

Trabajar con bases de datos en PHP.

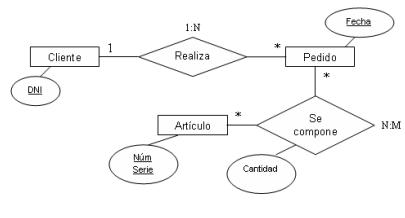
Caso práctico



Una de las tareas prioritarias que tienen que abordar en el nuevo proyecto de **BK Programación** es el almacenamiento de la información que utilizará la aplicación web, y el método de acceso que se utilizará para manejarla desde PHP.

En una reunión de trabajo, **Juan** les informa que para la gestión de la empresa están utilizando una aplicación de código libre que almacena los datos en un servidor MySQL. Afortunadamente, este servidor es el más utilizado en la programación con lenguaje **PHP**, por lo que no tendrán problemas en integrar la nueva aplicación web con la ya existente. Solo necesitan conocer la estructura de los datos que se almacenan, y ver qué métodos pueden usar para manejar la información.

En la unidad anterior aprendiste a utilizar las principales estructuras de programación en lenguaje **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. Es por ello que la primera parte de esta unidad se centra en revisar los principales conceptos sobre la utilización del mismo. A continuación aprenderás a acceder desde **PHP** a bases de datos MySQL utilizando las extensiones **MySQLi** y **PDO**.



[Afpineda \(CC BY-SA\)](#)



[Ministerio de Educación y Formación Profesional](#) (Dominio público)

Materiales formativos de FP Online propiedad del Ministerio de Educación y Formación Profesional.

[Aviso Legal](#)

1.- Acceso a bases de datos desde PHP.

Caso práctico

Carlos es nuevo en el mundo de la programación web. Además, apenas ha trabajado con bases de datos, por lo que se asombra de la gran diversidad de opciones que existen en PHP para trabajar con datos almacenados en servidores de distintos tipos.



Algunos de los gestores sobre los que lee mientras revisa la documentación de PHP los conoce, otros simplemente le suenan, pero hay muchos de los que ni siquiera conocía su existencia. Sabe que debe centrarse en el servidor MySQL, que es el que usarán para desarrollar la aplicación, pero aun así el volumen de información disponible es tan grande que le cuesta decidirse por dónde empezar.

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



[rg1024](#) (Dominio público)

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 a utilizar eran distintos para cada extensión.

A partir de la versión 5 de PHP se introdujo en el lenguaje una extensión para acceder de una forma común a distintos sistemas gestores: PDO. 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.

En esta unidad vas a ver 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. La unidad 4 la dedicaremos a la POO en PHP pero veamos unas nociones básicas.

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();
```

Debes conocer

Es importante conocer las características más básicas de la utilización objetos en PHP.

[Características más básicas.](#)

2.- MySQL.

Caso práctico



Juan y Carlos deciden comenzar revisando el servidor que van a utilizar, MySQL. Aunque van a utilizar un servidor que ya está en funcionamiento, deben comprender sus capacidades y las herramientas de las que disponen para poder gestionar tanto el servidor como los datos que almacena.

María conoce bien MySQL y les orienta sobre los pasos necesarios para instalarlo y configurarlo. Con su ayuda, hacen una copia a algunos de los datos que necesitan, y los replican en un servidor local para poder trabajar con ellos. Por supuesto, se aseguran de no utilizar para las pruebas información sensible como la de los clientes o proveedores, que pueda ocasionarles problema legales.

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).



[Oracle](#) (Todos los derechos reservados)

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



[MariaDB Corporation Ab](#) (Todos los derechos reservados)

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.

Para saber más

En las siguientes secciones darás un rápido repaso a lo que debes saber sobre la instalación, configuración y las herramientas de administración de MySQL. Si necesitas ampliar información, puedes consultar el manual en línea de MySQL.

[Documentación en linea, en formato PDF de MySQL.](#)

Autoevaluación

¿A qué hacen referencia las siglas PDO?

- A un motor de almacenamiento utilizado por MariaDB o MySQL.
- A una extensión de PHP que permite acceder a varios gestores de bases de datos.

Incorrecto, revisa los distintos motores que se pueden utilizar con MySQL.

Correcto, los motores de almacenamiento de los que hablamos son MyISAM e InnoDB y sus equivalentes en MariaDB. Aunque no son los únicos que se pueden utilizar sí son los más comunes.

Solución

1. Incorrecto

2. Opción correcta

2.1.- Instalación y configuración.

En la primera unidad ya viste cómo podías instalar en un único paso una plataforma **LAMP** para desarrollar aplicaciones web en **Ubuntu**. En Linux, la instalación de MySQL se divide básicamente en dos paquetes que puedes instalar de forma individual según tus necesidades:

- ✓ **mysql-server**. Es el servidor en sí. Necesitas instalar este paquete para gestionar las bases de datos y permitir conexiones desde el equipo local o a través de la red.
- ✓ **mysql-client**. Son los programas cliente, necesarios para conectarse a un servidor MySQL. Solo necesitas instalarlos en aquel o aquellos equipos que se vayan a conectar (en nuestro caso, las conexiones se realizarán normalmente desde el mismo equipo en el que se ejecuta el servidor).



dannya (Dominio público)

Una vez instalado, puedes gestionar la ejecución del servicio de la misma forma que cualquier otro servicio del sistema:

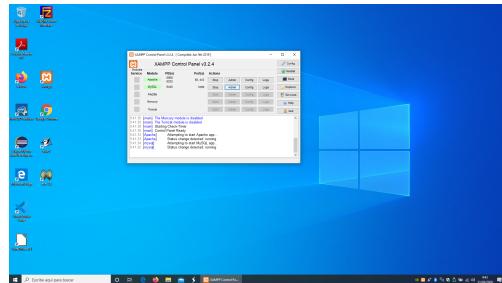
- ✓ `sudo systemctl start mysql` (inicia el servicio).
- ✓ `sudo systemctl stop mysql` (detiene el servicio).
- ✓ `sudo systemctl restart mysql` (reinicia el servicio).
- ✓ `sudo systemctl status mysql` (nos da información del servicio).

Si hemos instalado **LAMP** en Ubuntu, por defecto, MySQL/MariaDB viene sin password para el usuario **root** y con un plugin de autentificación (`auth_socket`) que nos obliga por defecto a usar un usuario de Ubuntu con permiso de administrador, es decir la forma de entrar como usuario **root** en MySQL será: `sudo mysql -u root` . La recomendación es, que esta configuración no se cambie. Los pasos recomendados cuando empecemos un proyecto **PHP** el que necesitemos una Base de Datos y acceso a ella serán:

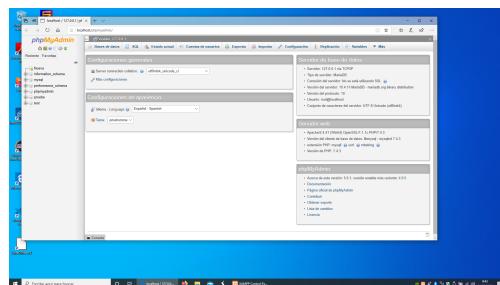
- ✓ Nos conectamos como **root** a MySQL: `sudo mysql -u root` (terminal linux).
- ✓ Creamos la base de datos.: `create database proyecto;` (estamos en MySQL/MariaDB).
- ✓ Creamos un usuario y le asigno un password: `create user usuario@'localhost' identified by "password";` (estamos en MySQL/MariaDB).
- ✓ Le doy permiso al usuario en la base de datos anterior: `grant all on proyecto.* to usuario@'localhost';` (estamos en MySQL/MariaDB).
- ✓ Salimos: `quit;` (estamos en MySQL/MariaDB y salimos a la terminal)
- ✓ Comprobamos que todo funciona conectándonos con este usuario a la bases de datos creada: `mysql -u usuario -p proyecto` (terminal linux).

Luego ya podemos usar PhpMyAdmin, o cualquier otro programa, con las credenciales de este usuario para administrar la base de datos.

Si hemos instalado **Xampp** en Windows podemos acceder directamente a phpMyAdmin desde el panel de control (pulsando en Admin al lado de MySQL), por defecto ya entramos como usuario **root**.



Xampp (Elaboración propia.)



phpmyadmin (Elaboración propia)

El servidor se ejecuta por defecto en el Puerto TCP 3306. Esto lo debes tener en cuenta para permitir el acceso a través del cortafuegos en configuraciones en red.

El fichero de configuración del servidor **MySQL** se llama `my.cnf` y se encuentra alojado en `/etc/mysql`. Su contenido se divide en secciones. Las opciones que contiene cada una de las secciones afectan al comportamiento de un módulo concreto. Entre las secciones disponibles destacan:

- ✓ **[client]**. Sus parámetros influyen sobre los distintos clientes que se conectan al servidor MySQL.
- ✓ **[mysqld]**. Contiene opciones relativas a la ejecución del servidor.

Entre los parámetros que puedes configurar en el fichero `my.cnf` tienes:

- ✓ **port**. Indica el puerto TCP en el que escuchará el servidor y con el que se establecerán las conexiones.
- ✓ **user**. Nombre del usuario que se utilizará para ejecutar el servidor.
- ✓ **datadir**. Directorio del servidor en el que se almacenarán las bases de datos.

Para saber más

En el manual propuesto en el Punto 2 tienes también información sobre todas las opciones de configuración disponibles para ajustar su funcionamiento.

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 un interface gráfico 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.
- ✓ **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 usara "localhost".

Por ejemplo, para establecer una conexión al servidor **local** con la herramienta **mysql**, podemos hacer:

- ✓ Si queremos conectarnos como **root** y hemos hecho caso de la recomendación del punto 2.1: `sudo mysql -u root` (necesitamos un usuario linux con permisos de administrador). Si estamos en el sistema como usuario **root** (muy mala idea) simplemente: `mysql -u root`.
- ✓ Si estuviésemos en **Windows** 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 ejecutables del sistema) basta con teclear "`mysql -u root`".
- ✓ Para entrar con cualquier otro usuario: `mysql -u usuario -p baseDeDatos`. Por ejemplo si soy el usuario "**juan**" y quiero entrar directamente a la base de datos: "**proyecto**" el comando sería: "`mysql -u juan -p proyecto`".



[Moini](#) (Dominio público)

Recomendación

Conviene no indicar nunca la contraseña en la misma línea de comandos. En caso de que la cuenta esté convenientemente protegida por una contraseña, es mejor utilizar solo la opción **-p** como en el ejemplo anterior. De esta forma, la herramienta solicita la introducción de la contraseña y ésta no queda almacenada en ningún registro como puede ser el historial de comandos del sistema.

Debes conocer

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

MySQL Workbench es una herramienta genérica con interface 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.

[MySQL Workbench](#).

phpMyAdmin es una aplicación web muy popular para la administración de servidores MySQL. Presenta un interface 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.

[phpMyAdmin](#).

Autoevaluación

Relaciona cada herramienta de administración con el tipo de interface que utiliza:

Ejercicio de relacionar

Herramienta.	Relación.	Tipo de interface.
MySQL Workbench.	<input type="checkbox"/>	1. Línea de comandos (administración).
mysql.	<input type="checkbox"/>	2. Web.
phpMyAdmin.	<input type="checkbox"/>	3. Nativo.
mysqladmin.	<input type="checkbox"/>	4. Línea de comandos.

Enviar

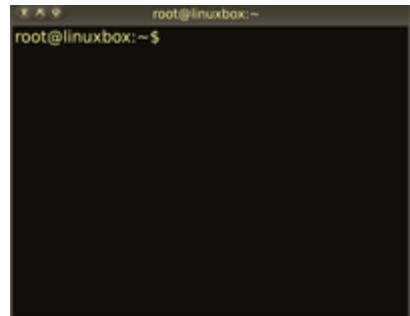
Es importante conocer las diferentes herramientas de administración de MySQL.

2.2.1.- mysql y 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.

[shokunin \(Dominio público\)](#)



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

Las sentencias que tecleas 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`). Por ejemplo y suponiendo que nos encontramos en el directorio donde se encuentra el archivo `crearTablas.sql` y queremos que este archivo se ejecute en la base de datos proyecto:

```
mysql -u usuario -p proyecto < crearTablas.sql
```

Si todo ha ido bien volveremos a la consola en la que estemos trabajando, no nos aparece ningún mensaje, si algo ha ido mal nos aparecerá el mensaje de error.

Recomendación

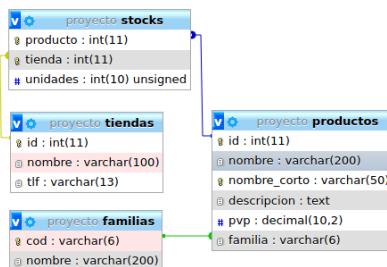
¿Podemos o no almacenar "emojis" en **MySQL**? La respuesta es si, pero no utilizando el "`utf-8`" que normalmente se usaba.

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.

A la hora de crear nuestra base de datos si queremos implementar esta característica, lo haremos de la siguiente manera:

```
mysql>create database mi_base_de_datos CHARACTER SET = utf8mb4 COLLATE = utf8m
```

Ejercicio resuelto



Captura de pantalla de phpMyAdmin (Elaboración propia)

Crearemos como usuario MySQL **root** y usando un archivo de procesamiento por lotes, 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**". Las tablas serán:

- 1.- **productos** (**id, nombre, nombre_corto, descripción, pvp, familia**)
- 2.- **tiendas** (**id, nombre, tlf**)
- 3.- **familias** (**cod, nombre**)
- 4.- **stocks** (**producto, tienda, unidades**)

Es decir el archivo ".sql", estará todo el código para crear la base de datos, las tablas, el usuario y los permisos del mismo para acceder a ella.

La solución deberá ser el archivo ".sql" y el comando, que tenemos que usar en el terminal, para "cargar" el archivo en la base de datos MySQL.

[Mostrar retroalimentación](#)

Descárgate una propuesta del archivo "proyecto.sql".
[Descargar](#) (pdf - 19869 B).

Utilizando la herramienta mysql ejecuta en el mismo directorio donde tengas "proyecto.sql" el comando siguiente:

```
sudo mysql -u root < proyecto.sql
```

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 quieras 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
```

Autoevaluación

Si quieres saber si en una tabla de una base de datos existe o no un registro, ¿qué herramienta en línea de comandos puedes usar?

- mysqladmin.
- mysql.

No es correcto, revisa las tareas que permite llevar a cabo esta herramienta.

Efectivamente. La herramienta `mysqladmin` la puedes utilizar para realizar tareas administrativas, pero no para ejecutar consultas sobre el contenido de las bases de datos.

Solución

1. Incorrecto
2. Opción correcta

2.2.2.- phpMyAdmin.

Al contrario que las dos herramientas anteriores, **phpMyAdmin** no se instala con el servidor MySQL. Debes instalarlo de forma individual, 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/>) 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**.

En ambos casos para poder entrar, debes indicar un nombre de usuario y contraseña válidos. Si realizaste el ejercicio anterior, se habrá creado en tu servidor un usuario "**gestor**" con contraseña "**secreto**" con todos los permisos para la base de datos " **proyecto**". Si utilizas ese usuario para entrar en la aplicación, ésta te permitirá gestionar la base de datos " **proyecto**".

Si quieras entrar en **phpMyAdmin** como usuario **root**, si has instalado **Xampp** en **Windows** es fácil, lo vimos en el apartado **2.1**. En Ubuntu la recomendación (para no tener que cambiar el método de autenticación del usuario **root**) es que crees un usuario, por ejemplo, **admin**, le des todos los permisos y entre como ese usuario. La forma de hacerlo sería la siguiente:

```
sudo mysql -u root
mysql>create user admin@'localhost' identified by 'secreto';
mysql>grant all on *.* to admin@'localhost' with grant option;
mysql>flush privileges;
```

Una vez creado ya podemos usar las credenciales **admin**, **secreto** para entrar en **phpMyAdmin** como un usuario de **MySQL** con todos los permisos (y por ello mucha precaución).

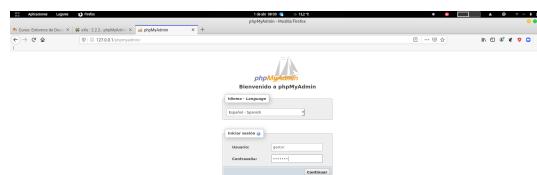
El interface de la aplicación se compone de un panel de navegación a la izquierda, donde se muestran las bases de datos, y un panel principal con un menú en la parte superior y una serie de acciones e información en la parte central. Si seleccionas la base de datos " **proyecto**", la información en 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 la misma.
- ✓ **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. En la siguiente presentación sobre **phpMyAdmin** tienes información sobre el manejo básico de la aplicación.

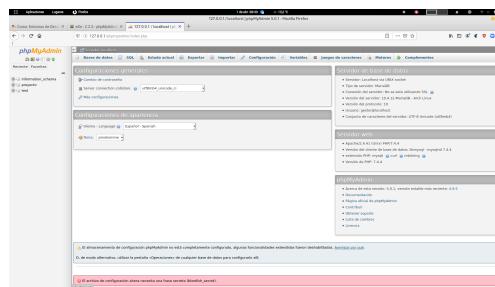
◀ Página de Login ▶



Captura de pantalla phpMyAdmin (Elaboración propia, uso educativo)

Podemos ver la página de login de phpMyAdmin, utilizaremos las credenciales **gestor** y **secreto** del ejercicio anterior.

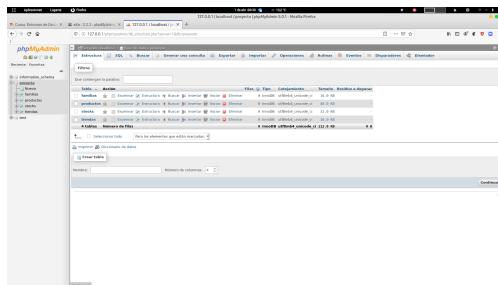
Página de inicio



Captura de pantalla de phpMyAdmin (Elaboración propia, uso educativo)

Una vez dentro observamos las bases de datos a las que el usuario gestor tiene acceso, nos centraremos en **proyecto**.

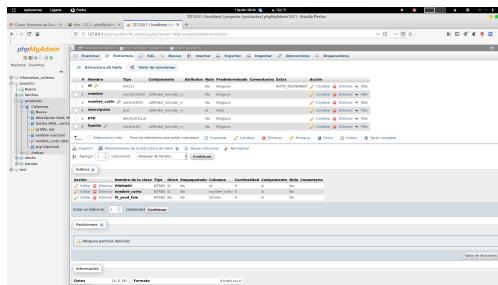
Tablas de la base de datos proyecto



Captura de pantalla de phpMyAdmin (Elaboración propia, uso educativo)

Si pinchamos en **proyecto** podremos ver las tablas que contiene y una serie de operaciones sobre ellas.

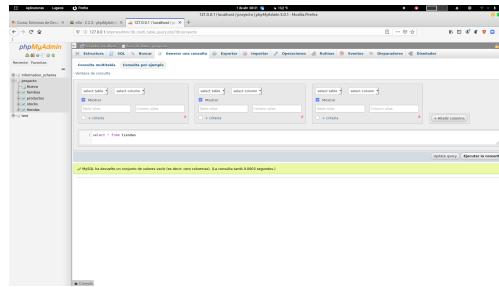
Estructura de las tablas



Captura de pantalla phpMyAdmin (Elaboración propia, uso educativo.)

Podemos fácilmente ver la estructura de una tabla y sus índices, podemos borrar, modificar y crear nuevos campos fácilmente.

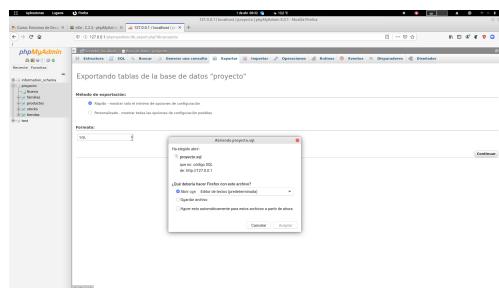
Generando Consultas



Captura de pantalla de phpMyAdmin (Elaboración propia, uso educativo)

En la pestaña "Generar una Consulta" podemos generar fácilmente una consulta. sólo tenemos que elegir la o las tablas, los campos y los distintos criterios

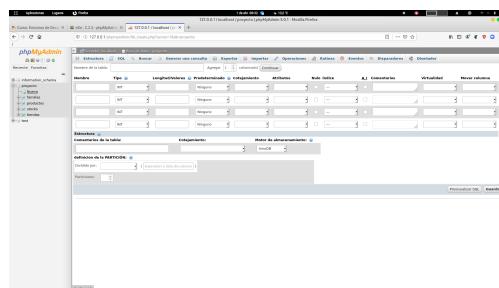
Exportando tablas



Captura de pantalla de phpMyAdmin (Elaboración propia, uso educativo)

En la pestaña "Exportar" podemos generar un archivo "SQL" con las instrucciones para crearnos todas las tablas y todos los datos que contengan.

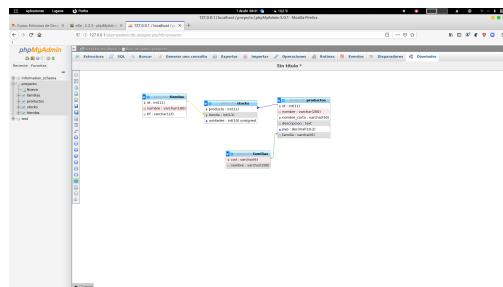
Creando nuevas Tablas



Captura de pantalla phpMyAdmin (Elaboración propia, uso educativo.)

Si le damos a "nueva" podremos fácilmente crearnos una tabla nueva en nuestra base de datos.

Diseñador



Captura de pantalla de phpMyAdmin (Elaboración propia, uso educativo)

De un vistazo podemos ver la estructura de nuestra base de datos, las tablas que contiene y como están relacionadas.

1 2 3 4 5 6 7 8

[Resumen textual alternativo](#)

Para saber más

En la página web de la aplicación tienes documentación sobre su configuración y utilización.

[Página web de la aplicación.](#)

Ejercicio resuelto

Utiliza **phpMyAdmin** para ejecutar las consultas del siguiente fichero, que rellenan con datos las tablas de la base de datos "proyecto". Esta información la utilizaremos en los próximos ejercicios.

[Fichero por lotes de SQL para insertar datos en la base de datos "proyecto" \(sql - 17,13 KB\).](#)

Mostrar retroalimentación

Selecciona en el panel de la izquierda la base de datos, y utilizando la pestaña Importar, elige el botón examinar para elegir el archivo "datosProyecto.sql" descargado. Las opciones que vienen por defecto suelen funcionar, pero échale un vistazo. Una vez elegido el archivo le damos a continuar y nos aseguramos en la información que nos da **phpMyAdmin** que no ha habido ningún error.

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

Caso práctico



Entre **María, Juan y Carlos**, han creado una pequeña base de datos con cuatro tablas y unas decenas de registros que usarán en las pruebas de la nueva aplicación web.

Juan, que ha tenido cierta experiencia programando aplicaciones en PHP, se da cuenta que el lenguaje ha evolucionado mucho en los últimos tiempos. Y uno de los aspectos que más ha evolucionado es precisamente el que concierne al acceso a bases de datos **MySQL**.

En las aplicaciones que había realizado hace ya algunos años, siempre había utilizado la misma extensión. Y ahora, por lo que ha estado viendo, existen otras maneras más eficientes o más genéricas de llevar a cabo esa tarea.

Para estar seguro, busca consejo en algunos programadores amigos y llega a una conclusión: tendrá que **escoger entre una extensión nativa, MySQLi, y PDO**. Revisa la documentación sobre ambas y realiza un pequeño estudio comparativo. Además, diseña unas pruebas con la ayuda de **María y Carlos** y poder tomar una decisión. Siempre es mejor asegurarse antes de empezar, aunque eso implique alargar algo más los plazos.

Como ya viste, 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.

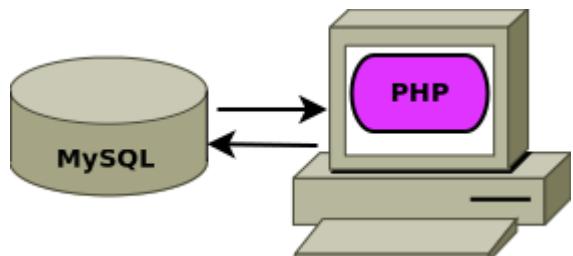


Imagen realizada con DIA (Elaboración propia)

- ✓ 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 driver 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 un 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:



Imagen Gimp (Elaboración propia con GIMP)

```
// utilizando constructores y métodos de la programación orientada a objetos

$conexion = new mysqli('servidor', 'usuario', 'contraseña', 'base_de_datos');
print $conexion->server_info;

// utilizando llamadas a funciones

$conexion = mysqli_connect('servidor', 'usuario', 'contraseña', 'base_de_datos');
print mysqli_get_server_info($conexion);
```

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.

Para saber más

Toda la información relativa a la instalación y utilización de la extensión, incluyendo las funciones y métodos propios de la extensión, se puede consultar en el manual de PHP.

[Manual de PHP.](#)

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

- ✓ Interface orientado a objetos.
- ✓ Soporte para transacciones.
- ✓ Soporte para consultas preparadas.
- ✓ Mejores opciones de depuración y seguridad.

Como ya viste en la primera unidad, 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.

Para saber más

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".

[Lista completa de las directivas.](#)

Autoevaluación

¿Qué interface o interfaces de programación admite la extensión MySQLi?

- Orientado a objetos únicamente.
- Dos interfaces de programación: procedural y orientado a objetos.

Incorrecto, admite interface orientado a objetos, sí, pero ¿solo ese?

Correcto, aunque el interface orientado a objetos es una mejora sobre la antigua extensión mysql, MySQLi mantiene también de forma paralela otro procedural.

Solución

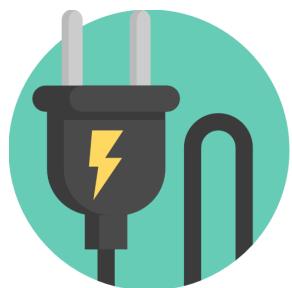
1. Incorrecto

2. Opción correcta

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:



[Enchufes Inteligentes \(CC BY-SA\)](#)

- ✓ El nombre o dirección **IP** del servidor **MySQL** al que te quieras 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:

```
// utilizando el constructor de la clase

$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:

```
// utilizando el método connect

$conProyecto = new mysqli();

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

Por el contrario, utilizando el interface procedimental de la extensión:

```
// utilizando llamadas a funciones

$conProyecto = mysqli_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 `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.

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');  
$error = $conProyecto->connect_errno;  
if ($error != null) {  
    echo "<p>Error $error conectando a la base de datos: $conProyecto->connect_error</p>";  
    die();  
}
```

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.

Operador de control de errores @.

Si una vez establecida la conexión, quieras 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, lógicamente el usuario con el que hemos iniciado la conexión debe tener permisos en la nueva.

```
// utilizando el método connect  
  
$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`).



[Ignacio javier igjav \(CC BY-SA\)](#)

```
@$conProyecto = new mysqli('localhost', 'gestor', 'secreto', 'proyecto'
$error = $conProyecto->connect_errno;
if ($error == null) {
    $resultado = $conProyecto->query('DELETE FROM stock WHERE unidades=
if ($resultado) {
    echo "<p>Se han borrado $conProyecto->affected_rows registros.<
}
$conProyecto->close(); //cerramos la conexion
}
```



En el caso de 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 stock', MYSQLI_USE_RESULT);
```



Debes conocer

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`.

[Método real_query](#).

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();
```

Autoevaluación

De las dos opciones que admite el método `query`, `MYSQLI_STORE_RESULT` y `MYSQLI_USE_RESULT`, ¿qué opción será recomendable utilizar para ejecutar una consulta que devuelva una enorme cantidad de datos?

- MYSQLI_STORE_RESULT.**
- MYSQLI_USE_RESULT.**

Incorrecto. ¿Qué sucede con los datos cuando se utiliza esta opción?

Efectivamente. Con esta opción se van obteniendo los datos del servidor a medida que se vayan necesitando. Si utilizaras la otra opción, los datos tendrían que transferirse todos juntos al ejecutar la consulta.

Solución

1. Incorrecto
2. Opción correcta

3.1.3.- Transacciones.

Como ya comentamos, 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`).



[Everaldo Coelho \(GNU/GPL\)](#)

```
$conProyecto->autocommit(false); // desabilitamos el modo transaccio
```

◀ ▶

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 stock WHERE unidades=0'); // Inicia una transacción  
  
$conProyecto->query('UPDATE stock SET unidades=3 WHERE producto="STYLUSSX515W"');  
...  
$conProyecto->commit(); // Confirma los cambios
```

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

Ejercicio resuelto

Según la información que figura en la tabla `stock` de la base de datos **proyecto**, la tienda 1 (CENTRAL) tiene 2 unidades del producto de código **3DSNG** y la tienda 3 (SUCURSAL2) ninguno. Suponiendo que los datos son esos (no hace falta que los compruebes en el código), utiliza una transacción para mover una unidad de ese producto de la tienda 1 a la tienda 3.

[Mostrar retroalimentación](#)

Deberás hacer una consulta de actualización (para poner unidades=1 en la tienda 1) y otra de inserción (pues no existe ningún registro previo para la tienda 3). Observa el código de la solución. Comprueba que se ejecuta bien solo la primera vez, pues en ejecuciones posteriores ya no es posible insertar la misma fila en la tabla.

[Código de la solución](#) (pdf - 30,44 KB)

Autoevaluación

En el modo de gestión de transacciones que se utiliza por defecto, ¿es posible revertir los cambios que se aplican al ejecutar una consulta de acción?

- No.
- Sí.

Correcto, el modo por defecto indica que se realice un "commit" automático por cada consulta, con lo cual es imposible revertir los cambios que se han realizado.

Incorrecto, revisa el funcionamiento del modo de gestión de transacciones que se aplica por defecto.

Solución

1. Opción correcta
2. Incorrecto

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 un interface 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.



[Everaldo Coelho \(GNU/GPL\)](#)

```
$resultado = $conProyecto->query('SELECT producto, unidades FROM stocks WHERE unidades<2');

$stock = $resultado->fetch_array(); // Obtenemos el primer registro

$producto = $stock['producto']; // 0 también $stock[0];

$unidades = $stock['unidades']; // 0 también $stock[1];

echo "<p>Producto $producto: $unidades unidades.</p>";
```

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.

Para 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.

```
$resultado = $conProyecto->query('SELECT producto, unidades FROM stocks WHERE unidades<2');

$stock = $resultado->fetch_object();
```

```
while ($stock != null) {  
  
    echo "<p>Producto $stock->producto: $stock->unidades unidades.</p>";  
  
    $stock = $resultado->fetch_object();  
  
}
```

Para saber más

En el manual de [PHP](#) tienes más información sobre los métodos y propiedades de la clase [mysqli_result](#).

[Clase mysqli_result](#).

Recomendación



Bootstrap
(Dominio público)

A la hora de empezar a elaborar proyectos más complejos, cuidar la presentación de las páginas es muy importante y el [CSS](#) nos puede llevar mucho tiempo, por eso se recomienda encarecidamente que uses frameworks de hojas de estilos ya diseñadas como [Bootstrap](#).

[Enlace a Documentación Bootstrap](#)

Ejercicio resuelto

Crea una página web en la que se muestren las unidades existentes de un determinado producto en cada una de las tiendas. Para seleccionar el producto concreto utiliza un cuadro de selección dentro de un formulario en esa misma página. Puedes usar como base los siguientes ficheros.

[Mostrar retroalimentación](#)

Revisa la solución propuesta y verifica que los resultados que obtuviste son correctos.

[Descargar Solución](#) (pdf - 41,86 KB)

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.



[Troy Hunt](#) (Dominio público)

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`:

```
$stmt = $conProyecto->stmt_init();
$stmt->prepare('INSERT INTO familias (cod, nombre) VALUES ("TABLET", "Tablet PC")');
$stmt->execute();
$stmt->close();
$conProyecto->close();
```

El problema que ya habrás 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.

Caracteres indicativos del tipo de los parámetros en una consulta preparada.

Carácter.	Tipo del parámetro.
i.	Número entero.
i.	Número real (doble precisión).
s.	Cadena de texto.
b.	Contenido en formato binario (BLOB).

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

```
$stmt = $conProyecto->stmt_init();
$stmt->prepare('INSERT INTO familias (cod, nombre) VALUES (?, ?)');
$cod_producto = "TABLET";
$nombre_producto = "Tablet PC";
$stmt->bind_param('ss', $cod_producto, $nombre_producto);
$stmt->execute();
$stmt->close();
$conProyecto->close();
```

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. Aprenderás a usar paso de parámetros por referencia en una unidad posterior.

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:

```
$stmt = $conProyecto->stmt_init();
$stmt->prepare('SELECT producto, unidades FROM stocks WHERE unidades<2');
$stmt->execute();
$stmt->bind_result($producto, $unidades);
while($stmt->fetch()) {
    echo "<p>Producto $producto: $unidades unidades.</p>";
}
$stmt->close();
$conProyecto->close();
```

Debes conocer

En el manual de [PHP](#) tienes más información sobre consultas preparadas y la clase `mysqli_stmt`.

[Consultas preparadas y la clase mysqli_stmt.](#)

Recomendación

Tanto `$stmt->prepare()` como `$stmt->execute()` devuelven un dato de tipo booleano, podemos usar esto para controlar errores, fíjate en el ejemplo siguiente:

```
$stmt=$conProyecto->stmt_init();
$cod=1;
$consulta="select nombre from productos where id=?";
if(!($stmt->prepare($consulta))){
    echo "Se ha producido un error: " . $conProyecto->error();
    die();
}
$stmt->bind_param('i', $cod);
if(!$stmt->execute()){
    //error
}
```

Ejercicio resuelto

A partir de la página web obtenida en el ejercicio anterior, añade la opción de modificar el número de unidades del producto en cada una de las tiendas. Utiliza una consulta preparada para la actualización de registros en la tabla stocks. No es necesario tener en cuenta las tareas de inserción (no existían unidades anteriormente) y borrado (si el número final de unidades es cero).

[Mostrar retroalimentación](#)

En esta ocasión es necesario crear un nuevo formulario en la página, en la sección donde se muestra el número de unidades por tienda. Revisa el código de la solución y pruebalo.

[Código de la solución al ejercicio propuesto. \(0.01 MB\)](#)

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 la misma. 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**.



Everaldo Coelho (GNU/GPL)

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 driver o 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.



[Enchufes Inteligentes \(CC BY-SA\)](#)

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

```
$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";
```

Para saber más

En el manual de PHP puedes consultar más información sobre los controladores existentes, los parámetros de las cadenas DSN y las opciones de conexión particulares de cada uno.

[Manual de 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. Por ejemplo, para obtener la versión del servidor puedes hacer:

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

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

```
$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);
```

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;
```

Para saber más

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.

[Manual `getAttribute`.](#)

[Manual `setAttribute`.](#)

Autoevaluación

Para establecer una conexión con MySQL utilizando PDO, ¿dónde se puede indicar el número de puerto TCP?

- En la cadena DSN que indica el origen de datos.
- En el array en que figuran las opciones específicas de conexión con el servidor.

Correcto. Se incluye como parte de la cadena DSN utilizando el parámetro "port".

Incorrecto, revisa los parámetros que se utilizan para establecer la conexión.

Solución

1. Opción correcta
2. Incorrecto

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.



[Ignacio javier igjav \(CC BY-SA\)](#)

```
$registros = $conProyecto->exec('DELETE FROM stocks WHERE unidades=0');
echo "<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_WARNING);
$resultado = $conProyecto->query("SELECT producto, unidades FROM stock");
```

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 $dwees->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`.

Ejercicio resuelto

De una forma similar al anterior ejercicio de transacciones, utiliza **PDO** para repartir entre las tiendas las tres unidades que figuran en unidades, del producto con código **PAPYRE62GB**, en la tienda de código **1**. Es decir primero con un `update` ponemos el número de unidades de unidades del producto de nombre corto **PAPYRE62GB** a **1** en la tienda de código **1** y luego hacemos un `insert` para insertar **dos** unidades de dicho producto en la tienda de código **2**.

[Mostrar retroalimentación](#)

En esta ocasión, para comprobar si los cambios se hacen correctamente en la base de datos y confirmamos la transacción, se revisa el número de registros afectados por la ejecución de las consultas. Revisa el código de la página y comprueba que la segunda vez que intentas ejecutarlo no actualizará los datos, tal y como sucedía en el ejercicio equivalente de la extensión **MySQLi**.

[Código con la solución propuesta.](#) (pdf - 37,29 KB)

Autoevaluación

Si programas tu aplicación correctamente utilizando "beginTransaction" antes de realizar un cambio, ¿siempre será posible revertirlo utilizando "rollback"?

- Sí.
- No.

Incorrecto. ¿Seguro? Piensa en qué sucede con los distintos motores de almacenamiento de **MySQL**, por ejemplo.

Correcto, no siempre; depende de si el motor de almacenamiento que estás utilizando soporta o no transacciones.

Solución

1. Incorrecto
2. Opción correcta

3.2.3.- Obtención y utilización de conjuntos de resultados.

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



[Everaldo Coelho \(GNU/GPL\)](#)

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

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.

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

- ✓ **PDO::FETCH_LAZY**. Devuelve tanto el objeto como el array con clave dual anterior.
- ✓ **PDO::FETCH_BOUND**. Devuelve true y asigna los valores del registro a variables, según se indique con el método `bindColumn`. Este método debe ser llamado una vez por cada columna, indicando en cada llamada el número de columna (empezando en 1) y la variable a asignar.

```
$conProyecto = new PDO("...");  
$resultado = $conProyecto->query("SELECT producto, unidades FROM stocks");  
 . . .
```

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

También podemos utilizar `fetchAll()` 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.

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

Ejercicio resuelto

Modifica la página web que muestra las unidades de un producto en las distintas tiendas, obtenida en un ejercicio anterior utilizando **MySQLi**, para que use PDO.

[Mostrar retroalimentación](#)

Comprueba en la solución propuesta los cambios realizados respecto a la solución del ejercicio equivalente que utilizaba **MySQLi**.

[Solución propuesta](#) (pdf - 41,73 KB).

Autoevaluación

¿Cuál es el comportamiento por defecto del método "fetch"?

- Devuelve un array con claves numéricas y asociativas.
- Devuelve un array asociativo.

Efectivamente. Es similar a utilizar el parámetro "PDO::FETCH_BOTH".

Incorrecto, revisa los distintos valores que se pueden utilizar para el parámetro opcional que admite.

Solución

1. Opción correcta
2. Incorrecto

3.2.4.- 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.



[Troy Hunt](#) (Dominio público)

```
$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` de la clase **PDOStatement**. Si utilizas signos de interrogación para marcar los parámetros, el procedimiento es equivalente al método `bindColumn` que acabamos de ver.

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

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

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

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`.

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

También existe otra forma de pasar valores a los parámetros. Hay un método `execute`, 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]);
]);
```

Para saber más

Puedes consultar la información sobre la utilización en [PDO](#) de consultas preparadas y la clase [PDOStatement](#) en el manual de [PHP](#).

[Consultas preparadas y la clase PDOStatement](#).

Ejercicio resuelto

Modifica el ejercicio sobre consultas preparadas que realizaste con la extensión **MySQLi**, el que modificaba el número de unidades de un producto en las distintas tiendas, para que utilice ahora la extensión [PDO](#).

[Mostrar retroalimentación](#)

Como puedes comprobar, para obtener la solución se puede aprovechar la mayoría del código existente en el ejercicio anterior.

[Solución al ejercicio propuesto](#) (pdf - 45,97 KB).

4.- Errores y manejo de excepciones.

Caso práctico



En sus primeros pasos en la programación en lenguaje PHP, **Carlos se ha encontrado frecuentemente con pequeños errores en el código**. En la mayoría de las ocasiones los errores estaban causados por fallos de programación. Pero en las recientes pruebas llevadas a cabo con bases de datos, se ha dado cuenta de que **en algunas ocasiones se pueden producir fallos ajenos al programa**.

Por ejemplo, puede obtener un error porque **no esté disponible el servidor de bases de datos** con el que se ha de conectar la aplicación, o porque **se acaba de borrar de la base de datos un registro al que estaba accediendo**. En éstos, y muchos otros, casos es necesario que la aplicación que desarrollen se comporte de forma sólida y coherente. Es necesario estudiar a fondo las **posibilidades que ofrece PHP para la gestión de errores**.

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 (**Visual Studio Code**), 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 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 guión, y la constante **E_ERROR** engloba errores fatales que provocan que se interrumpa forzosamente la ejecución.



[Everaldo Coelho \(GNU/GPL\)](#)

Debes conocer

La lista completa de constantes la puedes consultar en el manual de PHP, donde también se describe el tipo de errores que representa.

[Lista completa de constantes en 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.

Para saber más

Existen otros parámetros que podemos utilizar en `php.ini` para ajustar el comportamiento de PHP cuando se produce un error.

[Parámetros que podemos ajustar en `php.ini`.](#)

Desde código, puedes usar la función `error_reporting` con las constantes anteriores para establecer el nivel de notificación en un momento determinado. Por ejemplo, si en algún lugar de tu código figura una división en la que exista la posibilidad de que el divisor sea cero, cuando esto ocurra obtendrás un mensaje de error en el navegador. Para evitarlo, puedes desactivar la notificación de errores de nivel `E_WARNING` antes de la división y restaurarla a su valor normal a continuación:

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

Al usar la función `error_reporting` solo controlas qué tipo de errores va a notificar PHP. A veces puede ser suficiente, pero para obtener más control sobre el proceso existe también la posibilidad de reemplazar la gestión de los mismos por la que tú definas. Es decir, puedes programar una función para que sea la que se ejecuta cada vez que se produce un error. El nombre de esa función se indica utilizando `set_error_handler` y debe tener como mínimo dos parámetros obligatorios (el nivel del error y el mensaje descriptivo) y hasta otros tres opcionales con información adicional sobre el error (el nombre del fichero en que se produce, el número de línea, y un volcado del estado de las variables en ese momento).

```
set_error_handler("miGestorDeErrores");  
$resultado = $dividendo / $divisor;  
restore_error_handler();  
function miGestorDeErrores($nivel, $mensaje)  
{
```

```
switch($nivel) {  
    case E_WARNING:  
        echo "Error de tipo WARNING: $mensaje.<br />";  
        break;  
    default:  
        echo "Error de tipo no especificado: $mensaje.<br />";  
}  
}
```

La función `restore_error_handler` restaura el manejador de errores original de PHP (más concretamente, el que se estaba usando antes de la llamada a `set_error_handler`).

4.1.- 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**.
- ✓ Despues 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**.



[Everaldo Coelho \(GNU/GPL\)](#)

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

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

PHP ofrece una clase base **Exception** para utilizar como manejador (handler) de excepciones. Para lanzar una excepción no es necesario indicar ningún parámetro, aunque de forma opcional se puede pasar un mensaje de error (como en el ejemplo anterior) y también un código de error.

Entre los métodos que puedes usar con los objetos de la clase **Exception** están:

- ✓ **getMessage**. Devuelve el mensaje, en caso de que se haya puesto alguno.
- ✓ **getCode**. Devuelve el código de error si existe.

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).

Para saber más

Utilizando la clase **ErrorException** es posible traducir los errores a excepciones.

[Clase ErrorException](#).

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 quieras utilizar excepciones con la extensión **PDO**, debes configurar la conexión haciendo:

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

Por ejemplo, el siguiente código:

```
$host = "localhost";
$db = "proyecto";
$user = "gestor";
$pass = "1234";
$dsn = "mysql:host=$host;dbname=$db";
try {
    $conProyecto = new PDO($dsn, $user, $pass);
    $conProyecto->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $ex) {
    die("Error en la conexión, mensaje de error: " . $ex->getMessage());
}
```

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 conexión, mensaje de error: SQLSTATE[HY000] [1045] Access denied for user 'gestor'
```



Ejercicio resuelto

Agrega control de excepciones para controlar los posibles errores de conexión que se puedan producir en el último ejercicio del apartado **3.2.4**. Mostraremos los posibles errores abajo de la página.

[Mostrar retroalimentación](#)

Revisa en la solución propuesta los cambios realizados para utilizar excepciones en el establecimiento de la conexión.

[Solución al ejercicio propuesto](#) (pdf - 49,49 KB) .

Autoevaluación

¿Cuántos bloques "catch" se han de utilizar después de un bloque "try"?

- Uno.
- Uno o más.

Incorrecto, siempre debe haber al menos uno, pero ¿puede haber más?

Efectivamente, debe haber como mínimo uno, pero puede haber más. En este caso se ejecutará el bloque "catch" cuyo manejador coincida que se ha lanzado mediante "throw". Esto sólo tiene sentido si se utilizan distintos manejadores extendiendo la base que incluye PHP, "Exception".

Solución

1. Incorrecto
2. Opción correcta