

Trabajar con bases de datos en PHP.

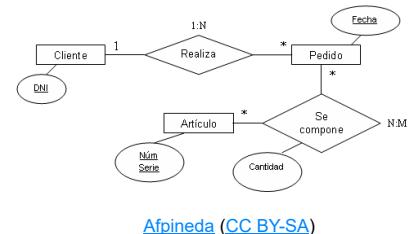
Caso práctico



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 efectos, a lo largo de



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

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.

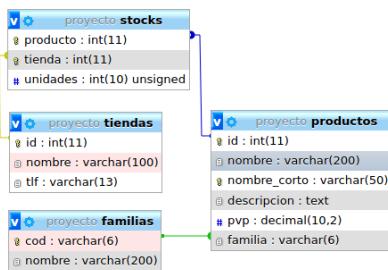
Incorrecto, revisa los distintos motores que se pueden utilizar con MySQL.

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.- Base de datos proyecto



Captura de pantalla de phpMyAdmin (Elaboración propia)

A continuación se presenta la base de datos que usaremos en las distintas secciones de esta unidad. Se trata de 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, descripcion, pvp, familia**)
- 2.- **tiendas** (**id, nombre, tlf**)
- 3.- **familias (cod, nombre)**
- 4.- **stocks (producto, tienda, unidades)**

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.

```

-- 1.- Creamos la Base de Datos
create database proyecto DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
-- Seleccionamos la base de datos "proyecto"
use proyecto;
-- 2.- Creamos las tablas
-- 2.1.1.- Tabla tiendas
create table if not exists tiendas(
    id int auto_increment primary key,
    nombre varchar(100) not null,
    tlf varchar(13) null
);

-- 
-- Volcado de datos para la tabla `tiendas`
--

INSERT INTO `tiendas` (`id`, `nombre`, `tlf`) VALUES
(1, 'CENTRAL', '600100100'),
(2, 'SUCURSAL1', '600100200'),
(3, 'SUCURSAL2', NULL);

-- 
-- 2.1.2 .- Tabla familias
create table if not exists familias(
    cod varchar(6) primary key,
    nombre varchar(200) not null
);

-- 
-- Volcado de datos para la tabla `familias`
--

```

```

INSERT INTO `familias` (`cod`, `nombre`) VALUES
('CAMARA', 'Cámaras digitales'),
('CONSOL', 'Consolas'),
('EBOOK', 'Libros electrónicos'),
('IMPRES', 'Impresoras'),
('MEMFLA', 'Memorias flash'),
('MP3', 'Reproductores MP3'),
('MULTIF', 'Equipos multifunción'),
('NETBOK', 'Netbooks'),
('ORDENA', 'Ordenadores'),
('PORTAT', 'Ordenadores portátiles'),
('ROUTER', 'Routers'),
('SAI', 'Sistemas de alimentación ininterrumpida'),
('SOFTWA', 'Software'),
('TV', 'Televisores'),
('VIDEOC', 'Videocámaras');

-----

-- 2.1.3.- Tabla productos
create table if not exists productos(
    id int auto_increment primary key,
    nombre varchar(200) not null,
    nombre_corto varchar(50) unique not null,
    descripcion text null,
    pvp decimal(10, 2) not null,
    familia varchar(6) not null,
    constraint fk_prod_fam foreign key(familia) references familias(cod) on update
    cascade on delete cascade
);

--

-- Volcado de datos para la tabla `productos`
--



INSERT INTO `productos` (`id`, `nombre`, `nombre_corto`, `descripcion`, `pvp`, `familia`) VALUES
(1, 'Nintendo 3DS negro', '3DSNG', 'Consola portátil de Nintendo que permitirá disfrutar de efectos de 3D sin gafas.', '150', 'CONSOL'),
(2, 'Acer AX3950 I5-650 4GB 1TB W7HP', 'ACERAX3950', 'Características:\r\n\r\n\r\nSistema Operativo: Windows 7 Home Premium. Procesador: Intel Core i5-650. Memoria RAM: 4GB. Almacenamiento: 1TB HDD. Pantalla: 15.6 pulgadas Full HD. Tarjeta gráfica: Intel HD Graphics 4600. Tamaño: 33.8 x 25.8 x 2.5 cm. Peso: 1.7 kg. Sistema operativo: Windows 7 Home Premium. Procesador: Intel Core i5-650. Memoria RAM: 4GB. Almacenamiento: 1TB HDD. Pantalla: 15.6 pulgadas Full HD. Tarjeta gráfica: Intel HD Graphics 4600. Tamaño: 33.8 x 25.8 x 2.5 cm. Peso: 1.7 kg.'),
(3, 'Archos Clipper MP3 2GB negro', 'ARCLPMP32GBN', 'Características:\r\n\r\n\r\nAlmacenamiento Interno: 2GB. Reproductores de medios: MP3, AAC, WMA, OGG, FLAC. Entradas: USB 2.0, microSDHC. Salidas: auriculares. Dimensiones: 10.5 x 2.5 x 1.8 cm. Peso: 100g. Sistema operativo: Archos Clipper OS. Procesador: Qualcomm MSM6225. Memoria RAM: 512MB. Almacenamiento interno: 2GB. Reproductores de medios: MP3, AAC, WMA, OGG, FLAC. Entradas: USB 2.0, microSDHC. Salidas: auriculares. Dimensiones: 10.5 x 2.5 x 1.8 cm. Peso: 100g.'),
(4, 'Sony Bravia 32IN FULLHD KDL-32BX400', 'BRAVIA2BX400', 'Características:\r\n\r\n\r\nFull HD: Verifica la calidad de imagen y sonido. Procesador: 1660 Mhz. Tamaño: 32 pulgadas. Resolución: 1920x1080. Tarjeta gráfica: BRAVIA Engine 2. Sistemas operativos: Smart TV. Dimensiones: 79.5 x 45.5 x 18.5 cm. Peso: 10.5 kg. Sistema operativo: Smart TV. Procesador: 1660 Mhz. Tamaño: 32 pulgadas. Resolución: 1920x1080. Tarjeta gráfica: BRAVIA Engine 2. Sistemas operativos: Smart TV. Dimensiones: 79.5 x 45.5 x 18.5 cm. Peso: 10.5 kg.'),
(5, 'Asus EEEPC 1005PXD N455 1 250 BL', 'EEEPC1005PXD', 'Características:\r\n\r\n\r\nProcesador: 1660 MHz. Memoria RAM: 1GB. Almacenamiento: 250GB. Pantalla: 10.1 pulgadas. Tarjeta gráfica: Intel GMA 3100. Sistemas operativos: Windows 7 Home Premium, Linux. Dimensiones: 25.5 x 18.5 x 1.8 cm. Peso: 1.2 kg. Sistema operativo: Windows 7 Home Premium, Linux. Procesador: 1660 MHz. Memoria RAM: 1GB. Almacenamiento: 250GB. Pantalla: 10.1 pulgadas. Tarjeta gráfica: Intel GMA 3100. Sistemas operativos: Windows 7 Home Premium, Linux. Dimensiones: 25.5 x 18.5 x 1.8 cm. Peso: 1.2 kg.'),
(6, 'HP Mini 110-3120 10.1LED N455 1GB 250GB W7S negro', 'HPMIN1103120', 'Características:\r\n\r\n\r\nTamaño: 25.5 x 18.5 x 1.8 cm. Peso: 1.2 kg. Sistema operativo: Windows 7 Home Premium. Procesador: Intel Celeron N455. Memoria RAM: 1GB. Almacenamiento: 250GB. Pantalla: 10.1 pulgadas LED. Tarjeta gráfica: Intel HD Graphics 3000. Dimensiones: 25.5 x 18.5 x 1.8 cm. Peso: 1.2 kg. Sistema operativo: Windows 7 Home Premium. Procesador: Intel Celeron N455. Memoria RAM: 1GB. Almacenamiento: 250GB. Pantalla: 10.1 pulgadas LED. Tarjeta gráfica: Intel HD Graphics 3000. Dimensiones: 25.5 x 18.5 x 1.8 cm. Peso: 1.2 kg.'),
(7, 'Canon Ixus 115HS azul', 'IXUS115HSAZ', 'Características:\r\n\r\n\r\nHS System (12,1 MP) \r\n\r\nZoom óptico: 3x. Pantalla: 3 pulgadas. Tarjeta SDHC. Sistemas operativos: Windows 7 Home Premium, Mac OS X. Dimensiones: 11.5 x 7.5 x 3.5 cm. Peso: 180g. Sistema operativo: Windows 7 Home Premium, Mac OS X. Tarjeta SDHC. Dimensiones: 11.5 x 7.5 x 3.5 cm. Peso: 180g. Sistema operativo: Windows 7 Home Premium, Mac OS X. Tarjeta SDHC. Dimensiones: 11.5 x 7.5 x 3.5 cm. Peso: 180g.'),
(8, 'Kingston DataTraveler 16GB DT101G2 USB2.0 negro', 'KSTD101G2', 'Características:\r\n\r\n\r\nCapacidad: 16GB. Interfaz: USB 2.0. Dimensiones: 7.5 x 1.5 x 1.5 cm. Peso: 10g. Sistema operativo: No requiere sistema operativo. Tarjeta SDHC. Dimensiones: 7.5 x 1.5 x 1.5 cm. Peso: 10g. Sistema operativo: No requiere sistema operativo. Tarjeta SDHC. Dimensiones: 7.5 x 1.5 x 1.5 cm. Peso: 10g.'),
(9, 'Kingston DataTraveler G3 32GB rojo', 'KSTDGTG332GBR', 'Características:\r\n\r\n\r\nTipo de producto: memoria externa. Capacidad: 32GB. Interfaz: USB 3.0. Dimensiones: 7.5 x 1.5 x 1.5 cm. Peso: 10g. Sistema operativo: No requiere sistema operativo. Tarjeta SDHC. Dimensiones: 7.5 x 1.5 x 1.5 cm. Peso: 10g. Sistema operativo: No requiere sistema operativo. Tarjeta SDHC. Dimensiones: 7.5 x 1.5 x 1.5 cm. Peso: 10g.'),
(10, 'Kingston MicroSDHC 8GB', 'KSTMSDH8GB', 'Kingston tarjeta de memoria flash 8 GB microSDHC', '8', 'CONSOL'),
(11, 'Canon Legria FS306 plata', 'LEGRIAFS306', 'Características:\r\n\r\n\r\nGrabación en tarjeta de memoria: Sí. Dimensiones: 11.5 x 7.5 x 3.5 cm. Peso: 180g. Sistema operativo: No requiere sistema operativo. Tarjeta SDHC. Dimensiones: 11.5 x 7.5 x 3.5 cm. Peso: 180g. Sistema operativo: No requiere sistema operativo. Tarjeta SDHC. Dimensiones: 11.5 x 7.5 x 3.5 cm. Peso: 180g.'),
(12, 'LG TDT HD 23 M237WDP-PC FULL HD', 'LGM237WDP', 'Características:\r\n\r\n\r\nGeneral\r\n\r\nTamaño: 23 pulgadas. Resolución: 1920x1080. Tarjeta gráfica: LG TDT HD 23 M237WDP-PC. Dimensiones: 54.5 x 34.5 x 10.5 cm. Peso: 10.5 kg. Sistema operativo: No requiere sistema operativo. Tarjeta SDHC. Dimensiones: 54.5 x 34.5 x 10.5 cm. Peso: 10.5 kg. Sistema operativo: No requiere sistema operativo. Tarjeta SDHC. Dimensiones: 54.5 x 34.5 x 10.5 cm. Peso: 10.5 kg.'),
(13, 'HP Laserjet Pro Wifi P1102W', 'LJPROP1102W', 'Impresora laserjet P1102W es facil de instalar', '10', 'IMPRES'),
(14, 'Pentax Optio LS1100', 'OPTIOLS1100', 'La LS1100 con funda de transporte y tarjeta de memoria', '15', 'CAMARA'),
(15, 'Lector ebooks Papyre6 con SD2GB + 500 ebooks', 'PAPYRE62GB', 'Marca Papyre \r\n\r\nModelo Pap', '10', 'EBOOK'),
(16, 'Packard Bell I8103 23 I3-550 4G 640GB NVIDIAAG210', 'PBELLI810323', 'Características:\r\n\r\n\r\n', '10', 'CONSOL'),
(17, 'Canon Pixma IP4850', 'PIXMAIP4850', 'Características:\r\n\r\n\r\nTipo: chorro de tinta cartucho', '10', 'IMPRES'),
(18, 'Canon Pixma MP252', 'PIXMAMP252', 'Características:\r\n\r\n\r\nFunciones: Impresora, Escáner', '10', 'IMPRES'),
(19, 'PS3 con disco duro de 320GB', 'PS3320GB', 'Este Pack Incluye:\r\n\r\n- La consola PlayStation 3', '10', 'CONSOL'),
(20, 'Canon Powershot A3100 plata', 'PWSHTA3100PT', 'La cámara PowerShot A3100 IS, inteligente', '10', 'CAMARA'),
(21, 'Samsung CLX3175', 'SMSGCLX3175', 'Características:\r\n\r\n\r\nFunción: Impresión color, copia', '10', 'IMPRES'),
(22, 'Samsung N150 10.1LED N450 1GB 250GB BAT6 BT W7 R', 'SMSN150101LD', 'Características:\r\n\r\n\r\n', '10', 'CONSOL'),
(23, 'Samsung SMX-C200PB EDC ZOOM 10X', 'SMSSMXC200PB', 'Características:\r\n\r\n\r\nSensor de Imagen', '10', 'EBOOK');

```

```
(24, 'Epson Stylus SX515W', 'STYLUSSX515W', 'Características:\r\n\r\nResolución máxima 5760 x 1440', 'Toshiba SD16GB Class10 Jewel Case', 'TSSD16GBC10J', 'Características:\r\n\r\nDensidad: 160 GB', 'Creative Zen MP4 8GB Style 300', 'ZENMP48GB300', 'Características:\r\n\r\n8 GB de capacidad'),  
-- -----  
-- 2.1.4 Tabla stocks  
create table if not exists stocks(  
    producto int,  
    tienda int,  
    unidades int unsigned not null,  
    constraint pk_stock primary key(producto, tienda),  
    constraint fk_stock_prod foreign key(producto) references productos(id) on update cascade on delete cascade,  
    constraint fk_stock_tienda foreign key(tienda) references tiendas(id) on update cascade on delete cascade  
);  
  
--  
-- Volcado de datos para la tabla `stocks`  
--  
  
INSERT INTO `stocks` (`producto`, `tienda`, `unidades`) VALUES  
(1, 1, 2),  
(1, 2, 1),  
(2, 1, 1),  
(3, 2, 1),  
(3, 3, 2),  
(4, 3, 1),  
(5, 1, 2),  
(5, 2, 1),  
(6, 2, 1),  
(6, 3, 2),  
(7, 2, 2),  
(8, 3, 1),  
(9, 1, 1),  
(9, 2, 2),  
(10, 2, 2),  
(10, 3, 2),  
(11, 2, 1),  
(12, 1, 1),  
(13, 2, 2),  
(14, 1, 3),  
(14, 2, 1),  
(15, 1, 2),  
(15, 3, 1),  
(16, 2, 1),  
(17, 2, 1),  
(17, 3, 2),  
(18, 2, 1),  
(19, 1, 1),  
(20, 2, 2),  
(20, 3, 2),  
(21, 2, 1),  
(22, 3, 1),  
(23, 2, 1),  
(24, 1, 1),  
(25, 3, 2),  
(26, 1, 3),
```

```
(26, 2, 2),  
(26, 3, 2);
```

```
-- 3.- Creamos un usuario  
create user gestor@'localhost' identified by "secreto";  
-- 4.- Le damos permiso en la base de datos "proyecto"  
grant all on proyecto.* to gestor@'localhost';
```

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.
- ✓ **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".

Por ejemplo, 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 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`".

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



[Moini](#) (Dominio público)

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 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`).

Para crear la base de datos proyecto en MySQL podemos utilizar el fichero `proyecto.sql` que presentamos en la sección anterior.

```
mysql -u root -p < proyecto.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

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.

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 = utf8mb4
```



2.2.2.- phpMyAdmin.

Como vimos en secciones anteriores XAMPP incluye la herramienta **phpMyAdmin** con lo que no hace falta ninguna instalación adicional. Para ejecutarlo pinchamos 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/>) 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**.

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.

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.



[Michael Keck \(CC BY-SA\)](#)

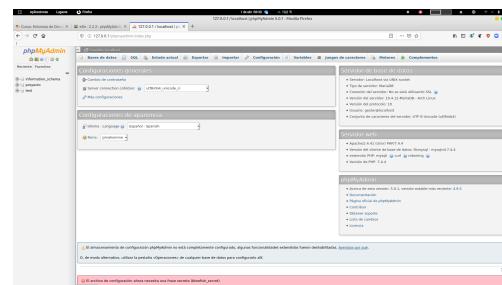
Página de Login



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.

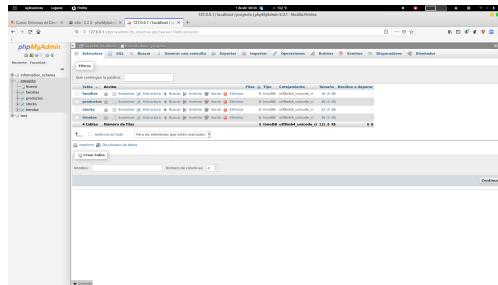
Página de inicio



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**.

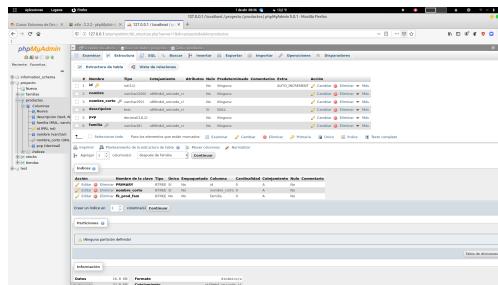
Tablas de la base de datos proyecto



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.

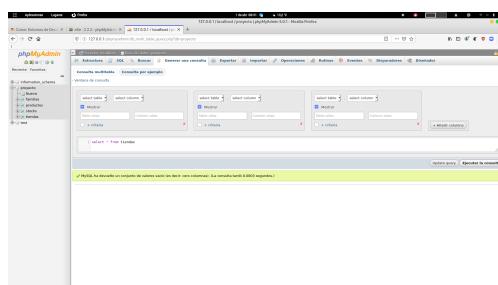
Estructura de las tablas



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

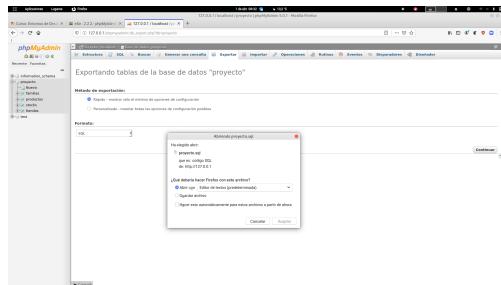


Captura de pantalla de phpMyAdmin (Elaboración propia, uso educativo)

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

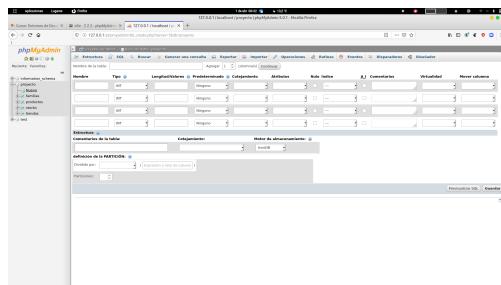
Exportando tablas



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.

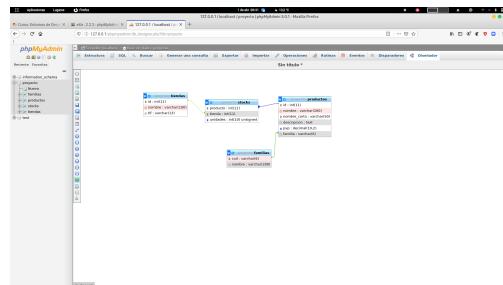
Creando nuevas Tablas



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.

[Resumen textual alternativo](#)

Para saber más

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

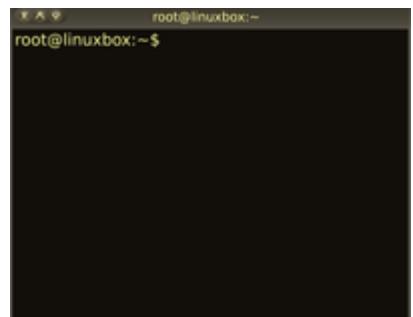
[Página web de la aplicación](#)

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.

[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".

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 disponible usando "`mysqladmin --help`":

```
mysqladmin --help
sudo mysqladmin status
sudo mysqladmin version
```

Fíjate que los comandos anteriores no necesitan especificar usuario por que 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

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.
- ✓ Ejecutar procedimientos almacenados.

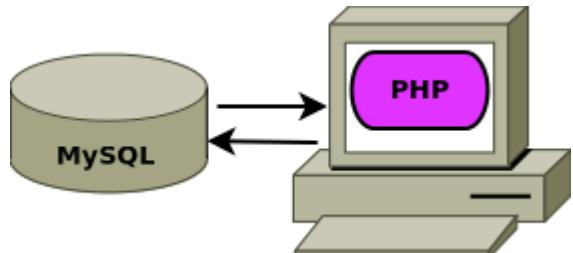


Imagen realizada con DIA (Elaboración propia)

- ✓ 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
// servidor podría ser localhost, usuario podría ser gestor, contraseña podría ser secreto y base de datos
$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');
```

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**

```
$controlador = new mysqli_driver();
$controlador->report_mode = MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT;
```

Si quisieramos 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.

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

```
<?php
$controlador = new mysqli_driver();
$controlador->report_mode = MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT;
$conProyecto = new mysqli('localhost', 'gestor', 'secreto', 'proyecto');
try {
    $conProyecto->query("DELETE FROM productos WHERE
        nombre_corto='TSSD16GBC10J'");
}
catch (mysqli_sql_exception $e) {
    echo "Error en la consulta: " . $e->getMessage();
}
?>
<p>"Se han borrado <?= $conProyecto->affected_rows ?> registros."</p>
<?php
$conProyecto->close();
```



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 stocks', 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); // deshabilitamos el modo transaccion
```

◀ ▶

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.

Ejercicio resuelto

Transacciones

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.

```
<?php
$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());
}

//Definimos una variable para comprobar que no tenemos errores
$commit = true;
//Iniciamos la transacción
$conProyecto->autocommit(false);
$update = "update stocks set unidades=1 where producto=(select id from producto
if (!$conProyecto->query($update)) {
    $commit = false;
}
//fíjate en este insert, el select devolverá el productos.id del producto
//estamos haciendo un insert into stocks(producto, tienda, unidades) lo vamos a
$insert = "insert into stocks(producto, tienda, unidades) select id, 3, 1
if (!$conProyecto->query($insert)) {
    $commit = false;
}
//Si todo fue bien hacemos el commit si no el rollback
if ($commit) {
    $conProyecto->commit();
} else {
    $conProyecto->rollback();
}
$conProyecto->close();
?>
<!DOCTYPE html>
<html lang="es">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport"
            content="width=device-width, user-scalable=no, initial-scale=1.0,
            maximum-scale=1.0, minimum-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <!-- enlaces para usar Bootstrap -->
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" integrity="sha384-T3cCqPf4dWnPQyJLzOxHkZDnYJNtDmDkG0vZJ3JLWZJGgZDSCmG1MXxSR1C" rel="stylesheet">
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js" integrity="sha384-KnH9C9i0D07Uz0ZoZw01OZcDZL0d0J1+M6J3PdQa0DZVXZ6QfJZ7vZu0D1" crossorigin="anonymous">
        <title>Transacciones MySQLi </title>
    </head>
```

```
<body class="bg-primary">
    <h3 class="text-center mt-2 font-weight-bold">Transacción con MySQL</h3>
    <div class="container mt-3">
        <?php if ($commit): ?>
            <p class='text-primary font-weight-bold'>Los cambios se han aplicado con éxito!</p>
        <?php else: ?>
            <p class='text-danger font-weight-bold'>No se han podido realizar los cambios</p>
        <?php endif ?>
    </div>
</body>
</html>
```

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:



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

- ✓ `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.

```
<?php
$controlador = new mysqli_driver();
$controlador->report_mode = MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT;
try {
    $conProyecto = new mysqli('localhost', 'gestor', 'secreto', 'proyecto');
    $resultado = $conProyecto->query('SELECT producto, unidades FROM stocks
        WHERE unidades<2');
} catch (mysqli_sql_exception $e) {
    die ("Error en la conexión o la consulta: " . $e->getMessage());
}
$stock = $resultado->fetch_array(); // Obtenemos el primer registro
$producto = $stock['producto']; // 0 también $stock[0];
$unidades = $stock['unidades']; // 0 también $stock[1];
?>
<p>Producto <?= "$producto : $unidades unidades." ?></p>
<?php
$conProyecto->close(); //cerramos la conexion
```

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.

```
<?php
$controlador = new mysqli_driver();
$controlador->report_mode = MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT;
try {
    $conProyecto = new mysqli('localhost', 'gestor', 'secreto', 'proyecto');
    $resultado = $conProyecto->query('SELECT producto, unidades FROM stocks
        WHERE unidades<2');
} catch (mysqli_sql_exception $e) {
    die ("Error en la conexión o la consulta: " . $e->getMessage());
}
?>
<?php while ($stock = $resultado->fetch_object()): ?>
    <p><?= "Producto {$stock->producto}: {$stock->unidades} unidades." ?></p>
<?php endwhile ?>
<?php
$conProyecto->close(); //cerramos la conexion
```

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 Obtención y Utilización de Resultados 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

```
<?php
$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());
}
if (filter_has_var(INPUT_POST, 'enviar')) {
    $petStock = true;
    $codProd = filter_input(INPUT_POST, 'producto', FILTER_UNSAFE_RAW);
    try {
        $consultaProducto = "select nombre , nombre_corto from productos w";
        $resultadoConsultaProducto = $conProyecto->query($consultaProducto);
        $datosProducto = $resultadoConsultaProducto->fetch_assoc();
        $resultadoConsultaProducto->free();
    } catch (mysqli_sql_exception $e) {
        $errorConsultaProducto = true;
    }
    try {
        $consultaStock = "select unidades, tiendas.nombre as tienda_nombre";
        $resultadoStockProducto = $conProyecto->query($consultaStock);
    } catch (mysqli_sql_exception $e) {
        $errorConsultaStock = true;
    }
} else {
    try {
        $consultaProductos = "select id, nombre, nombre_corto from productos";
        $resultadoProductos = $conProyecto->query($consultaProductos);
        $datosProductos = $resultadoProductos->fetch_all(MYSQLI_ASSOC);
        $resultadoProductos->free();
    } catch (mysqli_sql_exception $e) {
        $errorConsultaProductos = true;
    }
}
?>
<!DOCTYPE html>
<html lang="es">
```

```

<head>
    <meta charset="UTF-8">
    <meta name="viewport"
        content="width=device-width, user-scalable=no, initial-scale:1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <!-- enlaces para usar Bootstrap -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/t
        integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1C
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/t
        integrity="sha384-C6RzsynM9kWDrMNeT87bh950GNyZPhcTNXj1NW7RuBCsyN/o
    <title>Conjuntos de resultados en MySQLi </title>
</head>
<body class="bg-info">
    <div class="container mt-3">
        <h3 class="text-center mt-2 font-weight-bold">Unidades de un producto</h3>
        <?php if (isset($petStock)): ?>
            <h4 class="mt-3 mb-3 text-center">Unidades del Producto: <i></i><?php echo $petStock['nombre'] ?></h4>
            <a href=<?= $_SERVER['PHP_SELF'] ?>" class="btn btn-success">Actualizar Stock</a>
            <?php if (isset($errorConsultaStock)): ?>
                <p class="font-weight-bold text-success mt-3">Problema resuelto</p>
            <?php else: ?>
                <table class="table table-striped table-dark">
                    <thead>
                        <tr class="text-center font-weight-bold">
                            <th>Nombre Tienda</th>
                            <th>Stock</th>
                        </tr>
                    </thead>
                    <tbody>
                        <?php while ($stockProducto = $resultadoStockProducto->fetch_assoc()): ?>
                            <tr>
                                <td><?= $stockProducto['tienda_nombre'] ?></td>
                                <td class="textcenter"><?= $stockProducto['stock'] ?></td>
                            </tr>
                        <?php endwhile ?>
                    </tbody>
                </table>
            <?php
                $resultadoStockProducto->close();
            ?>
        <?php endif ?>
        <?php else: ?>
            <form name="formulario" action=<?= $_SERVER['PHP_SELF'] ?>>
                <div class="row">
                    <label for="p" class="font-weight-bold">Elige un producto</label>
                    <?php if (!isset($errorConsultaProductos)): ?>
                        <select class="form-control" id="p" name="producto_id">
                            <?php foreach ($datosProductos as $producto): ?>
                                <option value=<?= $producto['id'] ?>>
                            <?php endforeach ?>
                        </select>
                    <?php else: ?>
                        <h4><?= "No se han podido recuperar productos" ?></h4>
                    <?php endif ?>
                </div>
                <div class="mt-2">
                    <input type="submit" class="btn btn-warning me-3" value="Actualizar Stock" />
                </div>
            </form>
        <?php
            $resultadoStockProducto->close();
        ?>
    </div>
</body>

```

```
        name="enviar">
    </div>
</form>
</div>
<?php endif ?>
</body>
</html>
<?php
$conProyecto->close();
?>
```

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.



Troy Hunt (Dominio público)

```
$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();

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 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.
d	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:

```
<?php
$controlador = new mysqli_driver();
$controlador->report_mode = MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT;
$conProyecto = new mysqli('localhost', 'gestor', 'secreto', 'proyecto');
$stmt = $conProyecto->stmt_init();
try {
$stmt->prepare('SELECT producto, unidades FROM stocks WHERE unidades<2');
$stmt->execute();
} catch (mysqli_sql_exception $e) {
die("Error en la preparación o ejecución de la consulta: " . $e->getMessage())
}
$stmt->bind_result($producto, $unidades);
?>
<?php while ($stmt->fetch()): ?>
<p><?= "Producto $producto: $unidades unidades." ?></p>
<?php endwhile ?>
<?php
$stmt->close();
$conProyecto->close(); //cerramos la conexión
```

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`.](#)

Ejercicio Consultas Preparadas 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

```
<?php
$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());
}
if (filter_has_var(INPUT_POST, 'enviar')) {
    $petStock = true;
    $codProd = filter_input(INPUT_POST, 'producto', FILTER_UNSAFE_RAW);
    try { // sentencia preparada. Acceso al resultado directamente a través
        $consultaProducto = "select nombre , nombre_corto from productos w";
        $stmtConsultaProducto = $conProyecto->prepare($consultaProducto);
        $stmtConsultaProducto->bind_param('i', $codProd);
        $stmtConsultaProducto->execute();
        $stmtConsultaProducto->bind_result($nombre, $nombreCorto);
        $stmtConsultaProducto->fetch(); //esta consulta solo devuelve una fila
        $stmtConsultaProducto->free_result();
        $stmtConsultaProducto->close();
    } catch (mysqli_sql_exception $e) {
        $errorConsultaProducto = true;
    }
    try { // Sentencia preparada. Acceso al conjunto de resultados a través
        $consultaStock = "select unidades, tienda as tienda_id, producto as";
        $stmtConsultaStock = $conProyecto->prepare($consultaStock);
        $stmtConsultaStock->bind_param('i', $codProd);
        $stmtConsultaStock->execute();
        $resultadoStockProducto = $stmtConsultaStock->get_result();
    } catch (mysqli_sql_exception $e) {
        $errorConsultaStock = true;
    }
} elseif (filter_has_var(INPUT_POST, 'enviar_stock')) {
    $petActualStock = true;
    $codTienda = filter_input(INPUT_POST, 'codigo_tienda', FILTER_UNSAFE_RAW);
    $codProducto = filter_input(INPUT_POST, 'codigo_producto', FILTER_UNSAFE_RAW);
    $unidades = filter_input(INPUT_POST, 'stock', FILTER_UNSAFE_RAW);
    try { // Sentencia preparada.
        $consultaUpdateStock = "update stocks set unidades=? where producto=?";
        $stmtConsultaUpdateStock = $conProyecto->prepare($consultaUpdateStock);
        $stmtConsultaUpdateStock->bind_param('ii', $unidades, $codProducto);
        $stmtConsultaUpdateStock->execute();
        $stmtConsultaUpdateStock->close();
    } catch (mysqli_sql_exception $e) {
        $errorActualizacionStock = true;
    }
} else {
    try { // Setencia preparada. Acceso a los resultados directamente a través
        $consultaProductos = "select id, nombre from productos order by nom";
    }
}
```

```
$stmtConsultaProductos = $conProyecto->prepare($consultaProductos);
$stmtConsultaProductos->execute();
$stmtConsultaProductos->store_result();
$stmtConsultaProductos->bind_result($producto['id'], $producto['nombre']);
$numProductos = $stmtConsultaProductos->num_rows();
} catch (mysqli_sql_exception $e) {
    $errorConsultaProductos = true;
}
}

?>
<!DOCTYPE html>
<html lang="es">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport"
            content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <!-- enlaces para usar Bootstrap -->
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/t
            integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1C
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/t
            integrity="sha384-C6RzsM9kWDrMNeT87bh950GNyZPhcTNXj1NW7RuBCsyN/o6
            <title>Sentencias preparadas en MySQLi</title>
    </head>
    <body class="bg-info">
        <h3 class="text-center mt-2 font-weight-bold">Actualización de stock</h3>
        <div class="container mt-3">
            <?php if (isset($petStock)): ?>
                <h4 class="mt-3 mb-3 text-center">Unidades del Producto: <i><?php echo $petStock['stock'] ?></i></h4>
                <a href=<?= $_SERVER['PHP_SELF'] ?>" class="btn btn-success">Actualizar</a>
                <?php if (isset($errorConsultaStock)): ?>
                    <p class="font-weight-bold text-success mt-3">Problema resuelto</p>
                <?php else: ?>
                    <table class="table table-striped table-dark">
                        <thead>
                            <tr class="text-center font-weight-bold">
                                <th>Nombre Tienda</th>
                                <th>Stock</th>
                            </tr>
                        </thead>
                        <tbody>
                            <?php while ($stock = $resultadoStockProducto): ?>
                                <tr>
                                    <td><?= $stock['tienda_nombre'] ?></td>
                                    <td class="textcenter">
                                        <form name='formulario_actualiza_s
                                            <div class="input-group">
                                                <input type="number" class="form-control" value="<?= $stock['stock'] ?>" name="stock">
                                                <input type="hidden" name="id" value="<?= $stock['id'] ?>">
                                                <input type="hidden" name="tienda_nombre" value="<?= $stock['tienda_nombre'] ?>">
                                                <input type="submit" class="btn btn-primary" value="Actualizar">
                                            </div>
                                        </form>
                                    </td>
                                </tr>
                            <?php endwhile ?>
                        </tbody>
                    </table>
                <?php endif: ?>
            </div>
        </div>
    </body>
</html>
```

```
$stmtConsultaStock->free_result();
$stmtConsultaStock->close();
?>
</tbody>
</table>
<?php endif ?>
<?php elseif (isset($petActualStock)): ?>
<p class="font-weight-bold text-success mt-3">
    <?= isset($errorActualizacionStock) ? "Problemas con la actualización de stock." : "Stock actualizado con éxito." ?>
    <a href="<?= $_SERVER['PHP_SELF'] ?>" class="btn btn-success">Actualizar Stock</a>
<?php else: ?>
    <form name="formulario" action="<?= $_SERVER['PHP_SELF'] ?: '#>" method="post">
        <div class="row">
            <label for="p" class="font-weight-bold">Elige un producto para consultar su stock:</label>
            <?php if (!isset($errorConsultaProductos)): ?>
                <select class="form-control" id="p" name="producto_id">
                    <?php while ($stmtConsultaProductos->fetch()): ?>
                        <option value='<?= $producto['id'] ?>'><?= $producto['nombre'] ?></option>
                    <?php endwhile ?>
                <?php
                $stmtConsultaProductos->free_result();
                $stmtConsultaProductos->close();
            ?>
            </select>
            <?php else: ?>
                <h4><?= "No se han podido recuperar productos" ?>
            <?php endif ?>
        </div>
        <div class="mt-2">
            <input type="submit" class="btn btn-warning me-3" value="Consultar Stock" <?= isset($errorConsultaProductos) ? 'disabled' : '' ?> <?php if (isset($errorConsultaProductos)): ?> <small><?= $errorConsultaProductos ?></small></input>
        </div>
    </form>
</div>
<?php endif ?>
</body>
</html>
<?php
$conProyecto->close();
?>
```

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**.

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.



Everaldo Coelho (GNU/GPL)

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);
```

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());
}
```

```
//se recomienda guardar los datos(host, user...) en variables porque si estos cambian  
//solo tenemos que actualizar el valor de estas variables
```

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

```
<?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");
```

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.- 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**.

Ejercicio resuelto

Transacciones

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 PAPYRE62GB 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

La solución propuesta es:

```

<?php
$host = "localhost";
$db = "proyecto";
$user = "gestor";
$pass = "secreto";
$dsn = "mysql:host=$host;dbname=$db;charset=utf8mb4";
try {
    $conProyecto = new PDO($dsn, $user, $pass);
    $conProyecto->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    die("Error con la conexión a la base de datos" . $e->getMessage());
}
$commit = true;
// Iniciamos la transacción
$conProyecto->beginTransaction();
$update = "update stocks set unidades=1 where producto=(select id from productos
if (!$conProyecto->exec($update))
    $commit = false;
$insert = "insert into stocks select id, 2, 1 from productos where nombre_categoría='Libros'
if (!$conProyecto->exec($insert))
    $commit = false;
// Si fue bien, confirmamos los cambios
// y en caso contrario los deshacemos
if ($commit) {
    $conProyecto->commit();
} else {
    $conProyecto->rollBack();
}
//Cerramos la conexión.
$conProyecto = null;
?>
<!DOCTYPE html>
<html lang="es">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport"
            content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <!-- enlaces para usar Bootstrap -->
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatjzcDSCmG1MXxSR1C" rel="stylesheet">
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js" integrity="sha384-C6RzsynM9kWDrMNeT87bh950GNyZPhcTNXj1NW7RuBCsyN/oA3p3egisV9n�q" rel="script">
            <title>Transacciones en PDO </title>
    </head>
    <body class="bg-info">
        <h3 class="text-center mt-2 font-weight-bold">Ejercicio Transacciones</h3>
        <div class="container mt-3">
            <?php if ($commit): ?>
                <p class='text-primary font-weight-bold'>Los cambios se realizaron con éxito.</p>
            <?php else: ?>
                <p class='text-danger font-weight-bold'>No se han podido realizar cambios.</p>
            <?php endif ?>
        </div>

```

```
</body>  
</html>
```

3.2.4.- 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)

```
<?php
$host = "localhost";
$db = " proyecto ";
$user = " gestor ";
$pass = " secreto ";
$dsn = "mysql:host=$host;dbname=$db;charset=utf8mb4";
try {
    $conProyecto = new PDO($dsn, $user, $pass);
    $conProyecto->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $resultado = $conProyecto->query("SELECT producto, unidades FROM stocks");
    $conProyecto = null; //cerramos la conexion
} catch (PDOException $e) {
    die("Error en la conexión o la consulta: " . $e->getMessage());
}
?>
<?php while ($registro = $resultado->fetch()): ?>
    <p><?= "Producto {$registro['producto']}: {$registro['unidades']} uni
<?php endwhile ?>
```



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.

```
<?php
$host = "localhost";
$db = " proyecto ";
$user = " gestor ";
$pass = " secreto ";
$dsn = "mysql:host=$host;dbname=$db;charset=utf8mb4";
```

```

try {
    $conProyecto = new PDO($dsn, $user, $pass);
    $conProyecto->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $resultado = $conProyecto->query("SELECT producto, unidades FROM stocks");
    $conProyecto = null; //cerramos la conexion
} catch (PDOException $e) {
    die("Error en la conexión o la consulta: " . $e->getMessage());
}
?>
<?php while ($registro = $resultado->fetch(PDO::FETCH_OBJ)): ?>
    <p><?= "Producto {$registro->producto}: {$registro->unidades} unidades." ?></p>
<?php endwhile ?>

```

- ✓ **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.

```

<?php
$host = "localhost";
$db = "proyecto";
$user = "gestor";
$pass = "secreto";
$dsn = "mysql:host=$host;dbname=$db;charset=utf8mb4";
try {
    $conProyecto = new PDO($dsn, $user, $pass);
    $conProyecto->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $resultado = $conProyecto->query("SELECT producto, unidades FROM stocks");
    $resultado->bindColumn(1, $producto);
    $resultado->bindColumn(2, $unidades);
    $conProyecto = null; //cerramos la conexion
} catch (PDOException $e) {
    die("Error en la conexión o la consulta: " . $e->getMessage());
}
?>
<?php while ($registro = $resultado->fetch(PDO::FETCH_BOUND)): ?>
    <p><?= "Producto $producto: $unidades unidades." ?></p>
<?php endwhile ?>

```

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.

```

<?php
$host = "localhost";
$db = "proyecto";
$user = "gestor";
$pass = "secreto";
$dsn = "mysql:host=$host;dbname=$db;charset=utf8mb4";
try {
    $conProyecto = new PDO($dsn, $user, $pass);
    $conProyecto->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $resultado = $conProyecto->query("SELECT producto, unidades FROM stocks");
}

```

```
$registros = $resultado->fetchAll();
$conProyecto = null; //cerramos la conexión
} catch (mysqli_sql_exception $e) {
    die("Error en la conexión o la consulta: " . $e->getMessage());
}
?>
<?php foreach ($registros as $registro): ?>
    <p><?= "Producto {$registro['producto']}: {$registro['unidades']} unidades." ?></p>
<?php endforeach ?>
```

Ejercicio Obtención y utilización de resultados 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**.

En este caso extraemos el código asociado a la conexión a la base de datos a un script aparte:

conexion.php

```
<?php

$host = "localhost";
$db = "proyecto";
$user = "gestor";
$pass = "secreto";
$dsn = "mysql:host=$host;dbname=$db;charset=utf8mb4";
try {
    $conProyecto = new PDO($dsn, $user, $pass);
    $conProyecto->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    die("Error en la conexión a la base de datos: " . $e->getMessage());
}
```

index.php:

```
<?php
require_once 'conexion.php';

if (filter_has_var(INPUT_POST, 'enviar')) {
    $petStock = true;
    $codProd = filter_input(INPUT_POST, 'producto', FILTER_UNSAFE_RAW);
    try {
        $consultaProducto = "select nombre , nombre_corto from productos w";
        $resultadoConsultaProducto = $conProyecto->query($consultaProducto);
        $producto = $resultadoConsultaProducto->fetch(PDO::FETCH_OBJ);
    } catch (PDOException $e) {
        $errorConsultaProducto = true;
    }
    try {
        $consultaStock = "select unidades, tiendas.nombre as tienda_nombre";
        $resultadoStockProducto = $conProyecto->query($consultaStock);
        $stocksProducto = $resultadoStockProducto->fetchAll(PDO::FETCH_OBJ);
    } catch (PDOException $e) {
        $errorConsultaStock = true;
    }
} else {
    try {
        $consultaProductos = "select id, nombre, nombre_corto from product";
        $resultadoProductos = $conProyecto->query($consultaProductos);
        $productos = $resultadoProductos->fetchAll(PDO::FETCH_OBJ);
    } catch (Exception $exc) {
        $errorConsultaProductos = true;
    }
}
?>
<!DOCTYPE html>
<html lang="es">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport"
            content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <!-- enlaces para usar Bootstrap -->
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/t
            integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzcDSCmG1MXxSR1C
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/t
            integrity="sha384-C6RzsynM9kWDrMNeT87bh950GNYZPhcTNXj1NW7RuBCsyN/o
        <title>Conjuntos de resultados en PDO</title>
    </head>
    <body class="bg-info">
        <h3 class="text-center mt-2 font-weight-bold">Unidades de un producto</h3>
        <div class="container mt-3">
            <?php if (isset($petStock)): ?>
                <h4 class="mt-3 mb-3 text-center">Unidades del Producto: <i></i><a href=<?= $_SERVER['PHP_SELF'] ?>" class="btn btn-success"><?php if (isset($errorConsultaStock)): ?>
                    <p class="font-weight-bold text-success mt-3">Problema: <?php else: ?>
                        <table class="table table-striped table-dark">
                            <thead>
                                <tr class="text-center font-weight-bold">
                                    <th>Unidad </th>
                                    <th>Tienda </th>
                                    <th>Nombre </th>
                                    <th>Stock </th>
                                </tr>
                            </thead>
                            <tbody>
                                <tr class="text-center font-weight-bold">
                                    <td>1 </td>
                                    <td>Tienda 1 </td>
                                    <td>Nombre 1 </td>
                                    <td>Stock 1 </td>
                                </tr>
                                <tr class="text-center font-weight-bold">
                                    <td>2 </td>
                                    <td>Tienda 2 </td>
                                    <td>Nombre 2 </td>
                                    <td>Stock 2 </td>
                                </tr>
                            </tbody>
                        </table>
                <?php else: ?>
                    <table class="table table-striped table-dark">
                        <thead>
                            <tr class="text-center font-weight-bold">
                                <th>Unidad </th>
                                <th>Tienda </th>
                                <th>Nombre </th>
                                <th>Stock </th>
                            </tr>
                        </thead>
                        <tbody>
                            <tr class="text-center font-weight-bold">
                                <td>1 </td>
                                <td>Tienda 1 </td>
                                <td>Nombre 1 </td>
                                <td>Stock 1 </td>
                            </tr>
                            <tr class="text-center font-weight-bold">
                                <td>2 </td>
                                <td>Tienda 2 </td>
                                <td>Nombre 2 </td>
                                <td>Stock 2 </td>
                            </tr>
                        </tbody>
                    </table>
                <?php endif; ?>
            </div>
        </div>
    </body>
</html>
```

```

<th>Nombre Tienda</th>
<th>Stock</th>
</tr>
</thead>
<tbody>
<?php foreach ($stocksProducto as $stockProducto):
    <tr>
        <td><?= $stockProducto->tienda_nombre ?>
        <td class="textcenter"><?= $stockProducto->stock ?>
    </tr>
<?php endforeach ?>
</tbody>
</table>
<?php endif ?>
<?php else: ?>
    <form name="formulario" action="<?= $_SERVER['PHP_SELF'] ?:>">
        <div class="form-group">
            <label for="p" class="font-weight-bold">Elige un producto:</label>
            <?php if (!isset($errorConsultaProductos)): ?>
                <select class="form-control" id="p" name="producto">
                    <?php foreach ($productos as $producto): ?>
                        <option value="<?= $producto->id ?>"><?= $producto->nombre ?>
                    <?php endforeach ?>
                </select>
            <?php else: ?>
                <h4><?= "No se han podido recuperar productos" ?>
            <?php endif ?>
        </div>
        <div class="mt-2">
            <input type="submit" class="btn btn-warning me-3" value="Consultar Stock" <?php if (isset($errorConsultaProductos)): ?>
        </div>
    </form>
</div>
<?php endif ?>
</body>
</html>

```

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

3.2.5.- 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.



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** 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.

A continuación, se muestran ejemplos de uso de ambas funciones:

```
// Uso bindParam sustituyendo signos de interrogación y parámetros con nombre  
$cod_producto = "TABLET";  
$nombre_producto = "Tablet PC";  
$consulta->bindParam(1, $cod_producto, PDO::PARAM_STR, 20);  
$consulta->bindParam(2, $nombre_producto, PDO::PARAM_STR, 80);  
$consulta->bindParam(':cod', $cod_producto, PDO::PARAM_STR, 20);  
$consulta->bindParam(':nombre', $nombre_producto, PDO::PARAM_STR, 80);  
  
//Uso bindValue sustituyendo signos de interrogación y parámetros con nombre  
$cod_producto = "TABLET";  
$nombre_producto = "Tablet PC";
```

```
$consulta->bindValue(1, $cod_producto, PDO::PARAM_STR, 20);
$consulta->bindValue(2, $nombre_producto, PDO::PARAM_STR, 80);
$consulta->bindValue(':cod', $cod_producto, "TABLET");
$consulta->bindValue(':nombre', $nombre_producto, "Tablet PC");
```

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`, 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.

Mostramos un ejemplo completo de consulta a la base de datos con consultas preparadas.

```
<?php
$host = "localhost";
$db = " proyecto ";
$user = " gestor ";
$pass = " secreto ";
$dsn = "mysql:host=$host;dbname=$db";

if (filter_has_var(INPUT_POST, "buscar_productos")) {
    try {
        $bd = new PDO($dsn, $user, $pass);
        $bd->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        $familia = filter_input(INPUT_POST, 'familia', FILTER_SANITIZE_STRING);
```

```

$consultaProductos = 'SELECT id, nombre, pvp FROM productos WHERE familia=:familia';
$stmtConsultaProductosPorFamilia = $bd->prepare($consultaProductos);
$stmtConsultaProductosPorFamilia->bindParam(':familia', $familia, PDO::PARAM_STR, 30);
$stmtConsultaProductosPorFamilia->execute();
$productos = $stmtConsultaProductosPorFamilia->fetchAll(PDO::FETCH_OBJ);
$stmtConsultaProductosPorFamilia = null;
$bd = null;
} catch (PDOException $e) {
    die("Error en operación con la base de datos: " . $e->getMessage());
}
}

?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Conversión de números decimales</title>
        <link rel="stylesheet" href="stylesheet.css">
    </head>
    <body>
        <div class ="page">
            <h1>Productos de una familia</h1>
            <form class="form" name="form_conversion_base_x"
                action=<?= $_SERVER['PHP_SELF'] ?>" method="POST">
                <div class="input-section">
                    <label for="familia">Familia:</label>
                    <input id="familia" value=<?= $familia ?? '' ?>" name="familia"/>
                </div>
                <div class="submit-section">
                    <input class="submit" type="submit"
                        value="Buscar" name="buscar_productos" />
                </div>
                <?php if (filter_has_var(INPUT_POST, "buscar_productos")): ?>
                    <table>
                        <thead>
                            <tr>
                                <th scope="col">Id</th>
                                <th scope="col">Nombre</th>
                                <th scope="col">Precio</th>
                            </tr>
                        </thead>
                        <tbody>
                            <?php foreach ($productos as $producto): ?>
                                <tr>
                                    <td><?= $producto->id ?></td>
                                    <td><?= $producto->nombre ?></td>
                                    <td> <?= $producto->pvp ?></td>
                                </tr>
                            <?php endforeach ?>
                        </tbody>
                    </table>
                <?php endif ?>
            </form>
        </div>
    </body>
</html>

```

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 Consultas Preparadas 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.

conexion.php:

```
<?php

$host = "localhost";
$db = " proyecto ";
$user = " gestor ";
$pass = " secreto ";
$dsn = "mysql:host=$host;dbname=$db;charset=utf8mb4";
try {
    $conProyecto = new PDO($dsn, $user, $pass);
    $conProyecto->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    die("Error en la conexión a la base de datos: " . $e->getMessage());
}
```

index.php

```
<?php
require_once 'conexion.php';
```

```

$consultaProductos = "select id, nombre, nombre_corto from productos order by id asc";
$consultaProducto = "select nombre , nombre_corto from productos where id=:id";
$consultaStock = "select unidades, tienda as tienda_id, producto as producto from stocks";
$actualizaStock = "update stocks set unidades=:unidades where producto=:producto";

if (filter_has_var(INPUT_POST, 'enviar')) {
    $petStock = true;
    $codProd = filter_input(INPUT_POST, 'producto');
    try {
        $stmtConsultaProducto = $conProyecto->prepare($consultaProducto);
        $stmtConsultaProducto->execute([':id' => $codProd]);
        $producto = $stmtConsultaProducto->fetch(PDO::FETCH_OBJ); //esta linea es importante
        $stmtConsultaProducto = null;
    } catch (PDOException $e) {
        $errorConsultaProducto = true;
    }
    try {
        $stmtConsultaStock = $conProyecto->prepare($consultaStock);
        $stmtConsultaStock->execute([':id' => $codProd]);
        $stocksProducto = $stmtConsultaStock->fetchAll(PDO::FETCH_OBJ);
        $stmtConsultaStock = null;
    } catch (PDOException $e) {
        $errorConsultaStock = true;
    }
} elseif (filter_has_var(INPUT_POST, 'enviar_stock')) {
    $petActualStock = true;
    $codTienda = filter_input(INPUT_POST, 'codigo_tienda', FILTER_UNSAFE_RAW);
    $codProducto = filter_input(INPUT_POST, 'codigo_producto', FILTER_UNSAFE_RAW);
    $unidades = filter_input(INPUT_POST, 'stock', FILTER_UNSAFE_RAW);
    try {
        $stmtActualizaStock = $conProyecto->prepare($actualizaStock);
        $stmtActualizaStock->execute([':unidades' => $unidades, ':producto' => $codProducto]);
        $stmtActualizaStock = null;
    } catch (PDOException $e) {
        $errorActualizacionStock = true;
    }
} else {
    try {
        $stmtConsultaProductos = $conProyecto->prepare($consultaProductos);
        $stmtConsultaProductos->execute();
        $productos = $stmtConsultaProductos->fetchAll(PDO::FETCH_OBJ);
        $stmtConsultaProductos = null;
    } catch (PDOException $e) {
        $errorConsultaProductos = true;
    }
}
?>
<!DOCTYPE html>
<html lang="es">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport"
            content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
        <meta http-equiv="X-UA-Compatible" content="ie=edge">
        <!-- enlaces para usar Bootstrap -->
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" integrity="sha384-T4RQI3Bapeshow" rel="stylesheet">
    </head>
    <body>
        <div class="container">
            <h1>Listado de Productos</h1>
            <table border="1">
                <thead>
                    <tr>
                        <th>ID</th>
                        <th>Nombre</th>
                        <th>Nombre Corto</th>
                        <th>Unidades</th>
                        <th>Tienda</th>
                    </tr>
                <tbody>
                    <tr>
                        <td>1</td>
                        <td>Monitor de 24"</td>
                        <td>Monitor24</td>
                        <td>10</td>
                        <td>Tienda 1</td>
                    </tr>
                    <tr>
                        <td>2</td>
                        <td>Teclado mecánico</td>
                        <td>TecladoMechanico</td>
                        <td>5</td>
                        <td>Tienda 2</td>
                    </tr>
                    <tr>
                        <td>3</td>
                        <td>Mouse inalámbrico</td>
                        <td>MouseInalambrico</td>
                        <td>8</td>
                        <td>Tienda 3</td>
                    </tr>
                    <tr>
                        <td>4</td>
                        <td>Monitor de 27"</td>
                        <td>Monitor27</td>
                        <td>7</td>
                        <td>Tienda 4</td>
                    </tr>
                    <tr>
                        <td>5</td>
                        <td>Teclado gaming</td>
                        <td>TecladoGaming</td>
                        <td>3</td>
                        <td>Tienda 5</td>
                    </tr>
                </tbody>
            </table>
            <div class="text-right">
                <a href="#">Actualizar stock</a>
            </div>
        </div>
    </body>
</html>

```

```
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/t
integrity="sha384-C6RzsynM9kWDrMNeT87bh950GnyZPhcTNXj1NW7RuBCsyN/o
<title>Sentencias preparadas en PDO</title>
</head>
<body class="bg-info">
    <h3 class="text-center mt-2 font-weight-bold">Actualización de stock</h3>
    <div class="container mt-3">
        <?php if (isset($petStock)): ?>
            <h4 class="mt-3 mb-3 text-center">Unidades del Producto: <i></i><?php echo $petStock['stock'] ?></h4>
            <a href=""<?= $_SERVER['PHP_SELF'] ?>" class="btn btn-success">Actualizar</a>
            <?php if (isset($errorConsultaStock)): ?>
                <p class="font-weight-bold text-success mt-3">Problemas con la consulta</p>
            <?php else: ?>
                <table class="table table-striped table-dark">
                    <thead>
                        <tr class="text-center font-weight-bold">
                            <th>Nombre Tienda</th>
                            <th>Stock</th>
                        </tr>
                    </thead>
                    <tbody>
                        <?php foreach ($stocksProducto as $stockProducto): ?>
                            <tr>
                                <td><?= $stockProducto->tienda_id ?></td>
                                <td class="textcenter">
                                    <form name="formulario_actualiza_stock">
                                        <div class="input-group">
                                            <input type="number" class="form-control" name="stock" value="<?= $stockProducto->stock ?>"/>
                                            <input type="hidden" name="id" value="<?= $stockProducto->id ?>"/>
                                            <input type="hidden" name="tienda_id" value="<?= $stockProducto->tienda_id ?>"/>
                                            <input type="submit" class="btn btn-primary" value="Actualizar"/>
                                        </div>
                                    </form>
                                </td>
                            </tr>
                        <?php endforeach ?>
                    </tbody>
                </table>
            <?php endif ?>
        <?php elseif (isset($petActualStock)): ?>
            <p class="font-weight-bold text-success mt-3">
                <?= isset($errorActualizacionStock) ? "Problemas con la actualización" : "Actualización exitosa" ?>
            </p>
            <a href=""<?= $_SERVER['PHP_SELF'] ?>" class="btn btn-success">Volver</a>
        <?php else: ?>
            <form name="formulario" action="<?= $_SERVER['PHP_SELF'] ?>">
                <div class="row">
                    <label for="productos" class="font-weight-bold">Elige el producto:</label>
                    <?php if (!isset($errorConsultaProductos)): ?>
                        <select class="form-control" id="productos" name="id">
                            <?php foreach ($productos as $producto): ?>
                                <option value="<?= $producto->id ?>"><?= $producto->nombre ?></option>
                            <?php endforeach ?>
                        </select>
                    <?php else: ?>
                        <h4><?= "No se han podido recuperar productos" ?></h4>
                    <?php endif ?>
                </div>
                <div class="mt-2">
                    <input type="text" class="form-control" id="stock" name="stock" placeholder="Introduce el stock" value="<?= $errorActualizacionStock ?>"/>
                    <input type="submit" class="btn btn-primary" value="Actualizar"/>
                </div>
            </form>
        <?php endif ?>
    </div>
<div class="mt-2">
```

```
<input type="submit" class="btn btn-warning m-3" value="Consultar" />
<?= isset($errorConsultaProductos) ? "disabled" : "enabled" ?>
</div>
</form>
</div>
<?php endif ?>
</body>
</html>
```

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, 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.
- ✓ `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`.

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`](#).

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.

Como acabamos de comentar, a partir de PHP 7 el motor de ejecución solo genera excepciones. Sin embargo, todavía es posible generar errores de usuario y manejárselos 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:

```
<?php  
// Función de manejo de errores de usuario  
function miManejadorErrores($errno, $errstr, $errfile, $errline) {  
    echo "<b>Mi ERROR</b> [$errno] $errstr<br>\n";  
    echo "Error en la línea $errline en el fichero $errfile<br>\n";  
}  
  
//Establece el manejador de usuario  
set_error_handler("miManejadorErrores");  
  
$test = 100;  
  
//triggering user-defined error handler function  
if ($test == 100) {  
    trigger_error("Se ha generado un error de usuario", E_USER_ERROR);  
}
```

Para saber más

Puedes acceder al manual de PHP para conocer el resto de funciones relacionadas con la gestión de errores.

[Funciones de Manejo de Errores](#)

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.



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

- ✓ 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.

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

```
<?php
$dividendo = 100;
$divisor = 10;
try {
    if ($divisor == 0)
        throw new Exception("División por cero.");
    $resultado = $dividendo / $divisor;
    echo "El resultado es $resultado <br/>";
} catch (Exception $e) {
    echo "Se ha producido el siguiente error: " . $e->getMessage() . "<br/>";
} finally {
    echo "Esto se ejecuta siempre";
}
```

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. Por ejemplo en el siguiente código la excepción generada en el `metodoA` se maneja en el `metodoC`.

```
function metodoA()
{
    throw new Exception('error del metodo A');
}

function metodo B()
```

```

{
    metodoA();
}

function metodoC()
{
    try {
        metodoB();
    } catch (Exception $e) {
        // Maneja el error producido en el metodo A
    }
}

```

Es posible establecer multiples bloques de captura `catch` que definan la estrategia para tratar distintos tipos de excepciones. Por ejemplo, en el siguiente ejemplo se utilizará el manejador asociado a la `excepcionA`. Una vez encontrado un bloque `catch` adecuado ya no se ejecutará ningún bloque posterior aunque también sea aplicable dada la jerarquía de excepciones.

```

class ExcepcionA extends Exception{}

class ExcepcionB extends ExcepcionA{}

try {
    metodoQueLanzaExcepcionA();

} catch (ExceptionA $e) {
// manejador de excepcionA

} catch (ExceptionB $e) {
// manejador de excepcionB

} catch (Exception $e) {
// manejador de excepcion de alto nivel
}

```

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
  |- AssertionException 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.

```

<?php

class cadenaException extends Exception
{

```

```

public function miMensajeError()
{
    //mensaje de error
    $msgError = 'Error en la línea ' . $this->getLine() . ': <b>' . $this->getMessage() .
    return $msgError;
}

class numericoException extends Exception
{
    public function miMensajeError()
    {
        //mensaje de error
        $msgError = 'Error en la línea ' . $this->getLine() . ': <b>' . $this->getMessage() .
        return $msgError;
    }
}

function checkTipo($nombre, $edad)
{
    if (!is_string($nombre)) {
        throw new cadenaException($nombre);
    }
    if (!is_numeric($edad)) {
        throw new NumericoException($edad);
    } else {
        echo $nombre . " tiene la edad " . $edad . '<br/>';
    }
}

try {
    echo checkTipo("Sara", 25) . "\n";
    echo checkTipo(5, 10) . "\n";
}
catch (cadenaException $e) {
    echo $e->miMensajeError();
}
catch (numericoException $e) {
    echo $e->miMensajeError();
}

```

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.

```
<?php

function exception_handler($exception)
{
    echo "Uncaught exception: ", $exception->getMessage(), "\n";
}

set_exception_handler('exception_handler');

throw new Exception('Uncaught Exception');

echo "Not Executed\n";
?>
```

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).

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 coincide 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

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.

```

function myExceptionHandler($e) {
    error_log("{$e->getMessage()} in {$e->getFile()} on line {$e->getLine()}");
    http_response_code(500);
    if (filter_var(ini_get('display_errors'), FILTER_VALIDATE_BOOLEAN)) {
        echo $e;
    } else {
        echo "<h1>500 Internal Server Error</h1>
An internal server error has been occurred.<br>
Please try again later.";
    }
    exit;
}

set_exception_handler('myExceptionHandler');

set_error_handler(function ($level, $message, $file = '', $line = 0) {
    if (!(error_reporting() & $level)) {
        // If the error level is not included in error_reporting, don't handle it
        return;
    }

    if ($level & (E_USER_NOTICE | E_WARNING | E_STRICT | E_DEPRECATED)) {
        // Handle non-fatal errors (log them or take other actions)
        // For example, log the error:
        error_log("Non-fatal error: [$level] $message in $file on line $line\n");
        return false; // Return false to continue script execution
    } else {
        // Convert other errors to ErrorException and throw them
        throw new ErrorException($message, 0, $level, $file, $line);
    }
}, E_ALL);

```

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 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'@
```



Visto el funcionamiento básico de la generación y captura de excepciones 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 transformar 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:

```
try {
    $pdo->prepare("INSERT INTO users VALUES (NULL,?, ?, ?, ?)")->execute($data);
} catch (PDOException $e) {
    $existingkey = "Integrity constraint violation: 1062 Duplicate entry";
    if (strpos($e->getMessage(), $existingkey) !== FALSE) {

        // Take some action if there is a key constraint violation, i.e. duplicate name
    } else {
        throw $e;
    }
}
```