modifica las sentencias SQL para adaptarlas a las características específicas de cada servidor. Si esto fuera necesario, habría que programar una capa de abstracción completa.

La extensión PDO debe utilizar un controlador específico para el tipo de base de datos que se utilice. Para consultar los controladores disponibles en tu instalación de PHP, puedes utilizar la información que proporciona la función **phpinfo()**.

PDO se basa en las características de orientación a objetos de PHP, pero al contrario que la extensión **MySQLi**, no ofrece un interface de programación dual. Para acceder a las funcionalidades de la extensión tienes que emplear los objetos que ofrece, con sus métodos y propiedades. No existen funciones alternativas.

*En la unidad****5****se trata el tema de POO, veremos que la forma más potente de sacar partido a PDO es, precisamente, crearnos unas clases para cada una de las****tablas****de la base de datos e implementar en ellas todos los métodos para hacer un****CRUD****a la tabla.*

## 3.2.1.- 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**):

* Origen de datos (**DSN**). Es una cadena de texto que indica qué controlador se va a utilizar y a continuación, separadas por el carácter dos puntos, los parámetros específicos necesarios por el controlador, como por ejemplo el nombre o dirección IP del servidor y el nombre de la base de datos.
* Nombre de usuario con permisos para establecer la conexión.
* Contraseña del usuario.
* Opciones de conexión, almacenadas en forma de array.

Podemos establecer una conexión con la base de datos 'proyecto' creada anteriormente:

$host = "localhost";

$db = "proyecto";

$user = "gestor";

$pass = "secreto";

$dsn = "mysql:host=$host;dbname=$db";

$conProyecto=new PDO($dsn, $user, $pass);

//se recomienda guardar los datos(host, user...) en variables porque si estos cambian

//solo tenemos que actualizar el valor de estas variables

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

* **host**. Nombre o dirección IP del servidor.
* **port**. Número de puerto TCP en el que escucha el servidor.
* **dbname**. Nombre de la base de datos.
* **unix\_socket**. Socket de **MySQL** en sistemas Unix.

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

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

Más información sobre los controladores existentes, los parámetros de las cadenas DSN y las opciones de conexión particulares de cada uno: <https://www.php.net/manual/es/pdo.drivers.php>

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

$version = $conProyecto->getAttribute(PDO::ATTR\_SERVER\_VERSION);

echo "Versión: $version";

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

$version = $conProyecto->setAttribute(PDO::ATTR\_CASE, PDO::CASE\_UPPER);

Y muy importante para controlar los errores tendremos el atributo: **ATTR\_ERRMODE** con los posibles valores:

* **ERRMODE\_SILENT**: El modo por defecto, no muestra errores (se recomienda en entornos en producción).
* **ERRMODE\_WARNING***:*Además de establecer el código de error, emitirá un mensaje **E\_WARNING**, es el modo empleado para depurar o hacer pruebas para ver errores sin interrumpir el flujo de la aplicación.
* **ERRMODE\_EXCEPTION***:*Además de establecer el código de error, lanzará una **PDOException** que podemos capturar en un bloque **try catch()***.*Lo veremos en el apartado **4.1.**

$conProyecto->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);

De esta manera, podemos controlar el lanzamiento de excepciones en la operación de conexión con el código siguiente:

$host = "localhost";

$db = "proyecto";

$user = "gestor";

$pass = "secreto";

$dsn = "mysql:host=$host;dbname=$db";

try {

$conProyecto=new PDO($dsn, $user, $pass);

$conexion->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);

} catch (PDOException $e) {

die("Error en la conexión: " . $e->getMessage());

}

Para **cerrar** la conexión hay que saber que la misma permanecerá activa durante el tiempo de vida del objeto PDO. Para cerrarla, es necesario destruir el objeto asegurándose de que todas las referencias a él existentes sean eliminadas; esto se puede hacer asignando **null** a la variable que contiene el objeto.

$conProyecto = null;

Las funciones **getAttribute:**  <https://www.php.net/manual/es/pdo.getattribute.php>y **setAttribute:** <https://www.php.net/manual/es/pdo.setattribute.php>

## 3.2.2.- 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 las consultas de acción, como **INSERT, DELETE o UPDATE**, el método **exec** devuelve el número de registros afectados.

<?php

$host = "localhost";

$db = "proyecto";

$user = "gestor";

$pass = "secreto";

$dsn = "mysql:host=$host;dbname=$db";

try {

$conProyecto = new PDO($dsn, $user, $pass);

$conProyecto->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);

$registros = $conProyecto->exec('DELETE FROM stocks WHERE unidades=1');

} catch (PDOException $e) {

die("Error en la conexión o la consulta: " . $e->getMessage());

}

$conProyecto = null;

?>

<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**.

$host = "localhost";

$db = "proyecto";

$user = "gestor";

$pass = "secreto";

$dsn = "mysql:host=$host;dbname=$db";

$conProyecto=new PDO ($dsn, $user, $pass);

$conProyecto->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION)</span>;

$resultado = $conProyecto->query("SELECT</span> producto, unidades FROM stock");

## 3.2.3.- Transacciones

Por defecto PDO trabaja en modo "**autocommit**", esto es, confirma de forma automática cada sentencia que ejecuta el servidor. Para trabajar con transacciones, PDO incorpora **tres métodos:**

* **beginTransaction**. Deshabilita el modo "**autocommit**" y comienza una nueva transacción, que finalizará cuando ejecutes uno de los dos métodos siguientes.
* **commit**. Confirma la transacción actual.
* **rollback**. Revierte los cambios llevados a cabo en la transacción actual.

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

$ok = true;

$conProyecto->beginTransaction();

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

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

…

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

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

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

## 3.2.4.- Obtención y utilización de conjuntos de resultados.

Con **PDO** tienes varias posibilidades para tratar con el conjunto de resultados devuelto por el método **query**. La más utilizada es el método **fetch** de la clase **PDOStatement**. Este método devuelve un registro del conjunto de resultados, o **false** si ya no quedan registros por recorrer.

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\_CLASS**. Devuelve una nueva instancia de la clase solicitada, haciendo corresponder las columnas del conjunto de resultados con los nombres de las propiedades de la clase, y llamando al constructor después, a menos que también se proporcione **PDO::FETCH\_PROPS\_LATE**.
* **PDO::FETCH\_OBJ**. Devuelve un objeto cuyas propiedades se corresponden con los campos del registro.
* **PDO::FETCH\_LAZY**. Devuelve tanto el objeto como el array con clave dual anterior.
* **PDO::FETCH\_BOUND**. Devuelve true y asigna los valores del registro a variables, según se indique con el método **bindColumn**. Este método debe ser llamado una vez por cada columna, indicando en cada llamada el número de columna (empezando en 1) y la variable a asignar.

También podemos utilizar **fecthAll()** que te trae todos los datos de golpe, sin abrir ningún puntero, almacenándolos en un array. Se recomienda cuando no se esperan demasiados resultados, que podrían provocar problemas de memoria al querer guardar de golpe en un array muchas filas provenientes de una consulta.

## 3.2.5.- Consultas preparadas.

Al igual que con **MySQLi**, también utilizando **PDO** podemos preparar consultas parametrizadas en el servidor para ejecutarlas de forma repetida. El procedimiento es similar e incluso los métodos a ejecutar tienen prácticamente los mismos nombres.

Para preparar la consulta en el servidor **MySQL**, deberás utilizar el método **prepare** de la clase PDO. Este método devuelve un objeto de la clase **PDOStatement**. Los parámetros se pueden marcar utilizando signos de interrogación como en el caso anterior.

$conProyecto = new PDO(". . .");

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

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

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

Antes de ejecutar la consulta hay que asignar un valor a los parámetros utilizando el método **bindParam** o **bindValue** de la clase **PDOStatement**. Si utilizas parámetros con nombre, debes indicar ese nombre en la llamada a **bindParam** o **bindValue**.

* public PDOStatement::bindParam( mixed $parameter, mixed &$variable, int $data\_type = PDO::PARAM\_STR, int $length = ?, mixed $driver\_options = ?): bool -> bindParam vincula una variable a un parámetro de la consulta preparada. Cada vez que se ejecuta la consulta se usa el último valor almacenado en la variable.
* public PDOStatement::bindValue(mixed $parameter, mixed $value, int $data\_type = PDO::PARAM\_STR): bool -> bindValue vincula un literal o el valor almacenado en una variable. Cuando se ejecuta la consulta preparada se usa el valor en el momento de la creación del vínculo aunque el valor almacenado en la variable haya cambiado.

*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 el ejemplo anterior.*

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

* public**PDOStatement::execute**(array $input\_parameters = ?): bool Devuelve true en caso de éxito o false en caso de error.
* $stmt->execute();

También existe otra forma de pasar valores a los parámetros. Hay un método [lazy](https://aulavirtual32.educa.madrid.org/ies.lapaz.alcobendas/pluginfile.php/66460/mod_resource/content/9/index.html" \l "t84d2cd9f-cc47-30b4-0e89-f11b2ba73144" \o "), que funciona pasando los valores mediante un array, al método **execute()**.

$nombre="Monitores";

$codigo="MONI";

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

$stmt->execute([ ':cod'=>$codigo, ':nombre'=>$nombre]);

]);

En el caso de ejecutar una sentencia que devuelva resultados como es el caso de SELECT, podremos acceder a el conjunto de resultados mediante el método **fetch** y **fetchAll** tal y como presentamos en la sección anterior.

Consultas preparadas y la clase PDOStatement: <https://www.php.net/manual/es/class.pdostatement.php>

# 4.- Errores y manejo de excepciones.

A buen seguro que, conforme has ido resolviendo ejercicios o simplemente probando código, te has encontrado con errores de programación. Algunos son reconocidos por el entorno de desarrollo, y puedes corregirlos antes de ejecutar. Otros aparecen en el navegador en forma de mensaje de error al ejecutar el guion.  
PHP define una clasificación de los errores que se pueden producir en la ejecución de un programa y ofrece métodos para ajustar el tratamiento de estos. Para hacer referencia a cada uno de los niveles de error, PHP define una serie de constantes. 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.

Lista completa de constantes en PHP: <https://www.php.net/manual/es/errorfunc.constants.php>

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.
* **log\_errors**. En su valor por defecto (On), indica si los mensajes de error del script deberían de registrarse en el registro del servidor o en error\_log.

Parámetros que podemos ajustar en php.ini: <https://www.php.net/manual/es/errorfunc.configuration.php>

La gestión de errores en **PHP** ha ido evolucionando con el tiempo y ha ido mejorando con las sucesivas versiones del lenguaje. Inicialmente, el motor de ejecución **PHP** generaba distintos tipos de errores y podían ser manejados por una función de usuario siempre que fueran no fatales. El uso de Excepciones (**Exceptions**) se incorpora en el lenguaje a partir de **PHP 5** y permite manejar los errores con una filosofía más acorde a la programación OO. A partir de la versión **PHP 7** los errores fatales y no fatales son traducidos a excepciones del sistema con lo que ya pueden capturarse y manejarse en el código de la aplicación. En la versión **PHP 8** ambos mecanismos conviven, aunque hay una tendencia clara a fomentar el uso de excepciones en detrimento del mecanismo original basado en errores.

A partir de **PHP 7** el motor de ejecución solo genera excepciones. Sin embargo, todavía es posible generar errores de usuario y manejarlos a nivel global. El manejador de errores por defecto abortaría el script si se trata de un error fatal o continuaría con la ejecución si el error recibido no es fatal. Además, seguiría las indicaciones de los parámetros **display\_errors** y **log\_errors** para mostrarlo en pantalla o añadirlo al registro de mensajes de error. Por otro lado, es posible definir un manejador que permita refinar el manejo de los errores con código de usuario. Para estas tareas disponemos de las siguientes funciones:

* **trigger\_error**(string $error\_msg, int $error\_type = E\_USER\_NOTICE): bool Se usa para provocar una condición de error de usuario, se puede utilizar junto con el gestor de errores interno o con una función definida por el usuario que ha sido establecida como el nuevo gestor de errores (set\_error\_handler()).
* **set\_error\_handler**(callable $error\_handler, int $error\_types = E\_ALL | E\_STRICT): mixed Establece una función de usuario (error\_handler) para manejar los errores de un script. Los siguientes tipos de errores no pueden ser manejados con una función definida por el usuario: **E\_ERROR**, **E\_PARSE**, **E\_CORE\_ERROR**, **E\_CORE\_WARNING**, **E\_COMPILE\_ERROR**, **E\_COMPILE\_WARNING**, y la mayoría de **E\_STRICT** ocasionados en el archivo desde donde se llamó a **set\_error\_handler()**.

A continuación, se muestra un ejemplo de uso de estas funciones:

Funciones de Manejo de Errores: <https://www.php.net/manual/es/ref.errorfunc.php>

## 4.1.- Excepciones.

Las excepciones están muy ligadas al paradigma de programación orientado a objetos y se introducen en la versión PHP 5. Una excepción es generada por el código cuando el estado de la ejecución entra en un estado inesperado. En ese momento el flujo de control del programa salta al manejador que trata la anomalía de la mejor manera posible. Las excepciones ascienden por la pila de ejecución si no son manejadas adecuadamente hasta el nivel superior donde se aplica el manejador de excepciones de más alto nivel.

* 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**.
* Si se ha generado una excepción en el bloque **try** se comprobará si el bloque **catch** puede manejar dicha excepción.
* El bloque opcional **finally** se ejecuta en cualquiera de los escenarios, se haya lanzado la excepción o no.

Las excepciones ascienden por la pila de llamadas (call stack) que se haya producido hasta llegar a la ejecución del código que lanza la excepción. EL manejador de excepciones puede ubicarse en cualquier nivel de la cadena de llamadas.

Es posible establecer múltiples bloques de captura **catch** que definan la estrategia para tratar distintos tipos de excepciones.

Es importante que la información contenida en el objeto **Exception** sea suficientemente detallada para facilitar su tratamiento y/o posterior registro. El objeto **Exception** proporciona el siguiente **API** para obtener toda esa información.

* **final public getMessage()**: string Obtiene el mensaje de la excepción
* **final public getPrevious()**: Throwable Devuelve la excepción anterior
* **final public getCode()**: mixed Devuelve el código de una excepción
* **final public getFile()**: string Obtiene el nombre del fichero en el que fue creada la excepción.
* **final public getLine()**: int Devuelve el número de la línea donde se creó la excepción.
* **final public getTrace()**: array Devuelve la traza de pila de una excepción.
* **final public Exception::getTraceAsString()**: string Devuelve la traza de la pila de una excepción como una cadena de caracteres.

PHP ofrece una jerarquía de clases que cubren la variedad de tipos de excepciones que pueden ser generadas internamente y otras genéricas que pueden reutilizarse por los programadores. La jerarquía de clases es la siguiente:

|  |
| --- |
| interface Throwable  |- Error implements Throwable  |- CompileError extends Error  |- ParseError extends CompileError  |- TypeError extends Error  |- ArgumentCountError extends TypeError  |- ArithmeticError extends Error  |- DivisionByZeroError extends ArithmeticError  |- AssertionError extends Error  |- Exception implements Throwable  |- ClosedGeneratorException  |- DOMException  |- ErrorException  |- IntlException  |- LogicException  |- BadFunctionCallException  |- BadMethodCallException  |- DomainException  |- InvalidArgumentException  |- LengthException  |- OutOfRangeException  |- PharExceptionaddition  |- ReflectionException  |- RuntimeException  |- mysqli\_sql\_exception  |- OutOfBoundsException  |- OverflowException  |- PDOException  |- RangeException  |- UnderflowException  |- UnexpectedValueException  |- Custom Exception |

Todas las clases de las excepciones implementan el *interfaz* **Throwable**, lo que permite que sus objetos tengan el comportamiento de excepción visto anteriormente. Por otro lado, las clases de excepciones se van agrupando de la siguiente manera:

* Clase **Error**. Es la clase base para todos los errores internos de PHP. Las subclases generan excepciones que representan errores de compilación, de tipo, aritméticos y de aserción.
* Clase **Exception**. Estas excepciones se generan cuando ocurre alguna condición inesperada en el código y el flujo de ejecución normal debe de sustituirse por la ejecución del manejador correspondiente. Como puedes ver existen multitud de situaciones previstas en la jerarquía de clases.
* Además de las clases predefinidas en el lenguaje, es posible que el programador necesite definir nuevas clases para de excepciones para que las situaciones anómalas contempladas se ajusten mejor a la aplicación.

Vamos a ver algún ejemplo que muestra los conceptos descritos anteriormente.

El ejemplo realiza una comprobación de los tipos de los parámetros $nombre y $edad y lanza excepciones si los valores no son lo adecuados. Se definen varios manejadores para tratar los escenarios de error.

Se crea una función **checkTipo** que comprueba los tipos de los valores y genera excepciones si detecta un error. Las clases de usuario **cadenaException** y **numericoException** son tipos de excepciones creadas por el programador de manera específica para esta aplicación. Esta clase redefine el mensaje de error mostrado cuando se genera la excepción. Se invoca la función **checkTipo** en un bloque **try** de manera que si se lanza una excepción se captura en el bloque **catch**.

Es posible definir un manejador global de excepciones utilizando la función:

* **set\_exception\_handler(?callable $callback): ?callable** Establece el manejador de excepciones predeterminado si una excepción no es capturada dentro de un bloque **try/catch**. La ejecución se detendrá después de la llamada a exception\_handler.

Un ejemplo de este manejador se presenta a continuación.

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** (veremos cómo utilizar la herencia en una unidad posterior).

## 4.2.- Manejador global de excepciones

El código que se muestra a continuación puede utilizarse como manejador global de errores y excepciones. Puedes incluir este código en aquellos scripts que requieran gestión de errores y de excepciones.

Vamos a estudiar el código en detalle:

* La invocación **error\_reporting(ALL)** nos garantiza que vamos a informar de todos los tipos posibles de errores para que podamos resolverlos adecuadamente en la fase de desarrollo
* Convertimos los errores de usuario que puedan generarse durante la ejecución del script en excepciones para homogeneizar el tipo de errores a tratar. Este comportamiento lo logramos en la función utilizada como parámetro a la función **set\_error\_handler**.
* Definimos un manejador global de excepciones que guarda el texto del error en un registro con **error\_log** y decide si quiere mostrar el mensaje de error en pantalla o no dependiendo del valor del parámetro **display\_errors** que indica si la aplicación se está ejecutando en un contexto de desarrollo o de producción.

## 4.3.- Manejo de excepciones en PDO

Vimos en el apartado **3.2.1**que la clase **PDO** permitía definir la fórmula que usará cuando se produzca un error, utilizando el atributo **PDO::ATTR\_ERRMODE**. Las posibilidades eran:

* **PDO::ERRMODE\_SILENT**.
* **PDO::ERRMODE\_WARNING**.
* **PDO::ERRMODE\_EXCEPTION**. Cuando se produce un error lanza una excepción utilizando el manejador propio **PDOException.**

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

$conProyecto->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);

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

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

Visto el funcionamiento básico de la generación y captura de expeciones en PDO, conviene aclarar que no es necesario envolver todas las operaciones de PDO que pueden generar excepciones en un bloque try .. catch, sobre todo si la acción a tomar es terminar el script y mostrar el mensaje de error como en el ejemplo anterior. Desde luego, este comportamiento no sería el deseado si la aplicación estuviera en producción.

Por lo tanto, solo debes capturar excepciones en los siguientes escenarios:

* Si necesitas transfomar las excepciones PDO en excepciones más significativas para tu aplicación. La excepción de PDO puede usarse para construir y relanzar una nueva excepción de usuario más fácil de manejar por el resto de la aplicación.
* Si se debe tomar una acción en caso de fallo, por ejemplo realizar el **rollback** de una transacción.
* Si puedes tomar alguna medida correctora que resuelva el problema sin terminar el script. Puede ser que tengas que estudiar los datos internos de la excepción para tratar solo algún ejemplar de la excepción y/o volver a relanzarla si se trata de otro tipo. Por ejemplo, si se produce una excepción cuando se hace una inserción en la BBDD puede que necesite saber el código de error de MySQL para saber si puedo solucionar el problema o no. Por ejemplo:

FIN