

Tema 3: Trabajar con bases de datos en PHP

Desarrollo web en entorno servidor

2 DE NOVIEMBRE DE 2022

CIFP CARLOS III - CARTAGENA
SANTIAGO FRANCISCO SAN PABLO RAPOSO
2º CURSO DAW

Contenido

Índice de ilustraciones.	1
Índice de tablas.	1
1.- Acceso a bases de datos desde PHP.	2
2.- MySQL.	2
2.1.- Instalación y configuración.	3
2.2.- Herramientas de administración.	4
2.2.1.- MySQL y mySQLAdmin.	6
2.2.2.- phpMyAdmin.	7
3.- Utilización de bases de datos MySQL en PHP.	9
3.1.- Extensión MySQLi.	10
3.1.1.- Establecimiento de conexiones.	11
3.1.2.- Ejecución de consultas.	12
3.1.3.- Transacciones.	13
3.1.4.- Obtención y utilización de conjuntos de resultados.	14
3.1.5.- Consultas preparadas (clase mysqli_stmt).	15
3.2.- PHP Data Objects (PDO).	17
3.2.1.- Establecimiento de conexiones.	18
3.2.2.- Ejecución de consultas.	19
3.2.3.- Obtención y utilización de conjuntos de resultados.	20
3.2.4.- Consultas preparadas.	21
4.- Errores y manejo de excepciones.	23
4.1.- Excepciones.	25

Índice de ilustraciones.

No se encuentran elementos de tabla de ilustraciones.

Índice de tablas.

No se encuentran elementos de tabla de ilustraciones.

Resumen del tema 3

1.- Acceso a bases de datos desde PHP.

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 (ya sea para enviar consultas al gestor de la base de datos, o para solicitar eliminar o actualizar registros).

PHP soporta más de 15 sistemas gestores de bases de datos: SQLite, Oracle, SQL Server, PostgreSQL, IBM DB2, MySQL, etc.

Hasta la versión 5 de PHP, el acceso a las bases de datos se hacía principalmente utilizando extensiones específicas para cada sistema gestor de base de datos (extensiones nativas): Es decir, que si queríamos acceder a una base de datos 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.** Aunque a veces podemos seguir prefiriendo el uso de extensiones nativas por tener acceso a más funciones específicas del sistema gestor, o para una mejor velocidad.

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.**

En esta unidad, **vas a ver cómo acceder a PHP a bases de datos MYSQL utilizando tanto PDO como la extensión nativa MySQLi.** Previamente verás una pequeña introducción al manejo de MySQL.

Además, para el acceso a las funcionalidades de ambas extensiones deberás utilizar objetos.

En PHP se utiliza la palabra **new** para crear un 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: [las características más básicas de la utilización de objetos en PHP.](#)

2.- MySQL.


MySQL es un sistema gestor de bases de datos (SGBD) relacionales. Es un programa de código abierto que **se ofrece bajo licencia GNU GPL (Community Edition), aunque también ofrece una licencia comercial** (Standard Edition, Enterprise Edition) en caso de que quieras utilizarlo **para desarrollar aplicaciones de código propietario.**

Incorpora múltiples motores de almacenamiento, cada uno con características propias: unos son más veloces, otros, aportan mayor seguridad o mejores capacidades de búsqueda.

- **MyISAM** (motor por defecto): es muy rápido, pero a cambio no contempla integridad referencial ni tablas transaccionales.
- **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**.

Para saber más: [manual online de la instalación, configuración y herramientas de administración de MySQL](#).



Autoevaluación

¿A qué hacen referencia las siglas PDO?

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

Correcto, los motores de almacenamiento de los que hablamos son MyISAM e InnoDB. Aunque no son los únicos que se pueden utilizar con MySQL sí son los más comunes.

2.1.- Instalación y configuración.

La instalación de MySQL en Linux se divide en dos paquetes:

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

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

```
sudo service mysql status // también start, stop, restart
```

En una instalación típica, **el usuario root no tiene por defecto contraseña** de acceso al servidor. **Es importante asignarle una por razones de seguridad**.

```
mysqladmin -u root password nueva-contraseña
```

El servidor **se ejecuta por defecto en el puerto TCP 3306**. Debes permitir el acceso a través del firewall en configuraciones en red.

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

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

Para saber más: las opciones de ejecución se pueden aplicar (además de por el fichero global de configuración), por línea de comandos y obtener de otros orígenes distintos. [Manual online de la instalación, configuración y herramientas de administración de MySQL](#).

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

- **port**: indica el puerto TCP en el que escuchará el servidor y con el que se establecerán las conexiones.
- **user**: nombre del usuario que se utilizará para ejecutar el servidor.
- **datadir**: directorio del servidor en el que se almacenarán las bases de datos.

Para saber más: sobre opciones de configuración disponibles en MySQL. [Manual online de la instalación, configuración y herramientas de administración de MySQL](#)

2.2.- Herramientas de administración.

Existen muchas herramientas que permiten establecer una conexión con un servidor MySQL para realizar tareas de administración. **Herramientas incluidas en el servidor MySQL:**

- **mysql**. Permite conectarse a un servidor MySQL para ejecutar sentencias SQL.
- **mysqladmin**. Es un cliente específico para tareas de administración.
- **mysqlshow**. Muestra información sobre bases de datos y tablas.

Para saber más: [sobre las utilidades que incorpora MySQL](#) (Not found). [MySQL :: MySQL 5.0 Reference Manual :: 8 Programas cliente y utilidades MySQL \(archive.org\)](#)

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:

```
mysql -u root -p
```

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: destacar dos herramientas de administración independientes que podemos utilizar con MySQL:

- **MySQL Workbench:** herramienta genérica con interface gráfico nativo que permite **administrar tanto el servidor como las bases de datos que este gestiona. Es de los creadores de MySQL**, y se ofrece en dos ediciones, una de ellas de código abierto bajo licencia GPL.
- **phpMyAdmin:** es una aplicación web muy popular para la administración de servidores MySQL. Presenta 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.**



Autoevaluación

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

Ejercicio de relacionar

Herramienta.	Relación.	Tipo de interface.
MySQL Workbench.	3	1. Línea de comandos.
mysql.	1	2. Web.
phpMyAdmin.	2	3. Nativo.
mysqladmin.	4	4. Línea de comandos.

Reiniciar

Mostrar las respuestas

Tu puntuación es 4/4.

Es importante conocer las diferentes herramientas de administración de MySQL.

2.2.1.- MySQL y mySQLAdmin.

MySQL

La forma más fácil y habitual de utilizar la herramienta **MySQL** es en **modo interactivo**. Una vez te conectas al servidor MySQL, te presenta una línea de órdenes en la que puedes introducir sentencias SQL que se ejecutarán sobre la BB.DD seleccionada, y algunos comandos especiales.

Las sentencias SQL deben terminar en “;”. 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 BB.DD “dwes”, debes hacer:

```
mysql> use dwes
```

Debes conocer: la [sintaxis de las sentencias SQL admitidas por MySQL](#).

También puedes usar mysql en modo de procesamiento por lotes, para **ejecutar** sobre un servidor MySQL **todas las sentencias almacenadas en un fichero de texto** (normalmente con extensión .sql). **Por ejemplo**:

```
mysql -u root -pabc123. < crear_bd_dwes.sql
```

Ejercicio resuelto: utiliza las sentencias SQL que contiene el siguiente fichero para crear la estructura de la base de datos “dwes” en tu instalación de MySQL.

MySQLAdmin

Es una herramienta **no interactiva orientada a tareas de administración del propio servidor**. Dichas tareas 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**:

```
mysqladmin -u root -pabc123. status
```

Debes conocer: [los comandos que admite mysqladmin y su significado.](#)



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.

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

2.2.2.- phpMyAdmin.

Se instala mediante el siguiente comando:

```
sudo apt-get install phpmyadmin
```

Te pregunta por el servidor web a utilizar (escoger Apache2), y después 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>

Para poder entrar, debes indicar un nombre de usuario y contraseña válidos. Si realizaste el ejercicio anterior, se habrá creado en tu servidor un usuario "dwes" con contraseña "abc123." con permisos para la base de datos "dwes".

A la izquierda se muestra un panel de navegación, donde se muestran las bases de datos, y un panel principal con un menú superior y una serie de acciones e información en la parte central.

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.

phpMyAdmin

Base de datos: dwes (4)

dwes (4)

- familia
- producto
- stock
- tienda

Servidor: localhost:3316 Base de datos: dwes

Estructura SQL Buscar Generar una consulta Exportar Importar Diseñador Operaciones

Tabla	Acción	Registros ¹	Tipo	Cotejamiento	Tamaño	Residuo a depurar
familia		15	InnoDB	utf8_spanish_ci	16.0 KB	-
producto		26	InnoDB	utf8_spanish_ci	48.0 KB	-
stock		38	InnoDB	utf8_spanish_ci	32.0 KB	-
tienda		3	InnoDB	utf8_spanish_ci	16.0 KB	-
4 tabla(s)	Número de filas	82	MyISAM	utf8_spanish_ci	112.0 KB	0 Bytes

Marcar todos/as / Desmarcar todos

Vista de impresión Diccionario de datos

Crear nueva tabla en la base de datos dwes

Nombre: Número de campos:

Continuar

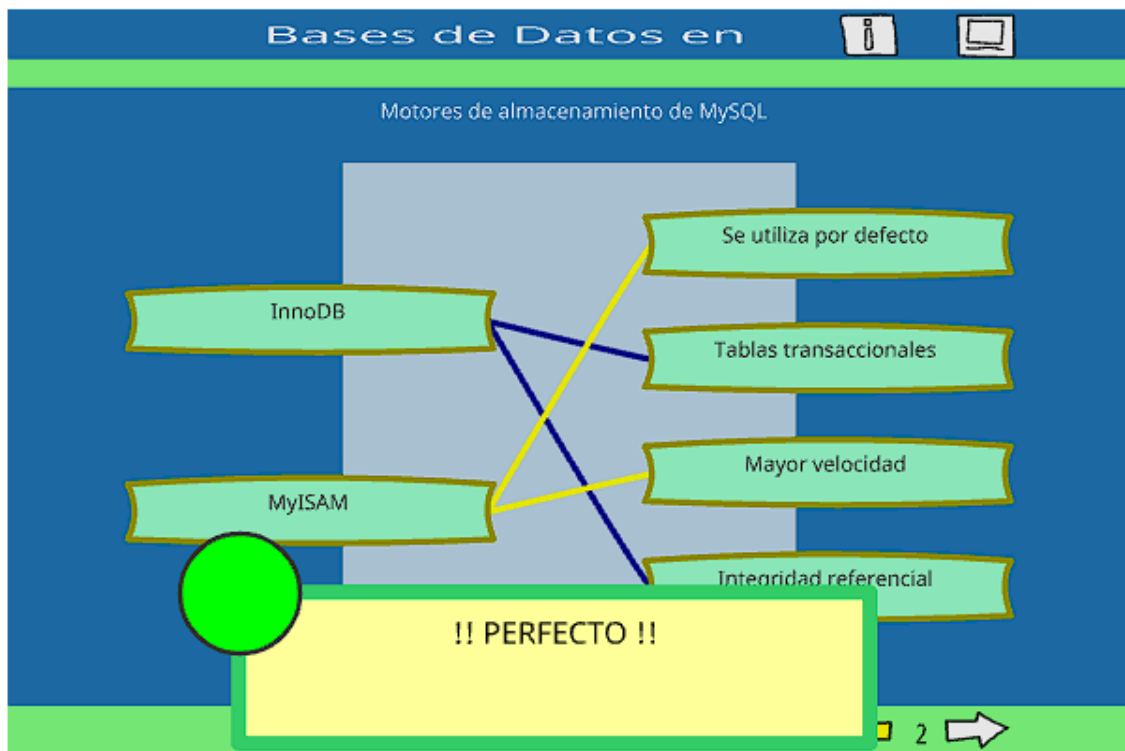
¹ Podría ser aproximado. Léase la FAQ 3.11

Abrir nueva ventana de phpMyAdmin

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

Para saber más: en la página web de la aplicación tienes documentación sobre su configuración y utilización.

Ejercicio resuelto: Utiliza phpMyAdmin para ejecutar las consultas del siguiente fichero, que rellenan con datos las tablas de la BB.DD “dwes”. Esta información la utilizaremos en los próximos ejercicios. Selecciona en el panel de la izquierda la base de datos, y utilizando el menú SQL pega todo el texto del fichero (las consultas a ejecutar) y pulsa el botón "Continuar" para ejecutarlas.





3.- Utilizaci3n de bases de datos MySQL en PHP.

Caso pr3ctico: tendr3 que escoger entre una extensi3n nativa, MySQLi, y PDO. Adem3s, diseña unas pruebas para llevar a cabo y poder tomar una decisi3n. Siempre es mejor asegurarse antes de empezar, aunque eso implique alargar algo m3s los plazos.

Existen dos formas de comunicarse con una BB.DD desde PHP: mediante una extensi3n nativa programada para un SGBD concreto, o utilizar una extensi3n que soporte varios tipos de bases de datos.

Tradicionalmente las conexiones se establec3an utilizando la extensi3n nativa **mysql**. Esta extensi3n 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 m3s habitual es elegir entre **MySQLi** (extensi3n nativa) y **PDO**. Con cualquiera de ambas extensiones, **podr3s realizar acciones sobre las bases de datos como:**

- Establecer **conexiones**.
- **Ejecutar sentencias SQL**.
- **Obtener** los **registros** afectados o devueltos por una sentencia SQL.
- Emplear **transacciones**.
- **Ejecutar procedimientos** almacenados.

- **Gestionar los errores** que se produzcan durante la conexión o en el establecimiento de la misma.

PDO y MySQLi (y también la antigua extensión mysql) **utilizan un driver de bajo nivel** para comunicarse con el servidor MySQL. Hasta hace poco, el único driver disponible era **libmysql**, que no estaba optimizado para ser utilizado desde PHP. A partir de la versión PHP 5.3, PHP 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 manera indiferente.

Por ejemplo: para establecer una conexión con un servidor MySQL y consultar su versión, podemos utilizar cualquiera de las siguientes formas:

```
// utilizando constructores y métodos de la programación orientada a objetos
$conexion = new mysqli('localhost', 'usuario', 'contraseña', 'base_de_datos');
print $conexion->server_info;

// utilizando llamadas a funciones
$conexion = mysqli_connect('localhost', 'usuario', 'contraseña', 'base_de_datos');
print mysqli_get_server_info($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: [información sobre la instalación y utilización de la extensión, incluyendo las funciones y métodos propios de la extensión.](#)

Mejoras que aporta a la antigua extensión mysql:

- Interface **orientado a objetos**.
- **Soporte para transacciones**.
- **Soporte para consultas preparadas**.
- **Mejores opciones de depuración**.

Las opciones de configuración de PHP 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** de configuración de 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: [lista completa de las directivas relacionadas con la extensión MySQLi que se pueden utilizar en php.ini](#). [Más información](#).



Autoevaluación

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

- ☐ Orientado a objetos únicamente.
☒ Dos interfaces de programación: procedimental y orientado a objetos.

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

3.1.1.- Establecimiento de conexiones.

Si utilizas la extensión MySQLi, se puede establecer una **conexión con el servidor** de tres formas diferentes:

Método	Cómo se hace
Constructor de la clase mysqli	<p>El constructor de la clase puede recibir seis parámetros (todos opcionales), aunque lo más habitual es usar los 4 primeros:</p> <pre>mysqli(nombre/dirección IP del servidor MySQL, username, password, nombre BBDD, puerto, socket o tubería con nombre (named pipe) a usar).</pre> <p>Ejemplo: para conectarte a la base de datos “dwes” del ejemplo práctico del tema:</p> <pre>// utilizando el constructor de la clase \$dwes = new mysqli('localhost', 'dwes', 'abc123.', 'dwes');</pre>
Constructor + Función connect	<p>Se utiliza el constructor para crear la instancia, y el método connect para realizar la conexión.</p> <p>Ejemplo:</p> <pre>// utilizando el método connect \$dwes = new mysqli(); \$dwes->connect('localhost', 'dwes', 'abc123.', 'dwes');</pre>
Interface procedimental	<p>Ejemplo:</p> <pre>// utilizando llamadas a funciones \$dwes = mysqli_connect('localhost', 'dwes', 'abc123.', 'dwes');</pre>

Para verificar la conexión con la BB.DD, puedes usar las siguientes **propiedades**. Estas propiedades (o **funciones equivalentes**) de la clase mysqli te dirán si se ha producido un error:

- **connect_errno** (o la función **mysqli_connect_errno**) devuelve el número de error o null si no se produce ningún error.

- **connect_error** (o la función **mysqli_connect_error**) devuelve el mensaje de error o null si no se produce ningún error.

Por ejemplo: el siguiente código comprueba el establecimiento de una conexión con la base de datos “dwes” y finaliza la ejecución si se produce algún error:

```
@ $dwes = new mysqli('localhost', 'dwes', 'abc123.', 'dwes');
$error = $dwes->connect_errno;
if ($error != null) {
    echo "<p>Error $error conectando a la base de datos: $dwes->connect_error</p>";
    exit();
}
```

[PHP: Operadores de control de errores - Manual](#) (@)

Cambiar de conexión/cerrar conexión

- Puedes también querer **cambiar de base de datos**: mediante **select_db**.

```
// utilizando el método connect
$dwes->select_db('otra_bd');
```

- También puedes **cerrar la conexión** (**close**) una vez finalizadas las tareas, para liberar los recursos que utiliza:

```
$dwes->close()
```

3.1.2.- Ejecución de consultas.

Método	Cómo se hace
query	Es la forma más inmediata. Es equivalente a la función <code>mysql_query</code> .
	Si se ejecutan consultas de acción (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).
	Si se ejecutan consultas de datos (SELECT), estos se devuelven a través del objeto mysqli_result .
	Tiene un parámetro opcional que afecta a cómo se obtienen internamente los resultados , pero no a la forma de utilizarlos posteriormente. <ul style="list-style-type: none"> • MYSQLI_STORE_RESULT (por defecto): los resultados se recuperan todos juntos de la base de datos y se almacenan de forma local. • MYSQLI_USE_RESULT: los datos se van recuperando del servidor según se van necesitando.

Ejemplo:

```
$resultado = $dwes->query('SELECT producto, unidades FROM stock',
MYSQLI_USE_RESULT);
```

real_query

(O en su defecto, la función **mysqli_real_query**), siempre devuelve true o false según se haya ejecutado correctamente o no.

- Si la consulta devuelve un conjunto de resultados, **se pueden recuperar utilizando el método store_result**
- También se pueden recuperar según vaya siendo necesario utilizando el **método use_result**.

Es importante saber que **para liberar el espacio utilizado en memoria** por los resultados almacenados, podemos utilizar el **método free** de la clase mysqli_result (o bien 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.

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.

3.1.3.- Transacciones.

Debes 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**).

```
$dwes->autocommit(false); // deshabilitamos el modo transaccional automático
```

¿Qué conseguimos desactivando las transacciones automáticas?

Las operaciones sobre la base de datos iniciarán una transacción que deberás finalizar utilizando:

- **commit**: (o la función **mysqli_commit**). Devuelve true si se ha efectuado correctamente los cambios.
- **rollback** (o la función **mysqli_rollback**). Devuelve true si se han revertido los cambios al estado de la BB.DD en el último **commit**.

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

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

Ejercicio resuelto: Según la información que figura en la tabla stock de la base de datos dwes, 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.

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). Se ejecuta bien solo la primera vez, pues en ejecuciones posteriores ya no es posible insertar la misma fila en la tabla.

[Ver ejercicio resuelto.](#)



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.

3.1.4.- Obtención y utilización de conjuntos de resultados.

Al ejecutar una consulta que devuelve datos, **obtienes un objeto de la clase mysqli_result**. Esta clase sigue el criterio de ofrecer un interface de programación dual (una función por cada método con la misma funcionalidad que este).

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

Método	Cómo se hace
fetch_array()	Obtiene un registro completo del conjunto de resultados y lo almacena en un array. Devuelve null cuando no haya más registros en el conjunto de resultados (se entiende mejor con un ejemplo de recorrido con bucle).
	Por defecto el array contiene tanto claves numéricas como asociativas (podemos referenciar a un campo por su posición o por su nombre).
	<p>Ejemplo:</p> <pre>\$resultado = \$dwes->query('SELECT producto, unidades FROM stock WHERE unidades<2'); \$stock = \$resultado->fetch_array(); // Obtenemos el primer registro \$producto = \$stock['producto']; // O también \$stock[0]; \$unidades = \$stock['unidades']; // O también \$stock[1]; print "<p>Producto \$producto: \$unidades unidades.</p>";</pre> <p>Se puede especificar qué tipo de claves queremos usar, utilizando un parámetro adicional, que puede tomar los siguientes valores:</p>

	<ul style="list-style-type: none"> • MYSQLI_NUM. Devuelve un array con claves numéricas. • MYSQLI_ASSOC. Devuelve un array asociativo. • MYSQLI_BOTH. Es el comportamiento por defecto, en el que devuelve un array con claves numéricas y asociativas.
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 .

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

```
$resultado = $dwes->query('SELECT producto, unidades FROM stock WHERE unidades<2');
$stock = $resultado->fetch_object();
while ($stock != null) {
    print "<p>Producto $stock->producto: $stock->unidades unidades.</p>";
    $stock = $resultado->fetch_object();
}
```

Para saber más: [más información sobre los métodos y propiedades de la clase **mysqli_result**](#).

Ejercicio resuelto: Crea una página web en la que se muestre el stock existente 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. Ver ejercicio resuelto.](#)

3.1.5.- Consultas preparadas (clase **mysqli_stmt**).

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. Esto significa que **podemos guardar en el servidor MySQL una serie de consultas que podremos volver a usar más tarde sin volver a definir las.**

Utilizando el método **stmt_init** de la clase **mysqli** (o la función **mysqli_stmt_init**), obtienes un objeto de dicha clase:

```
$dwes = new mysqli('localhost', 'dwes', 'abc123.', 'dwes');
$consulta = $dwes->stmt_init();
```

Pasos a seguir para ejecutar una consulta preparada:

- 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:

```
$consulta = $dwes->stmt_init();
$consulta->prepare('INSERT INTO familia (cod, nombre) VALUES ("TABLET", "Tablet PC")');
$consulta->execute();
$consulta->close();
$dwes->close();
```

En este ejemplo, se insertan siempre los mismos valores, por lo que **no es útil**.

Ahora sí, vamos a ver cómo crear consultas preparadas genéricas de inserción y actualización de datos

1. 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.**

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

2. Antes de ejecutar la consulta, el **método bind_param** (o la función **mysqli_stmt_bind_param**) sirve para **sustituir cada parámetro por su valor.**
Sintaxis:

```
bind_param(cadena donde cada carácter indica el tipo de un parámetro, parametro1, ..., parametroN)
```

Carácter	Tipo de parámetro
I (Integer)	Número entero.
D (Double)	Número real (doble precisión).
S (String)	Cadena de texto.
B (Binary)	Contenido en formato binario (BLOB = Binary Large Object).

Aviso importante

Muy importante pasar los parámetros (parámetro1, parámetroN) mediante variables, ya que se pasan por referencia. Si se pasan literales, se generará un error.

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

Ejemplo:

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

Consultas preparadas de obtención de datos

Podemos usar el método **bind_result** (o bien la función **mysqli_stmt_bind_result**) para **asignar a variables los campos que se obtienen tras la ejecución**. **Ejemplo:**

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

Debes conocer: [información sobre consultas preparadas y la clase **mysqli_stmt**](#).

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

En esta ocasión es necesario crear un nuevo formulario en la página, en la sección donde se muestra el número de unidades por tienda. Cuando se envía ese formulario, hay que preparar la consulta y ejecutarla una vez por cada registro de la tabla stock (una vez por cada tienda en la que exista stock de ese producto). [Revisa el código de la solución y pruébalo.](#)

3.2.- PHP Data Objects (PDO).

MySQLi ofrece acceso a todas las características del motor de BB.DD, 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 ello, antes de comenzar el desarrollo del proyecto, en el caso de que exista la posibilidad de utilizar otro servidor como almacenamiento, deberás adoptar una capa de abstracción para el acceso a los datos.

Existen varias alternativas como ODBC (Open DataBase Connectivity), pero sin duda la opción más recomendable en la actualidad es **PDO**. De esta forma, si en el futuro tienes que cambiar de SGBD, tendrás que realizar modificaciones mínimas en el código.

Así, podrás preparar aplicaciones para utilizar un almacenamiento u otro según se indique en el momento de la ejecución, pero este no es el objetivo principal de PDO, ya que 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.

La extensión PDO debe utilizar un driver específico para el tipo de BB.DD 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**.

3.2.1.- Establecimiento de conexiones.

Basta con instanciar un objeto de la clase PDO pasándole los siguientes parámetros (solo el primero es obligatorio):

```
PDO(origen de datos DSN, username, password, connectionOptions)
```

- **Origen de datos (DSN):** cadena de texto (separadas por el carácter dos puntos (:))
 - Indica el **controlador** que se va a utilizar
 - **Parámetros específicos necesarios por el controlador:**
 - **dirección IP** del servidor.
 - **nombre de la BB.DD.**
- **Nombre de usuario** con permisos para establecer la conexión.
- **Contraseña del usuario.**
- **Opciones de conexión**, almacenadas en forma de Array.

Por ejemplo: podemos establecer una conexión con la BB.DD “dwes” creada anteriormente:

```
$dwes = new PDO('mysql:host=localhost;dbname=dwes', 'dwes', 'abc123.');
```

Para conectar con MySQL, podemos utilizar los siguientes parámetros específicos en la cadena DSN:

- **host:** nombre o dirección IP del servidor.
- **port:** número de puerto TCP en el que escucha el servidor.
- **bbname:** nombre de la BB.DD.
- **unix_socket:** socket de MySQL en sistemas Unix.

Para indicar opciones de conexión mediante Array: por ejemplo, si quisiéramos indicar al servidor MySQL que queremos utilizar codificación UTF-8 para los datos que se transmitan, puede usar una opción específica de la conexión:

```
$opciones = array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8");
$dwes = new PDO('mysql:host=localhost;dbname=dwes', 'dwes', 'abc123.', $opciones);
```

Para saber más: información sobre los **controladores** existentes, parámetros de las cadenas DSN y opciones de conexión particulares de cada uno. [Ver aquí](#).

Obtener y modificar parámetros de la conexión una vez establecida: puedes usar 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 = $dwes->getAttribute(PDO::ATTR_SERVER_VERSION);
print "Versión: $version";
```

Y si, por ejemplo, quieres que todos los nombres de columnas se devuelvan en mayúsculas:

```
$version = $dwes->setAttribute(PDO::ATTR_CASE, PDO::CASE_UPPER);
```

Para saber más: sobre las funciones [getAttribute](#) y [setAttribute](#), y sus posibles parámetros.



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".

3.2.2.- Ejecución de consultas.

Tipo de consulta	Cómo se hace
Consulta de acción	Esto es, INSERT, DELETE o UPDATE . El método exec devuelve el número de registros afectados.
	Ejemplo: <pre><code>\$registros = \$dwes->exec('DELETE FROM stock WHERE unidades=0'); print "<p>Se han borrado \$registros registros.</p>";</code></pre>
Consulta de datos	Esto es, SELECT . Debes utilizar para ello el método query , que devuelve un objeto de la clase PDOStatement .
	Ejemplo: <pre><code>\$dwes = new PDO("mysql:host=localhost;dbname=dwes", "dwes", "abc123."); \$resultado = \$dwes->query("SELECT producto, unidades FROM stock");</code></pre>

3.2.2.1.- Transacciones

Por defecto, PDO **trabaja en modo "autocommit"**, como MySQLi. 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, y se volverá al modo automático.
- **commit.** Confirma la transacción actual.
- **rollback.** Revierte los cambios llevados a cabo en la transacción actual.

Ejemplo:

```
$ok = true;
$dwes->beginTransaction();
```

```
if($dwes->exec('DELETE ...') == 0) $ok = false;
if($dwes->exec('UPDATE ...') == 0) $ok = false;
...
if ($ok) $dwes->commit(); // Si todo fue bien confirma los cambios
else $dwes->rollback(); // y si no, los revierte
```

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

Ejercicio resuelto: De una forma similar al anterior ejercicio de transacciones, utiliza PDO para repartir entre las tiendas las tres unidades que figuran en stock del producto con código PAPYRE62GB.

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



Autoevaluación

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

- ☐ Sí.
- ☒ No.

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

3.2.3.- Obtención y utilización de conjuntos de resultados.

Al igual que con MySQLi, en PDO tienes varias posibilidades para tratar con el conjunto de resultados devuelto por el método **query**.

Método	Cómo se hace
fetch() de la clase PDOStatement	Devuelve un registro del conjunto de resultados , o false si ya no quedan registros por recorrer.
	Ejemplo: <pre>\$dwes = new PDO("mysql:host=localhost;dbname=dwes", "dwes", "abc123."); \$resultado = \$dwes->query("SELECT producto, unidades FROM stock"); while (\$registro = \$resultado->fetch()) { echo "Producto ".\$registro['producto'].": ".\$registro['unidades']."
"; }</pre>
	<p>Por defecto, genera y devuelve a partir de cada registro, un array con claves numéricas y asociativas.</p> <p>Para cambiar su comportamiento, admite un parámetro opcional en el que le podemos especificar uno de los siguientes valores:</p> <ul style="list-style-type: none"> • PDO::FETCH_ASSOC: devuelve solo un array asociativo. • PDO::FETCH_NUM: devuelve solo un array con claves numéricas.

- **PDO::FETCH_BOTH**: devuelve un array con claves numéricas y asociativas. **Es el comportamiento por defecto.**
- **PDO::FETCH_OBJ**: devuelve un objeto cuyas propiedades son los campos del registro.
- **PDO::FETCH_LAZY**: devuelve tanto el objeto como el array con clave dual anterior.
- **PDO::FETCH_BOUND**: devuelve true y asigna los valores del registro a variables, según se indique con el método *bindColumn*. Este método debe ser llamado una vez por cada columna, indicando en cada llamada el n° de columna (empezando en 1) y la variable a asignar.

Ejemplo con PDO::FETCH_OBJ:

```
$dwes = new PDO("mysql:host=localhost;dbname=dwes", "dwes",
"abc123.");
$resultado = $dwes->query("SELECT producto, unidades FROM stock");
while ($registro = $resultado->fetch(PDO::FETCH_OBJ)) {
    echo "Producto ".$registro->producto." ".$registro-
>unidades."<br />";
}
```

Ejemplo con PDO::FETCH_BOUND:

```
$dwes = new PDO("mysql:host=localhost;dbname=dwes", "dwes",
"abc123.");
$resultado = $dwes->query("SELECT producto, unidades FROM stock");
$resultado->bindColumn(1, $producto);
$resultado->bindColumn(2, $unidades);
while ($registro = $resultado->fetch(PDO::FETCH_OBJ)) {
    echo "Producto ".$producto." ".$unidades."<br />";
}
```

Ejercicio resuelto: Modifica la página web que muestra el stock de un producto en las distintas tiendas, obtenida en un ejercicio anterior utilizando MySQLi, para que use PDO. Omite la gestión de errores, que veremos en el último punto de este tema. [Comprueba en la solución propuesta los cambios realizados respecto a la solución del ejercicio equivalente que utilizaba MySQLi.](#)



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".

3.2.4.- Consultas preparadas.

Consultas de acción

Al igual que en MySQLi, también podemos preparar consultas parametrizadas en el servidor para ejecutarlas de forma repetida.

1. 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:

```
$dwes = new PDO("mysql:host=localhost;dbname=dwes", "dwes", "abc123.");  
$consulta = $dwes->prepare('INSERT INTO familia (cod, nombre) VALUES (?, ?)');
```

O también utilizando parámetros con nombre, no sin antes un símbolo de dos puntos (:):

```
$dwes = new PDO("mysql:host=localhost;dbname=dwes", "dwes", "abc123.");  
$consulta = $dwes->prepare('INSERT INTO familia (cod, nombre) VALUES (:cod, :nombre)');
```

2. Antes de ejecutar la consulta, **hay que asignarle un valor a los parámetros con bindParam** de la clase PDOStatement. Si utilizas signos de interrogación para marcar los parámetros, el procedimiento es equivalente al método bindColumn del apartado anterior.

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

Si utilizas parámetros con nombres, debes indicar ese nombre en la llamada a **bindParam**:

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

Tal como ocurría con MySQLi, **cuando uses bindParam, deberás usar siempre variables**, ya que se pasan los valores por referencia.

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

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

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

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

Para saber más: [sobre la utilización en PDO de consultas preparadas y la clase PDOStatement](#).

Ejercicio resuelto: Modifica el ejercicio sobre consultas preparadas que realizaste con la extensión MySQLi, el que modificaba el número de unidades de un producto en las distintas tiendas, para que utilice ahora la extensión PDO.

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

4.- Errores y manejo de excepciones.

Caso práctico: se pueden obtener errors porque no esté disponible el servidor de bases de datos, o porque se acaba de borrar de la base de datos un registro al que estaba accediendo. Todos estos y otros muchos casos, deben tener comportamientos sólidos y coherentes. Deben manejar los errores.

Algunos errores son reconocidos por el entorno de desarrollo (NetBeans), y puedes corregirlos antes de ejecutar. Otros aparecen en el navegador en forma de mensaje de error al ejecutar el guión.

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. **Cada nivel de error tiene definida en PHP una constante.**

- **E_NOTICE:** hace referencia a **avisos** que pueden indicar un error al ejecutar el guión.
- **E_WARNING:** advertencias en tiempo de ejecución (errores no fatales). La ejecución del script no se interrumpe.
- **E_ERROR:** engloba **errores fatales** que provocan que se interrumpa forzosamente la ejecución.
- **E_ALL:** todos los errores y advertencias soportados.

Debes conocer: [la lista completa de constantes, donde también se describe el tipo de errores que representa.](#)

Configuración inicial en php.ini

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**: indica que se notifiquen todos los errores (E_ALL) salvo los avisos en tiempo de ejecución (E_NOTICE).

- **display_errors.** En su valor por defecto (**On**), hace que los mensajes se envíen a la salida estándar (y por lo tanto se muestren en el navegador).
 - Se debe desactivar (**Off**) en los servidores que no se usan para desarrollo sino para producción.

Para saber más: sobre otros parámetros que podemos utilizar en php.ini para ajustar el comportamiento de PHP cuando se produce un error.

Tratamiento de errores desde código

- Puedes usar la función **error_reporting** con las constantes anteriores para establecer el nivel de notificación en un momento determinado.

Por ejemplo: si en algún lugar de tu código figura una división en la que exista la posibilidad de que el divisor sea cero, cuando esto ocurra obtendrás un mensaje de error en el navegador. Para evitarlo, puedes desactivar la notificación de errores de nivel E_WARNING antes de la división y restaurarla a su valor normal a continuación:

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

- Para modificar la gestión de los errores, **se puede crear una función que sea la que ejecute PHP cada vez que se produce un error.**
 - Se utiliza la **función set_error_handler.**
 - **Tiene dos parámetros obligatorios:** el **nivel del error** y el mensaje descriptivo.
 - **Tiene otros tres opcionales con información sobre el error:** nombre del **fichero** en que se produce, número de **línea** y un volcado del **estado** de las variables en ese momento).

Ejemplo:

```
set_error_handler("miGestorDeErrores");
$resultado = $dividendo / $divisor;
restore_error_handler();

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

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

4.1.- Excepciones.

A partir de la versión PHP 5, **se introdujo un modelo de excepciones similar al existente en otros lenguajes de programación**:

- El código susceptible de producir algún error se introduce en un bloque **try**.
- Cuando se produce algún error, **se lanza una excepción utilizando la instrucción throw**.
- Después del bloque try debe haber como mínimo un bloque **catch** encargado de procesar el error.
- **Si una vez acabado el bloque try no se ha lanzado ninguna excepción, se continúa con la ejecución** en la línea siguiente al bloque o bloques catch.

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

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

PHP ofrece una clase base **Exception** para utilizar como manejador de excepciones.

Para lanzar una excepción, no es necesario indicar ningún parámetro, aunque de forma opcional **se puede pasar un mensaje de error** (como en el ejemplo anterior) **y también un código de error**.

Métodos de la clase Exception:

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

Las funciones internas de PHP y muchas extensiones como MySQLi usan el sistema de errores visto anteriormente. **Solo las extensiones más modernas orientadas a objetos, como es el caso de PDO, utilizan este modelo de excepciones. En este caso, lo más común es que la extensión defina sus propios manejadores de errores heredando de la clase Exception** (veremos cómo utilizar la herencia en una unidad posterior).

Para saber más: utilizando la clase `ErrorException`, es posible traducir los errores a excepciones.

Manejo de errores en el uso de extensiones para acceso a BB.DD

Concretamente, la clase **PDO**, permite **definir la fórmula que usará cuando se produzca un error**, utilizando **el atributo PDO::ATTR_ERRMODE**. Las posibilidades son:

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

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

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

Por ejemplo, el siguiente código captura la excepción que lanza PDO debido a que la tabla no existe:

```
$dwes = new PDO("mysql:host=localhost; dbname=dwes", "dwes", "abc123.");
$dwes->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
try {
    $sql = "SELECT * FROM stox";
    $result = $dwes->query($sql);
    ...
}
catch (PDOException $p) {
    echo "Error ". $p->getMessage(). "<br />";
}
```

El bloque catch muestra el siguiente mensaje:

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

Ejercicio resuelto: Agrega control de excepciones para controlar los posibles errores de conexión que se puedan producir en el último ejercicio, mostrando en la parte inferior de la pantalla los errores que se produzcan. [Revisa en la solución propuesta los cambios realizados para utilizar excepciones en el establecimiento de la conexión.](#)





Autoevaluación

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

- ☐ Uno.
- ☒ Uno o 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 el base que incluye PHP, "Exception".

Extensiones de Bas  

Extensin mysqli (1)

Cul es la forma procedimental para establecer conexiones utilizando mysqli?

`mysqli_connect` ✓

En mysqli, de qu clase es el objeto que devuelve una llamada al mtodo query?

`mysqli_result` ✓

Si utilizas mysqli para establecer una conexin, qu mtodo usaras para utilizar otra base de datos?

`mysqli_select_db` ✓

!! PERFECTO !!

2 ➡

En InnoDB por defecto cada consulta se incluye dentro de su propia transaccin. Cul es el nombre de la funcin de mysqli que permite cambiar este comportamiento?

`mysqli_autocommit` ✓

En mysqli, cul debe ser el valor del parmetro del mtodo fetch_array para que devuelva un array asociativo?

`MYSQLI_ASSOC` ✓

Qu clase debes utilizar si quieres ejecutar consultas preparadas en mysqli?

`mysqli_stmt` ✓

!! PERFECTO !!

Extensiones de Bas

PDO

Qu valor debemos pasarle al mtodo fetch para que devuelva solamente un array numrico?

PDO::FETCH_NUM ✓

Qu mtodo se utiliza en PDO para obtener informacin del estado de la conexin?

getAttribute ✓

Qu mtodo se utiliza en PDO para indicar que se quiere comenzar una nueva transaccin?

beginTransaction ✓

Qu mtodo se utiliza en PDO para ejecutar una consulta de tipo DELETE?

exec ✓

!! PERFECTO !!

4

Extensiones de Bas

Errores y manejo de excepciones

Nombre de la funcin de PHP que permite establecer un manejador de errores propio.

set_error_handler ✓

Instruccin que debe seguir a un bloque try.

catch ✓

Parmetro de php.ini que indica qu errores se notificarn.

error_reporting ✓

Clase para manejar excepciones en PHP.

Exception ✓

!! PERFECTO !!

5