

Introducción a las bases de datos y los servicios PHP

PHP



Introducción a las bases de datos y los servicios PHP

Las bases de datos son el combustible para el desarrollo de multitud de aplicaciones y más valiosas que el oro para el crecimiento de las empresas. De ahí que sea necesario dedicar todo un fastbook a ellas.

En este tema aprenderemos, entre otras tareas, a realizar consultas SQL desde código PHP. Pero antes es importante que conozcas los tipos de bases de datos con los que podemos trabajar y sus características principales.

Autor: Jesús Donoso

- Bases de datos
- Bases de datos relacionales
- Operaciones en bases de datos
- Servicio PHP
- Consulta y gestión en bases de datos relacionales desde un servicio PHP
- Conclusiones
- Recursos de interés

Bases de datos



Las bases de datos son colecciones de datos almacenadas de manera estructurada siendo fácilmente accesibles, gestionados y administrados.

Actualmente, las bases de datos desempeñan un papel fundamental en la mayoría de las aplicaciones. Permiten a empresas y organizaciones gestionar y acceder a su información de manera más eficiente para la toma de decisiones o realización de desarrollos. Existen varios tipos de base de datos; a continuación, se describen algunos de ellos:

Relacionales

Se utilizan diversas tablas para organizar los datos y se establecen relaciones entre ellas. Los más populares son MySQL, PostgreSQL u Oracle.

No relacionales

Están diseñadas para manejar de manera más eficiente datos no estructurados y ofrecen una mayor flexibilidad en la gestión de esta clase de data. MongoDB o Redis son ejemplos de las bases de datos no relacionales.

Memoria

También los datos son almacenados en la memoria RAM por lo que este tipo de bases de datos permiten un acceso rápido a la información. Redis, además de ser no relacional, es un ejemplo de base de datos en memoria.

Distribuidas

Se encuentran repartidas en diferentes servidores de almacenamiento de información. Tiene una mejor escalabilidad y disponibilidad de la información. Google o Amazon proporcionan este tipo de almacenamiento.

Orientadas a gráficos

Se utilizan para almacenar y consultar datos de redes y relaciones complejas como puedan ser los de una red social. Un ejemplo de este tipo de base de datos es NEO4j.

En los siguientes apartados veremos con más detalle las **bases de datos relacionales**.

Además, detallaremos la importancia que tienen las bases de datos en el desarrollo de aplicaciones haciendo uso en la programación orientada a objetos. En concreto, queremos que entiendas **cómo puedes crear servicios** que conecten una aplicación con una base de datos para poder acceder a la información que contiene.

Bases de datos relacionales



Las bases de datos relacionales (RDBMS, siglas en inglés de *Relational Database Management System*) son **un tipo de sistemas de gestión de información fundamentados en un modelo de datos relacional**. Se basan en un conjunto de principios conocidos como el modelo relacional, desarrollado por Edgar F. Codd en la década de 1970.

Este modelo consiste en **organizar los datos en tablas que contienen registros**. Cada registro representa una entidad específica y cada columna una propiedad de una determinada entidad. Estas tablas a su vez se relacionan entre ellas para facilitar la búsqueda, recuperación y gestión de los datos.

Componentes de una base de datos relacional

A continuación, vamos a ver con más detenimiento estos **componentes característicos** de las bases de datos relacionales.

1

Tablas

Las tablas son las estructuras fundamentales que organizan y almacenan los datos.

En la programación orientada a objetos una tabla se puede relacionar o asemejar a una clase. Sus **componentes** son los siguientes.

EL NOMBRE DE LA TABLA	LAS FILAS	LAS COLUMNAS
Las tablas se identifican de manera única dentro del modelo de datos por su nombre, y es que se hace uso de ese nombre para referenciar a la tabla cuando se quiera operar con ella.		

EL NOMBRE DE LA TABLA	LAS FILAS	LAS COLUMNAS
Llamadas también registros o tuplas, son las entradas individuales que tienen cada una de las tablas. Representan un elemento o entidad única y su información relacionada. En la programación orientada a objetos una fila se puede asemejar con la instancia de una clase.		

EL NOMBRE DE LA TABLA	LAS FILAS	LAS COLUMNAS
Conocidas también como campos, son los atributos que tiene cada fila. En la programación orientada a objetos cada columna de una tabla se corresponde con las propiedades de una clase.		

A continuación, compartimos un ejemplo muy sencillo de una **tabla de clientes**. En esta tabla cada fila representa un cliente diferente. Las columnas representan las propiedades de los clientes; en este caso nombre, apellido y email.

Tabla de clientes			
ID	Nombre	Apellido	email
1	Luis	López	luis@bdd.com
2	Ana	Martínez	ana@bdd.com
3	Maria	Adan	maria@bdd.com

2

Claves

Una clave es un atributo o un conjunto de atributos de una tabla necesarios para identificar de manera única cada fila.

Son fundamentales en una **estructura de base de datos relacional** para poder establecer las relaciones entre las tablas. Garantizan la integridad y la coherencia de los datos. Estas son las **principales claves**:

Clave primaria (primary key)

Es un atributo o conjunto de atributos de una tabla que garantiza la exclusividad de cada registro. No pueden existir dos tuplas o registros con la misma clave primaria. Usualmente este atributo suele ser un número de identificación único para cada registro. Sin embargo, se puede considerar como clave cualquier otro atributo o conjunto de ellos que identifiquen de manera única los registros. En el ejemplo, más abajo, de una tabla de pedidos, el campo ID que representa el identificador de cada pedido es considerado la clave primaria.

Clave foránea (foreign key)

La clave foránea es un atributo de una tabla que se relaciona con la clave primaria de otra tabla. Estas claves se utilizan para establecer relaciones entre diferentes tablas. Las claves foráneas aseguran que los registros relacionados tienen información válida en la tabla a la que se le relaciona. En la tabla de pedidos de nuestro ejemplo, la propiedad ID de cliente se considera clave foránea, ya que relaciona los pedidos con los clientes que lo han realizado mediante su ID de cliente.

Tabla de pedidos			
ID	Cliente_ID	Producto	Cantidad
1	3	Producto A	2
2	1	Producto B	1
3	1	Producto A	3

Ejemplo de tabla de pedidos.

3

Relaciones

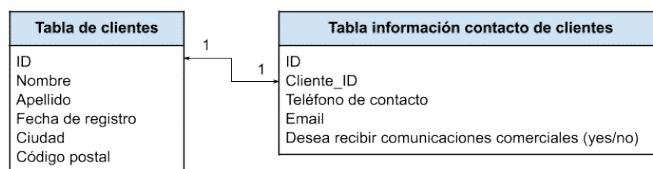
Las relaciones son un concepto totalmente fundamental en las bases de datos relacionales.

Las relaciones son las maneras que tienen las tablas de establecer conexiones entre sí a través de claves primarias y/o foráneas. Permiten la conexión entre tablas para organizar y almacenar los datos de una manera eficiente, evitando la redundancia de estos y facilitando su recuperación.

Existen **varios tipos de relaciones** entre tablas en una base de datos. A continuación, definiremos los tres tipos más comunes:

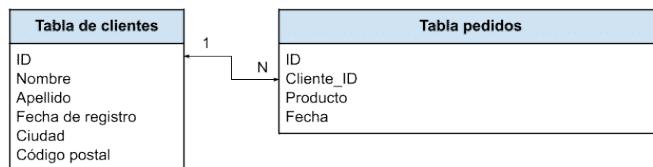
Relación uno a uno (one-to-one)

En este tipo de relación un registro de una tabla se encuentra asociado a un único registro de otra tabla. En el ejemplo, vemos una tabla con la información de los clientes y otra con la información de contacto de estos. Cada cliente únicamente puede tener una entrada en la tabla de información de contacto de clientes. Se puede observar también cómo una clave foránea es la clave primaria de la tabla con la que se está relacionando.



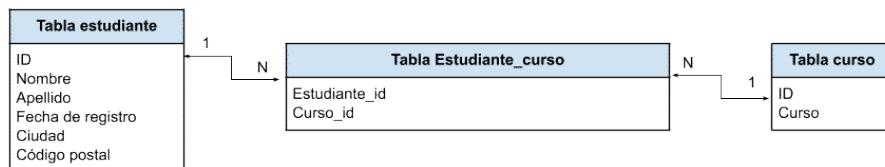
Relación uno a muchos (one-to-many)

En este tipo de relación un registro de una tabla está relacionado a muchos registros en otra tabla. En el ejemplo se puede entender cómo la tabla de pedidos puede encontrarse varios pedidos realizados por un mismo cliente. Al igual que en el caso anterior, una clave foránea es la clave primaria de la tabla con la que se está relacionando.



Relación muchos a muchos (many-to-many)

En este tipo de relación varios registros de una tabla pueden tener relación con varios registros de otra tabla. Para la representación de dicha relación, se genera una tabla intermedia la cual contiene la clave primaria de ambas tablas. En el ejemplo se indica cómo un estudiante puede estar inscrito en varios cursos y un curso puede tener varios alumnos registrados. En la tabla intermedia 'Estudiante_curso', se guardan las claves de la relación que tienen los estudiantes y los cursos.



Integridad de datos

La integridad en una base de datos es el concepto de la precisión, consistencia y validez de los datos almacenados en ella.

Asegura que los datos sean precisos, confiables y coherentes a lo largo del tiempo haciendo uso de reglas y restricciones para ello. Existen varios tipos de integridad en base de datos y cada uno ocupa aspectos específicos de la calidad del dato.

Integridad de entidad

Asegura que cada registro de la tabla sea único e identificable de manera única. Esto se logra a través de la clave primaria de una tabla. Cada registro debe de tener un único valor en el campo de una clave primaria.

Integridad de referencia

Asegura que las relaciones entre tablas se mantienen correctamente. Esto se consigue a través de las claves foráneas, que son los campos de una tabla que se relacionan con la clave primaria de otra. La integridad por referencia garantiza que las claves foráneas existan y siempre tengan los valores necesarios para poder establecer las relaciones.

Integridad de dominio

Asegura que todos los valores de una tupla cumplen con las condiciones que debe de tener cada columna o propiedad. Cada columna puede incluir restricciones de tipo de datos o de los valores de estos.

Integridad de usuario

Asegura que las reglas y restricciones especificadas por el usuario se cumplen para garantizar que los datos almacenados cumplan también con dichas reglas. Por ejemplo, pueden existir reglas que imposibiliten el registro de pedidos en días no laborables.

Integridad lógica

Asegura que los datos mantienen relaciones coherentes y lógicas entre sí. Esto se logra con una buena definición de la estructura de la base de datos. Por ejemplo, una base de datos que registre las ventas de un almacén debe controlar que los productos vendidos tengan stock disponible.

Operaciones en bases de datos



En las bases de datos se pueden realizar diversas operaciones que abarcan desde la consulta de registros de una tabla hasta la manipulación estructural de la misma. Tanto la recuperación de datos como la unión, actualización o eliminación de registros o la creación, modificación y eliminación de tablas constituye una parte esencial de la administración y trabajo con bases de datos.

Creación, modificación y borrado de tablas

La creación, modificación y eliminación de tablas son **operaciones básicas** que se realizan dentro las bases de datos y que requieren del uso de comandos específicos SQL.

1

Creación

Para crear una tabla se utiliza el comando 'CREATE TABLE'. Tras el comando se indica el nombre de la tabla y, entre paréntesis, **el nombre de los atributos y su tipo de dato**. Existen varios tipos de datos para definir una columna.

En el ejemplo que mostramos a continuación se ha creado una tabla 'Empleados' con **diferentes atributos y tipos de datos**.



Para ver todos los tipos de datos disponibles te recomendamos leer la siguiente [documentación](#).

```
CREATE TABLE Empleados (
    ID INT PRIMARY KEY, -- Número entero
    Nombre VARCHAR(50), -- Cadena de texto variable
    Salario DECIMAL(10, 2), -- Número decimal
    Departamento CHAR(30), -- Cadena de texto longitud determinada
    PeriodoPrueba BOOL(), -- Booleano
    FechaNacimiento DATE(), -- Fecha
);
```

Modificación

También es posible **modificar las columnas de una tabla ya creada**. Para ello, es necesario hacer uso del comando *ALTER TABLE* seguido del nombre de la tabla a modificar y de la acción a realizar.

Seguimos con el ejemplo y hacemos uso del comando *ADD COLUMN* para agregar el atributo *FechaContratación* a la tabla de 'Empleados'.

```
ALTER TABLE Empleados
ADD COLUMN FechaContratacion DATE;
```

Por otro lado, también se pueden modificar columnas ya existentes. A continuación, se modifica el atributo *Salario* de la tabla 'Empleados' con el comando *MODIFY COLUMN* para que el tipo de dato de la columna sea un número de 12 dígitos en total e incluya dos decimales.

```
ALTER TABLE Empleados
MODIFY COLUMN Salario DECIMAL(12, 2);
```

También se pueden borrar columnas de una tabla. En el ejemplo se elimina la columna 'FechaContratación' con el comando *DROP COLUMN*.

```
ALTER TABLE Empleados  
DROP COLUMN FechaContratacion;
```

Borrado

Para eliminar una tabla se emplea el comando *DROP TABLE*.

```
DROP TABLE Empleados;
```

Es importante destacar que el uso de este comando elimina permanentemente la tabla y todos sus datos. Se recomienda realizar copias de seguridad antes de ejecutar este tipo de comandos para evitar la pérdida de información esencial.

Consultas sobre las tablas

Las consultas son un concepto básico de las bases de datos.

Una consulta es la solicitud de información a una base de datos para recuperar, modificar, agregar o eliminar datos.

Las consultas son fundamentales para **interactuar con una base de datos y poder operar con la información que almacena**. La capacidad de escribir y ejecutar consultas es totalmente necesaria para el trabajo con bases de datos. En los ejemplos que se comparten a continuación, te explicamos algunas de las consultas más básicas en una base de datos relacional y en lenguaje SQL.

Selección (**SELECT Clause**)

Es la instrucción que permite **seleccionar una o varias columnas** para la obtención de información de la base de datos. La sintaxis básica de una instrucción SELECT es:

```
SELECT columna1, columna2 FROM tabla1;  
SELECT * FROM tabla2;
```

En el ejemplo anterior se seleccionan dos columnas de una tabla y con el símbolo asterisco se seleccionan todas las columnas de otra tabla. Es importante tener en cuenta que, en algunos casos, **el uso del asterisco para seleccionar todos los campos de una tabla puede incurrir en consultas muy pesadas y costosas**. Por lo tanto, aconsejo seleccionar únicamente las columnas que sean necesarias en la consulta.

Filtrado (**WHERE Clause**) y operadores lógicos

En una consulta se hace uso de filtros para obtener únicamente **los registros necesarios de una o varias tablas en base a las restricciones indicadas**. Normalmente, las consultas se filtran haciendo uso de la cláusula WHERE. En esta cláusula se indican la o las condiciones que deben cumplir los registros que se quieran seleccionar de la tabla.

Aquí puedes observar un ejemplo de la **sintaxis**:

```
SELECT columna FROM tabla WHERE (condición1 OR condición2);
```

Por otra parte, los operadores lógicos en SQL y en otros lenguajes de programación son herramientas esenciales que **sirven para filtrar y combinar datos**. Al utilizar la cláusula WHERE en una consulta se pueden aplicar varios operadores lógicos para establecer condiciones específicas. Entre **los operadores más comunes** se encuentran:

1

AND

Devuelve registros que **cumplen simultáneamente con dos o más condiciones**. En el ejemplo, devuelve empleados que trabajan en el departamento de Ventas y tienen un salario superior a 50000.

```
SELECT * FROM empleados WHERE departamento = 'Ventas' AND salario > 50000;
```

2

OR

Devuelve registros que cumplen **al menos una de las condiciones especificadas**. El ejemplo devuelve empleados que trabajan en el departamento de Ventas o tienen un salario superior a 50000.

```
SELECT * FROM empleados WHERE departamento = 'Ventas' OR salario > 50000;
```

3

NOT

Este operador **niega una condición, devolviendo registros que no cumplen con la condición especificada**. El ejemplo devuelve empleados que no trabajan en el departamento de 'Ventas'.

```
SELECT * FROM empleados WHERE NOT departamento = 'Ventas';
```

4

IN

Permite especificar **múltiples valores para una condición**, devolviendo registros que coincidan con alguno de esos valores. El ejemplo devuelve empleados que trabajan en los departamentos de 'Ventas' o 'Marketing'.

```
SELECT * FROM empleados WHERE departamento IN ('Ventas', 'Marketing');
```

5

BETWEEN

Se utiliza para **seleccionar valores dentro de un rango específico**, devolviendo registros que cumplen con esta condición. El ejemplo devuelve los productos cuyo precio se encuentra entre 50 o 100.

```
SELECT * FROM productos WHERE precio BETWEEN 50 AND 100;
```

6

LIKE

Este operador se utiliza para buscar patrones en columnas de texto, devolviendo registros que coincidan con el patrón especificado. El ejemplo devuelve clientes cuyos nombres comienzan con la letra 'A'.

```
SELECT * FROM clientes WHERE nombre LIKE 'A%';
```

Agrupación (**GROUP BY Clause**)

A la hora de realizar operaciones de agrupación la cláusula **GROUP BY** es fundamental. Se utiliza para **agrupar las tuplas que tengan el mismo valor en una o varias columnas** y aplicar operaciones de agregaciones como sumas o conteos.

Aquí tienes un ejemplo de uso.

```
SELECT columnna1, SUM(columnna2) FROM tabla GROUP BY columnna1;
```

Ordenación (**ORDER BY Clause**)

Se hace uso de la cláusula **ORDER BY** para ordenar **los resultados de la consulta en base a uno o varios criterios**. Se puede ordenar de manera ascendente con el parámetro **ASC** y descendente con el parámetro **DESC**. Muchas veces el orden facilita la lectura y análisis de los datos recuperados.

```
SELECT columnna1, columnna2 FROM tabla ORDER BY columnna1 DESC;
```

Actualización (**UPDATE Clause**)

La cláusula **UPDATE** se utiliza para modificar datos existentes en una tabla. Permite cambiar los valores de una o varias columnas en uno o varios registros que cumplan con las condiciones especificadas.

A continuación, te muestro un ejemplo donde, para las tuplas que cumplen la condición especificada en el *WHERE*, se modifican los valores en las columnas uno y dos como se indica en la cláusula *SET*.

```
UPDATE tabla  
SET columna1 = valor1, columna2 = valor2  
WHERE condición;
```

Es importante recalcar que la cláusula *UPDATE* puede afectar a muchos registros si no se integra correctamente la condición *WHERE*. Sin cláusula *WHERE*, se modificarían todos los registros de la tabla.

Inserción (*INSERT Clause*)

La cláusula *INSERT INTO* se utiliza para la agregación de registros a una tabla. La sintaxis básica, como se puede observar a continuación, consiste en indicar la tabla, los valores y las columnas en las que se deben insertar dichos valores.

```
INSERT INTO tabla (columna1, columna2, columna3, ...)  
VALUES (valor1, valor2, valor3, ...)
```

Tras la cláusula `INSERT INTO` se indica **la tabla donde se desea agregar los nuevos registros** y, a continuación, entre paréntesis, las columnas en las que se incluye cada nuevo valor. El parámetro `VALUES` es necesario para indicar los valores que se insertan en la tabla.

El orden en el que se nombran las columnas y los valores indica qué valor se inserta en cada columna.

En el ejemplo, el 'valor1' se inserta en la 'columna1' y así sucesivamente en orden.

Eliminación (*DELETE Clause*)

La palabra clave utilizada para eliminar registros en una tabla es la cláusula ***DELETE***. Se pueden eliminar uno o varios registros a la vez, siempre y cuando cumplan las condiciones que se le indiquen en la consulta.

Echa un vistazo a la sintaxis de *DELETE* en la siguiente imagen.

```
DELETE FROM tabla  
WHERE condición;
```

Importante: si no se especifica una condición en la cláusula `WHERE` se eliminarían todos los registros de la tabla.

Unión (*JOIN Clause*)

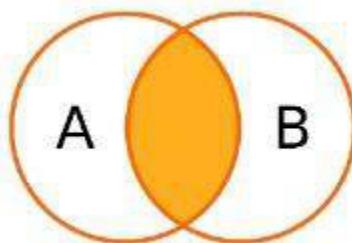
En base de datos, las uniones son operaciones que permiten combinar registros de dos o más tablas en función de una relación específica. **Las uniones son fundamentales para consultar y recuperar datos** cuando los datos están distribuidos en varias tablas relacionadas.

La palabra clave para realizar una unión en SQL es la cláusula *JOIN*. Existen **varios tipos de uniones**; a continuación, vamos a explicar las más frecuentes.

1

INNER JOIN

Devuelve solamente los registros que tienen coincidencias en ambas tablas.

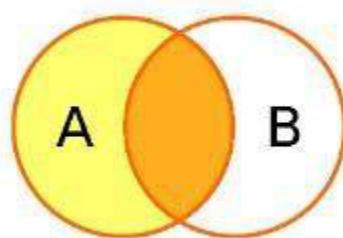


```
SELECT * FROM tabla1 INNER JOIN tabla2 ON tabla1.id =  
tabla2.id_relacionado;
```

2

LEFT JOIN

Devuelve **todos los registros de la tabla izquierda** (primera tabla mencionada) y los registros coincidentes de la tabla derecha (segunda tabla). Si no hay coincidencias en la tabla derecha, se rellena con valores *NULL*.

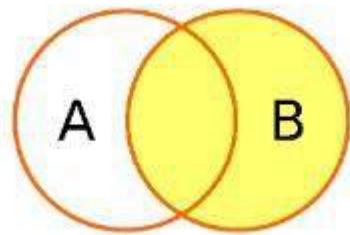


```
SELECT *  
FROM tabla1  
LEFT JOIN tabla2 ON tabla1.id = tabla2.id_relacionado;
```

3

RIGHT JOIN

Es similar a *LEFT JOIN*, pero devuelve todos los registros de la tabla derecha y los registros coincidentes de la tabla izquierda. Los registros que no tienen coincidencias en la tabla izquierda se llenan con valores *NULL*.

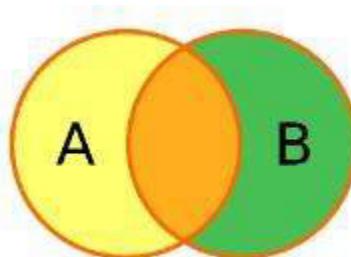


```
SELECT *  
FROM tabla1  
RIGHT JOIN tabla2 ON tabla1.id = tabla2.id_relacionado;
```

4

FULL JOIN

Devuelve todos los registros de ambas tablas. Si no hay coincidencias en una tabla, se rellenan con valores *NULL*.



```
SELECT *
FROM tabla1
FULL JOIN tabla2 ON tabla1.id = tabla2.id_relacionado;
```

Servicio PHP



Una vez introducidos los conceptos básicos de bases de datos, en este apartado, vamos a contextualizarlos en un proyecto en PHP. Para **conectar una base de datos a una aplicación PHP** es necesario **el desarrollo de un servicio**.

Los servicios son los encargados de gestionar las conexiones con las diferentes bases de datos.

Específicamente, un servicio en PHP se refiere a una parte de la aplicación que proporciona funcionalidades específicas o realizar tareas concretas con el servidor. Los servicios en PHP son componentes que pueden ser llamados y utilizados por otras partes de la aplicación o por aplicaciones externas a través de solicitudes HTTP.

Los servicios PHP son **esenciales en la modularidad y reutilización de código en aplicaciones web** y pueden tener diversas funciones. Ahora, vamos a estudiar las razones primordiales para el uso de servicios y lo entenderás mejor.

- **Separación de preocupaciones.** Utilizar un servicio PHP permite separar la lógica de negocio de la presentación. Esto se traduce en mantener el código organizado y modular, lo que facilita su mantenimiento y escalabilidad. La lógica de negocio se suele encontrar en los servicios, mientras las vistas se encargan de la representación de la información.
- **Acceso a bases de datos.** También se utilizan comúnmente los servicios para interactuar con las bases de datos. Pueden ejecutar consultas SQL, recuperar datos, agregar, actualizar o eliminar registros.
- **Servicios webs y APIs.** Permiten la comunicación entre diferentes aplicaciones y servicios. Estos servicios pueden proporcionar datos o funcionalidades a través de HTTP, lo que facilita la integración de sistemas y exposición de ellos.
- **Procesamiento de formularios y solicitudes.** Pueden procesar formularios y solicitudes HTTP entrantes. Pueden validar, procesar la información que es enviada al servicio y devolver respuestas adecuadas.
- **Seguridad y autenticación.** También puede utilizarse un servicio para la gestión de la autenticación autorización de usuarios, verificación de contraseñas, generación de tokens y aplicación de medidas de seguridad. Consiguiendo así proteger el sistema de potenciales amenazas.
- **Manipulación de archivos.** Pueden cargar, descargar, leer y escribir archivos en el servidor. Esto es útil para la descarga y gestión de ficheros de configuración.
- **Envío de correos electrónicos.** Se puede enviar correos electrónicos desde una aplicación web, lo que es útil para notificaciones de correo electrónico, confirmaciones de registro y comunicación con el usuario.
- **Generación de contenido dinámico.** Son capaces de generar contenido dinámico en función de los datos a tiempo real, como resultado de búsquedas.

i Recuerda: los servicios PHP son una herramienta que sirven para proporcionar funcionalidades específicas y realizar tareas en una aplicación. Ayudan a la modularización de código, mejoran la reutilización y permiten la interacción con bases de datos.

Conexión de una base de datos en PHP

De todas las ventajas vistas del uso de servicios en PHP, este apartado vamos a dedicarlo a cómo realizar **una conexión a una base de datos**.

Para establecer una conexión a una base de datos con PHP, generalmente se utilizan extensiones de PHP que son compatibles con el sistema de gestión de la base de datos que se utilice. Las extensiones más comunes para conectarse a una base de datos son MySQLi (*MySQL Improved*) y PDO (*PHP Data Objects*).

En esta tabla MySQLi y PDO podemos **comparar y ver las diferencias**.

	MySQLi	PDO
Soporte de base de datos	Diseñada exclusivamente para trabajar con base de datos MySQL y proporciona características específicas de MySQL.	Más genérica. Puede utilizarse con varios sistemas de gestión de bases datos, como PostgreSQL, SQLite, SQL Server, etc.

Estilo de programación	Ofrece dos estilos de programación: el orientado a objetos y el estilo procedural. Se debe utilizar el que más se adapte al código.	Utiliza un enfoque orientado a objetos de manera predeterminada. Esto significa que todas las interacciones con la base de datos se realizan a través de objetos.
Parámetros nombrados	No admite parámetros nombrados directamente en las consultas preparadas. Se deben utilizar símbolos de interrogación para representar parámetros en las consultas.	Admite parámetros nombrados en las consultas preparadas, lo que hace que las consultas sean más legibles.
Flexibilidad y portabilidad	Es específica para MySQL. Si se utiliza para otro gestor de bases de datos se deben de sobreescribir partes del código.	Es más flexible. Posibilita el uso en diferentes gestores de bases de datos sin cambiar el código.
Soporte de transacciones	Ambas permiten transacciones, lo que permite realizar operaciones atómicas.	Sin embargo, PDO proporciona un manejo más flexible y consistente de las transacciones.
Seguridad	Ambas ofrecen protección sobre la inyección SQL cuando se utilizan consultas preparadas correctamente.	Sin embargo, PDO se considera más seguro debido al soporte integrado para parámetros nombrados.

Viendo esta tabla, la conclusión a la que podemos llegar es que la elección entre MySQLi o PDO dependerá de las necesidades que contenga el proyecto y de la base de datos utilizada. Si se necesita una conexión más flexible y portátil la elección más adecuada es PDO.

A continuación, te comarto **ejemplos en código de la conexión entre PHP** y estas extensiones comentadas. Un ejemplo de **conexión con MySQLi**:

```
$servername = "localhost"; //Nombre del servidor de la base de datos
$username = "usuario"; //Nombre de usuario de la base de datos
$password = "contraseña"; //Contraseña de la base de datos
$database = "nombre_base_de_datos"; // Nombre de la base de datos

// Crear una conexión
$conn = new mysqli($servername, $username, $password, $database);
// Verificar la conexión
if ($conn->connect_error) {
    die("Error de conexión: ". $conn->connect_errno);
}
// Ahora se puede ejecutar consultas SQL usando esta conexión
```

Un ejemplo de **conexión con PDO**:

```
<?php  
$servername = "localhost"; //Nombre del servidor de la base de datos  
$username = "usuario"; //Nombre de usuario de la base de datos  
$password = "contraseña"; //Contraseña de la base de datos  
$database = "nombre_base_de_datos"; // Nombre de la base de datos  
  
try {  
    // Crear una conexión PDO
```

```
$conn = new PDO("mysql:host=$servername;dbname=$database", $username,  
$password);  
// Establecer el modo de error de PDO a excepciones  
$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);  
echo "Conexión exitosa";  
  
} catch (PDOException $e) {  
    die("Error de conexión: ". $e->getMessage());  
}  
// Ahora se puede ejecutar consultas SQL usando esta conexión
```

Consulta y gestión en bases de datos relacionales desde un servicio PHP



Qualentum Lab

Ahora vamos a ver cómo realizar consultas SQL desde código PHP y cómo transformar los resultados obtenidos en objetos utilizando PDO.

En los ejemplos, que te iremos compartiendo, se realiza **una conexión a una base de datos MySQL haciendo uso de una extensión PDO**. Igualmente, según lo explicado en el apartado anterior, podría realizarse con la extensión MySQLi.



Para obtener el detalle sobre cómo se realizan las operaciones de consultas y gestión con MySQLi en su [documentación oficial](#).

¡Empezamos! El primer paso es hacer uso del código especificado en el apartado anterior para crear una conexión a la base de datos haciendo uso de la extensión PDO. Es importante siempre comprobar que el host, el nombre de la base de datos, usuario y contraseñas son correctos.

1

Consultas en PDO

Para realizar peticiones a la base de datos es necesario hacer uso del método de PDO `prepare()`. Dentro de este método, se especifica en formato de cadena de caracteres la consulta a realizar. Aquí puedes **observar un ejemplo en código**:

```
$consulta = $pdo->prepare("SELECT * FROM usuarios WHERE id = :id");
$id = 1;
$consulta->bindParam(':id', $id, PDO::PARAM_INT);
$consulta->execute();
```

Se puede ver que, para dinamizar la consulta, en lugar de **incluir en valor del ID directamente en la consulta**, se hace uso del método `bindParam()` para *bindear* el parámetro.

Este concepto se refiere al proceso de **asociar valores a parámetros de una consulta**. Esto aporta seguridad contra atacantes insertando código malicioso en consultas, optimiza las consultas, promueve su reutilización y mejora la legibilidad del código al separar la consulta de los datos.

Una vez todo está preparado se hace uso del método `execute()` para ejecutar la consulta. Y tras la ejecución de la petición, podemos **recuperar todos los datos como un conjunto de objetos**, haciendo uso del método `fetchAll()` o únicamente como un resultado, haciendo uso de la función `fetch()`.

En ambas funciones es opcional especificar el modo de retorno. En el ejemplo, a continuación, se puede observar el uso de ambas funciones configuradas con el modo de respuesta PDO::FETCH_OBJ.

Los resultados de la consulta serán devueltos como objetos estándar de PHP. Cada fila se convierte en una propiedad del objeto, donde el nombre de la propiedad se corresponde al nombre de la columna en la base de datos.

```
// En esta consulta devuelve todos los resultados que contienen la consulta
$resultados = $consulta->fetchAll(PDO::FETCH_OBJ);
foreach ($resultados as $usuario) {
    echo $usuario->nombre . "<br>";
}
// Solamente devuelve un único valor de lo que devuelve la consulta
$usuario = $consulta->fetch(PDO::FETCH_OBJ);
echo $usuario->nombre;
$pdo = null;
```

Un último paso después de la **ejecución de las consultas** y de la recuperación de los resultados es el cierre de la conexión. Esto es importante para optimizar de forma adecuada los recursos.

Como se puede observar en el ejemplo anterior, el cierre de la conexión se realiza igualando a *null* la variable *\$pdo* de la instancia de la conexión. Se ha visto entonces la sintaxis básica para poder realizar consultas SQL haciendo uso de PDO y cómo transformar los resultados de la consulta en objetos en PHP.

**La prevención de errores es fundamental al trabajar
con PDO.**

Para **garantizar la seguridad y la integridad de las aplicaciones** es importante tener en cuenta algunas buenas prácticas que se detallan en este apartado.

Es importante, por ejemplo, declarar el código necesario para la generación de excepciones en caso de errores y configurar su manejo. En PDO se realiza la gestión de errores haciendo uso de los atributos 'PDO::ATTR_ERRMODE' en 'PDO::ERRMODE_EXCEPTION' durante la creación de una instancia de conexión a una base de datos.

Veamos el código necesario para el uso de estos atributos con el siguiente ejemplo.

```
try {
    // Crear una conexión PDO
    $conn = new PDO("mysql:host=$servername;dbname=$database", $username,
    $password);
    // Establecer el modo de error de PDO a excepciones
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Conexión exitosa";

} catch (PDOException $e) {
    die("Error de conexión: ". $e->getMessage());
}
```

Para el control de errores en las consultas es conveniente la utilización de *try-catch* para **capturar las excepciones y manejar errores de manera adecuada**. En el siguiente ejemplo, puedes observar un ejemplo de uso *try-catch* en el cual captura la excepción de PDO e imprimir el mensaje de error.

```
try {
    $consulta = $pdo->prepare("SELECT * FROM usuarios WHERE id = :id");
    $id = 1;
    $consulta->bindParam(':id', $id, PDO::PARAM_INT);
    $consulta->execute();

    // En esta consulta devuelve todos los resultados que contienen la
    // consulta
    $resultados = $consulta->fetchAll(PDO::FETCH_OBJ);
} catch (PDOException $e) {
    // Maneja el error, registra detalles o toma medidas apropiadas
    echo "Error: ". $e->getMessage();
}
```

Conclusiones



A lo largo de este documento has adquirido aprendido las nociones básicas sobre las bases de datos relacionales y lenguaje SQL, también has ampliado tu comprensión sobre la gestión de datos dentro del contexto de las aplicaciones web con PHP. Para poder llevar a cabo esa gestión de datos desde aplicaciones PHP se han visto extensiones como MySQLi y PDO, recursos que permiten realizar esas conexiones tan necesarias. Hemos visto cómo estas extensiones permiten a los desarrolladores ejecutar consultas SQL de manera eficiente, garantizando así la integridad y seguridad de los datos almacenados.

Ahora ya sabes que las bases de datos son un elemento fundamental hoy en día. La capacidad para diseñar bases de datos eficientes, la ejecución de consultas SQL optimizadas y la comunicación efectiva haciendo uso de los servicios externos supone la creación de una base sólida para el desarrollo de aplicaciones modernas. El uso de los servicios PHP permite la interacción entre las distintas bases de datos facilitando a su vez la creación de aplicaciones web dinámicas y el desarrollo de un código sólido y escalable.

Recursos de interés



Qualentum Lab

Con esta lectura, esperamos haberte acercado al mundo de las bases de datos y cómo trabajar con ellas en los servicios PHP. Como decimos, solo te hemos puesto en el camino. Ahora eres tú el que debe aprender paso a paso con la práctica. Eso sí, no está demás que antes des un repaso a los manuales oficiales que te recomendamos:

- Manual oficial PHP MySQLi.
- Manual oficial PHP PDO.



Puedes acceder desde estos enlaces a los manuales: [Manual oficial PHP MySQLi](#) y [Manual oficial PHP PDO](#).

¡Enhорabuena! Fastbook superado



Qualentum.com