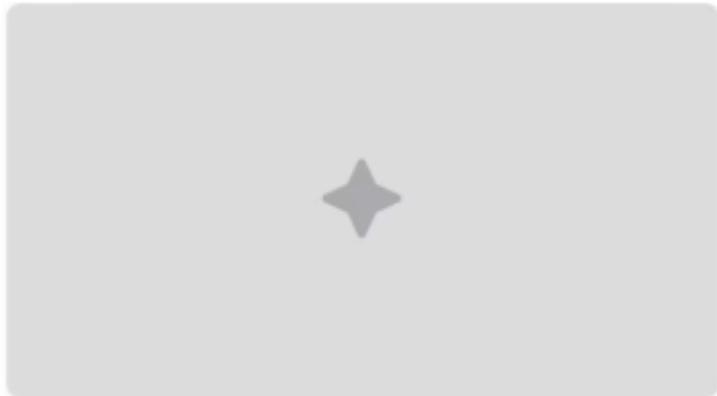


# Bases de Datos con MySQL en PHP: Guía Completa para Principiantes

Bienvenido a esta guía diseñada especialmente para principiantes absolutos. Aprenderás desde cero cómo conectar aplicaciones PHP con bases de datos MySQL, creando sistemas robustos y seguros. No necesitas experiencia previa: cada concepto se explica paso a paso con ejemplos prácticos que podrás seguir inmediatamente.

## Tu instructor: Ivan Malaver Fierro



### SENA CBA Mosquera

Como tu guía en este recorrido por MySQL y PHP, mi compromiso es explicar cada concepto de manera clara y sencilla. Mi enfoque didáctico se basa en explicaciones paso a paso, ejemplos prácticos y ejercicios que podrás seguir fácilmente.

Cada tema está estructurado para que entiendas perfectamente **qué es, para qué sirve y cómo se implementa**. Recuerda: aprender a programar es un proceso, y

estaré aquí para acompañarte en cada etapa del camino.

# ¿Qué vas a aprender en este curso?



## Fundamentos de SQL

Aprenderás qué es SQL, cómo crear bases de datos y tablas, y cómo establecer relaciones entre datos de manera profesional.

## Operaciones CRUD

Implementarás las operaciones básicas: Crear, Leer, Actualizar y Eliminar registros de forma segura y eficiente.

## Conexión PHP-MySQL

Dominarás los métodos mysqli y PDO para conectar tus aplicaciones PHP con bases de datos MySQL de manera

Aprenderás a prevenir inyecciones SQL, gestionar contraseñas de forma segura y proteger los datos de tus usuarios.

## Seguridad Avanzada

profesional.

# Requisitos técnicos y conocimientos previos

gestionar la información.

## PHP 7.4 o superior

Asegúrate de tener una versión moderna de PHP instalada para seguir todos los ejemplos del curso.

### Servidor local

Necesitarás XAMPP, WAMP o similar instalado en tu computadora para poder ejecutar PHP y MySQL.

### MySQL o MariaDB

El motor de base de datos que utilizaremos para almacenar y

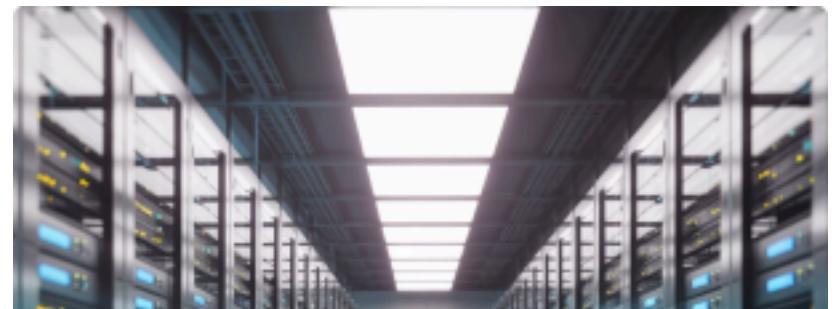
## Editor de código

Visual Studio Code, Sublime Text o cualquier editor que te resulte cómodo para escribir código.

**Conocimientos previos recomendados:** No necesitas experiencia en bases de datos, pero es útil tener nociones básicas de HTML, CSS y conceptos básicos de programación en PHP. ¡No te preocupes si eres principiante total! Explicaré cada concepto desde cero.

## ¿Qué es una base de datos?

Una base de datos es un **almacén organizado de información**. Imagina una colección de libretas donde guardas datos de forma ordenada y



estructurada.

En el mundo digital, las bases de datos nos permiten:

- Guardar grandes cantidades de información de manera eficiente
- Organizarla de forma estructurada y lógica
- Buscar datos rápidamente cuando los necesitamos
- Relacionar diferentes tipos de información entre sí
- Acceder a los datos desde diferentes aplicaciones simultáneamente

Aplicaciones como Facebook, YouTube, Amazon o cualquier tienda online utilizan bases de datos para guardar toda su información: usuarios, productos, comentarios, pedidos y mucho más.

## ¿Qué es MySQL?

MySQL es un **sistema de gestión de bases de datos relacionales**. Es como el programa que nos permite crear y administrar muchas libretas de datos a la vez de manera profesional.

### ¿Para qué sirve MySQL?

Crear bases de datos y tablas para organizar información de manera estructurada

### Consultas rápidas

Permitir consultas ultrarrápidas para recuperar exactamente la información que necesitas

## Almacenamiento eficiente

Almacenar millones de datos de forma optimizada ocupando el menor espacio posible

Mantener la integridad y consistencia de los datos mediante reglas y restricciones

## Integridad de datos

MySQL es **gratuito, rápido y muy popular** para aplicaciones web, por eso lo encontrarás en millones de sitios web y tutoriales en todo el mundo.

# ¿Qué es SQL?

SQL (**Structured Query Language**) es el lenguaje que usamos para comunicarnos con bases de datos como MySQL. Es como el idioma que debemos hablar para pedirle al sistema que guarde, busque, modifique o elimine datos.

1

## DDL - Definir datos

Comandos para crear, modificar o eliminar tablas y bases de datos completas.

CREATE TABLE, ALTER TABLE, DROP TABLE

3

## DCL - Controlar acceso

Comandos para gestionar permisos, seguridad y acceso a la información.

GRANT, REVOKE

2

## DML - Manipular datos

Comandos para insertar, actualizar, eliminar o consultar datos almacenados.

INSERT, UPDATE, DELETE, SELECT

## TCL - Controlar transacciones

Comandos para gestionar cambios en grupo y garantizar la consistencia.

COMMIT, ROLLBACK

4

SQL es un lenguaje **declarativo**: le dices **QUÉ** quieres hacer, no **CÓMO** hacerlo. El sistema se encarga de los detalles de implementación.

# Operaciones CRUD: La base de todo

Las operaciones CRUD son las **acciones básicas** que realizamos con los datos. CRUD es un acrónimo que representa las cuatro operaciones fundamentales en cualquier sistema de gestión de datos.

## CREATE (Crear)

Añadir nuevos datos a la base de datos.

```
INSERT INTO usuarios  
(nombre, email)  
VALUES ('Ana',
```

```
'ana@ejemplo.com');
```

```
FROM usuarios  
WHERE edad > 18;
```

## UPDATE (Actualizar)

Modificar datos que ya existen en la base de datos.

```
UPDATE usuarios  
SET nombre = 'Ana García'  
WHERE id = 5;
```

## READ (Leer)

Consultar o recuperar datos existentes de las tablas.

```
SELECT nombre, email
```

## DELETE (Eliminar)

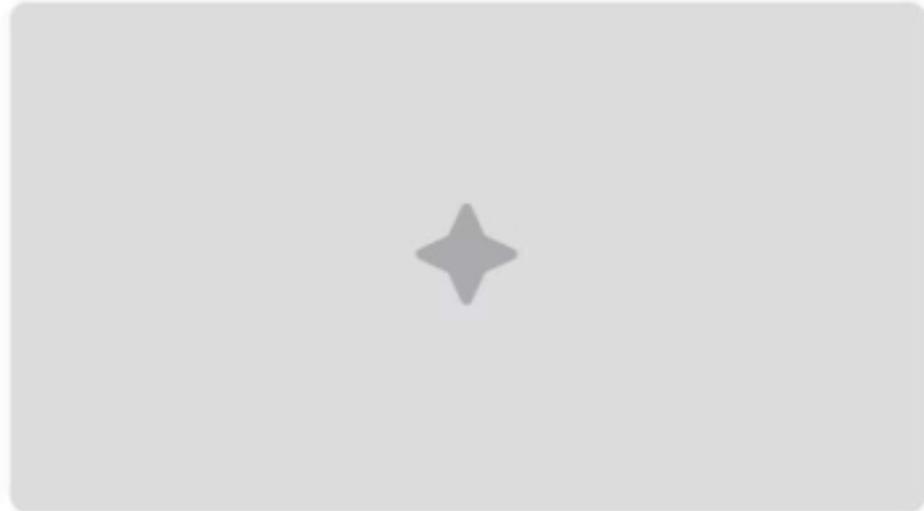
Borrar datos de la base de datos de forma permanente.

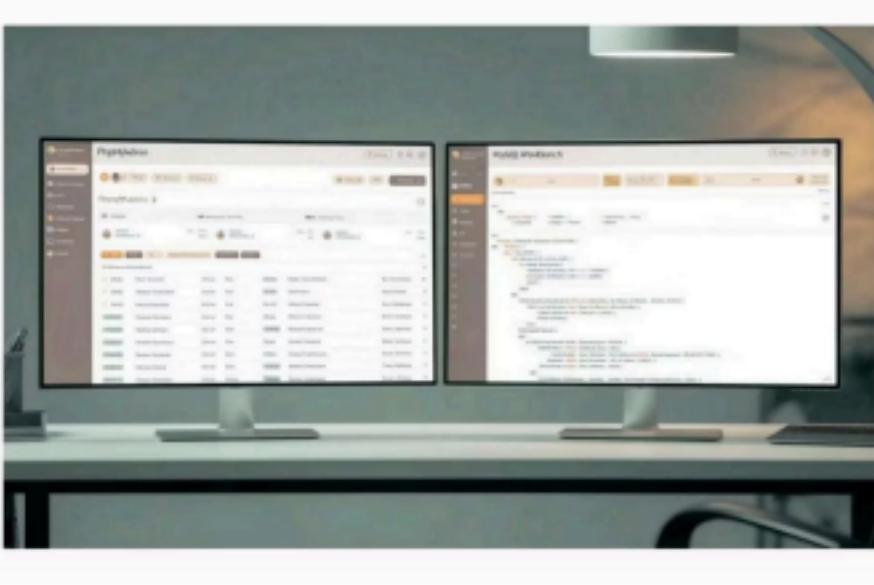
```
DELETE FROM usuarios  
WHERE inactivo = TRUE;
```

Estas cuatro operaciones son la **base de cualquier aplicación** que trabaje con datos persistentes. Dominarlas es esencial para convertirte en un desarrollador competente.

# Herramientas para trabajar con MySQL

**phpMyAdmin**  
**MySQL Workbench**





para  
Es una herramienta **gratuita con interfaz web** que viene incluida en paquetes como XAMPP o WAMP.

#### Con phpMyAdmin puedes:

- Crear y modificar bases de datos y tablas visualmente
- Ejecutar consultas SQL directamente desde el navegador
- Importar y exportar bases de datos completas

MySQL Workbench es una herramienta visual **más avanzada**

- Administrar usuarios y permisos fácilmente
- Ver relaciones entre tablas de forma gráfica

Es la herramienta que usaremos principalmente en este curso por su **facilidad de uso y disponibilidad inmediata**.

diseñar, administrar y documentar bases de datos MySQL de forma profesional.

#### Características principales:

Diseño visual de bases de datos con diagramas ER profesionales Editor SQL avanzado con autocompletado inteligente Monitoreo de rendimiento y optimización de consultas Gestión completa de usuarios y privilegios

Modelado visual de relaciones complejas

Es más potente que phpMyAdmin pero tiene una curva de aprendizaje mayor. Ideal para proyectos profesionales grandes.

# Instalación del entorno: XAMPP

01

## Descargar XAMPP

Ve a [apachefriends.org](http://apachefriends.org) y descarga la versión para tu sistema operativo (Windows, Mac o Linux). Es completamente gratuito y de código abierto.

03

## Iniciar los servicios

Abre el panel de control de XAMPP e inicia los servicios de **Apache** y **MySQL** haciendo clic en los botones "Start".

02

## Instalar XAMPP

Ejecuta el instalador y sigue las instrucciones. Asegúrate de incluir los componentes **Apache**, **MySQL** y **PHP** que son esenciales para el curso.

04

## Verificar la instalación

Abre tu navegador y visita: <http://localhost>. Deberías ver la página de bienvenida de XAMPP confirmando que todo funciona correctamente.

¡Ahora tienes todo lo necesario para empezar a trabajar con PHP y MySQL! Tu entorno de desarrollo local está listo para crear aplicaciones web profesionales.

## Accediendo a phpMyAdmin

Vamos a explorar phpMyAdmin, nuestra herramienta principal para gestionar bases de datos MySQL de manera visual e

intuitiva. [Abre phpMyAdmin](#)

### Inicia los servicios

Asegúrate de que Apache y MySQL están funcionando en el panel de control de XAMPP. Ambos botones deben estar en verde.

### Explora la interfaz

Verás un menú lateral con las bases de datos existentes y

opciones para crear nuevas. Familiarízate con el diseño.

En tu navegador, visita: <http://localhost/phpmyadmin>. Esta es la URL que siempre usarás para acceder.

Explora las pestañas como "Bases de datos", "SQL", "Estado", etc., para entender qué ofrece cada sección.

## Familiarízate con las pestañas

**Comprueba tu comprensión:** ¿Puedes encontrar dónde ejecutar una consulta SQL directamente en phpMyAdmin? Pista: busca la pestaña "SQL" en el menú superior.

# Creando nuestra primera base de datos

Vamos a crear nuestra primera base de datos MySQL usando phpMyAdmin. Este es un momento emocionante: ¡tu primer paso práctico!

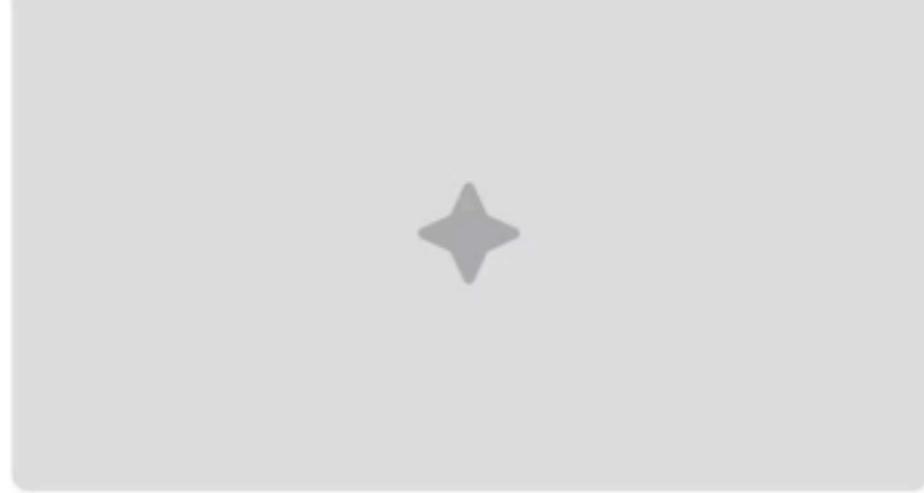
## Paso a paso:

Abre phpMyAdmin (<http://localhost/phpmyadmin>)1.

Haz clic en "**Nueva**" en el menú lateral izquierdo2.

3. Escribe "**mi\_primeras\_bd**" en el campo "Nombre de la base de datos"
4. Selecciona "**utf8mb4\_general\_ci**" como cotejamiento (permite caracteres especiales y emojis)
5. Haz clic en "**Crear**"

caracteres Unicode,



**¿Qué es el cotejamiento?** Define cómo se comparan y ordenan los caracteres. El cotejamiento "utf8mb4\_general\_ci" soporta todos los

### También con SQL:

```
incluidos emojis : CREATE DATABASE mi_primeras_bd CHARACTER SET utf8mb4  
COLLATE utf8mb4_general_ci;
```

## Creando nuestra primera tabla

Una tabla es donde realmente almacenamos los datos. Vamos a crear una tabla de usuarios en nuestra base de datos recién creada.

En el menú lateral de phpMyAdmin, haz clic en "mi\_primeras\_bd" para seleccionarla.

02

### Crear nueva tabla

Haz clic en "Nueva tabla". Escribe "usuarios" como nombre y "4" como número de columnas. Haz clic en "Continuar".

03

### Definir columnas

**id:** Tipo INT, AUTO\_INCREMENT, PRIMARY KEY

**nombre:** Tipo VARCHAR(100), NOT NULL

**email:** Tipo VARCHAR(100), NOT NULL, UNIQUE

**fecha\_registro:** Tipo TIMESTAMP, DEFAULT CURRENT\_TIMESTAMP

### Con SQL sería:

```
CREATE TABLE usuarios (
    id INT AUTO_INCREMENT PRIMARY KEY,
```

```
nombre VARCHAR(100) NOT NULL,  
email VARCHAR(100) NOT NULL UNIQUE,  
fecha_registro TIMESTAMP DEFAULT  
CURRENT_TIMESTAMP );
```

# Tipos de datos en MySQL

MySQL ofrece varios tipos de datos para almacenar diferentes tipos de información. Conocerlos es **fundamental** para diseñar bien tus tablas y optimizar el almacenamiento.

## Para texto

**CHAR(n)**: Texto de longitud fija (1-255 caracteres)

**VARCHAR(n)**: Texto de longitud variable (1-65,535 caracteres)

**TEXT**: Para textos largos (hasta 65,535 caracteres)

**LONGTEXT**: Para textos muy largos (hasta 4GB)

## Para fechas

**DATE**: Solo fecha (YYYY-MM-DD)

**TIME**: Solo hora (HH:MM:SS)

**DATETIME**: Fecha y hora completas

**TIMESTAMP**: Fecha y hora, útil para registrar cambios

## Para números

**INT**: Números enteros (-2,147,483,648 a 2,147,483,647)

**TINYINT**: Enteros pequeños (-128 a 127)

**DECIMAL(M,D)**: Números con decimales precisos

**FLOAT**: Números con decimales (precisión simple)

## Otros tipos

**BOOLEAN:** Valores verdadero/falso (1/0)

**ENUM:** Lista predefinida de valores posibles

**JSON:** Para almacenar datos en formato JSON

**BLOB:** Para datos binarios (imágenes, archivos)

Elegir el tipo correcto es importante para **optimizar el rendimiento** y el almacenamiento de tu base de datos.

## Claves y restricciones

Las claves y restricciones garantizan la **integridad y estructura** de tus datos, evitando errores y manteniendo la calidad de la información.

### Clave primaria (PRIMARY KEY)

Identifica de forma **única** cada registro en una tabla. No puede contener valores NULL y debe ser única.

id INT AUTO\_INCREMENT  
PRIMARY KEY

### Clave foránea (FOREIGN KEY)

Establece **relaciones entre tablas**, manteniendo la integridad referencial.

usuario\_id INT, FOREIGN KEY  
(usuario\_id) REFERENCES

### Restricción UNIQUE

Garantiza que todos los valores en una columna sean **diferentes**.

email VARCHAR(100) UNIQUE  
**Restricción NOT NULL**

Asegura que una columna **no pueda tener un valor vacío** (NULL).

nombre VARCHAR(100) NOT NULL

Estas restricciones ayudan a mantener la **calidad y consistencia** de tus datos, evitando errores comunes que podrían comprometer tu aplicación.

# Insertar datos en una tabla

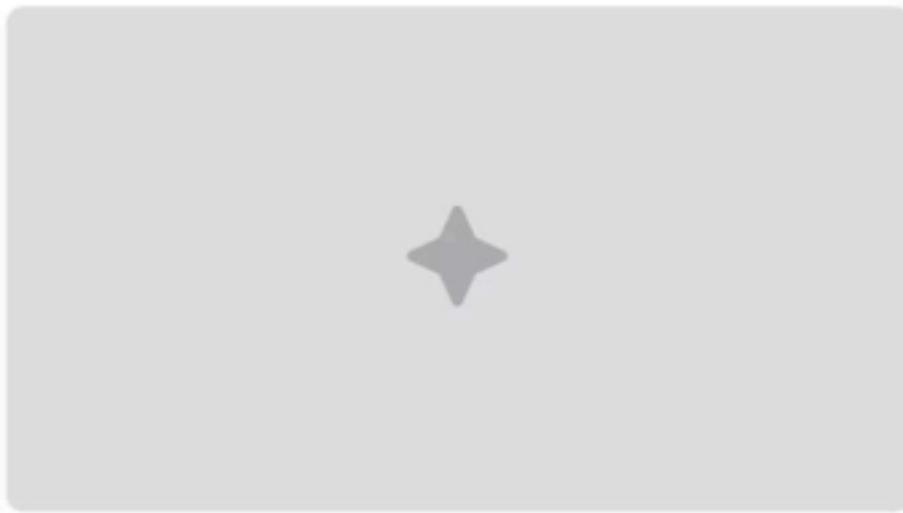
Una vez creada la tabla, necesitamos añadir información. Vamos a insertar algunos usuarios en nuestra tabla usando diferentes métodos.

## Usando phpMyAdmin:

Selecciona la tabla "usuarios" en el menú lateral1.

2. Haz clic en la pestaña "**Insertar**"

Rellena los campos (no es necesario llenar el ID ni la fecha)3. 4. Haz clic en "**Continuar**"



## Usando SQL (más común en PHP):

```
INSERT INTO usuarios  
(nombre, email)  
VALUES
```

```
('Juan Pérez', 'juan@ejemplo.com');
```

```
INSERT INTO usuarios  
(nombre, email)  
VALUES  
('María García', 'maria@ejemplo.com');
```

**Importante:** No necesitamos incluir el **id** porque es **AUTO\_INCREMENT** (se genera automáticamente) ni la **fecha\_registro** porque tiene un valor predeterminado.

**Comprueba tu comprensión:** ¿Qué pasaría si intentamos insertar dos usuarios con el mismo email? La restricción **UNIQUE** lo impedirá y mostrará un error.

## Consultar datos con SELECT

Para recuperar información de nuestra base de datos usamos el comando **SELECT**. Es probablemente el comando más usado en SQL.

### Consulta básica

```
SELECT * FROM usuarios;
```

Devuelve **todas las columnas** (\*) de todos los registros en la tabla usuarios.

## Con condición

```
SELECT * FROM usuarios  
WHERE id = 1;
```

Solo devuelve el usuario con **ID igual a 1**.

## Con límite

```
SELECT * FROM usuarios  
LIMIT 5;
```

Devuelve solo los **primeros 5 usuarios**.

## Columnas específicas

```
SELECT nombre, email  
FROM usuarios;
```

Solo muestra las columnas **nombre** y **email**, optimizando la consulta.

## Con ordenamiento

```
SELECT * FROM usuarios  
ORDER BY nombre ASC;
```

Devuelve todos los usuarios **ordenados alfabéticamente** por nombre.

## Consulta combinada

```
SELECT nombre, email  
FROM usuarios  
WHERE id > 2  
ORDER BY nombre DESC  
LIMIT 10;
```

Combina múltiples condiciones para consultas **más específicas**.

En phpMyAdmin, puedes escribir estas consultas en la pestaña "**SQL**" y hacer clic en "Ejecutar" para ver los resultados inmediatamente.

## Actualizar datos con UPDATE

Para modificar información existente usamos el comando **UPDATE**. Es fundamental para mantener los datos actualizados. **Sintaxis básica:**

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

**Importante:** Siempre incluye una condición **WHERE**, o actualizarás **TODOS** los registros de la tabla. Esto puede ser catastrófico.

## Ejemplo 1: Actualizar un usuario específico

```
UPDATE usuarios  
SET nombre = 'Juan López'  
WHERE id = 1;
```

Actualiza solo el usuario con ID 1.

**CUIDADO:** Si omites el WHERE, modificarás **TODOS los registros**. Siempre verifica tus consultas UPDATE antes de ejecutarlas.

## Eliminar datos con DELETE

Para borrar registros de una tabla utilizamos el comando **DELETE**. Esta operación es **permanente** y requiere precaución. **Sintaxis**

**básica:**

## Ejemplo 2: Actualizar múltiples campos

```
UPDATE usuarios  
SET nombre = 'Ana Martínez',  
email = 'ana.nueva@ejemplo.com'  
WHERE id = 3;
```

Modifica varios campos a la vez.

## Ejemplo 3: Actualizar múltiples registros

```
UPDATE usuarios  
SET activo = 0  
WHERE last_login < '2023-01-01';
```

Desactiva todos los usuarios que no han iniciado sesión desde 2023.

```
DELETE FROM nombre_tabla WHERE condición;
```

**ADVERTENCIA:** Al igual que con UPDATE, siempre incluye una condición **WHERE**, o eliminarás **TODOS** los registros de la tabla.

## Ejemplo 1: Eliminar un usuario específico

```
DELETE FROM usuarios  
WHERE id = 5;
```

Elimina únicamente el usuario con ID

5.



## Ejemplo 2: Eliminar múltiples registros

```
DELETE FROM usuarios  
WHERE fecha_registro < '2022-01-01';
```

Elimina todos los usuarios registrados antes de 2022.

## Eliminar todos los registros (¡Cuidado!)

```
DELETE FROM usuarios;
```

Esta consulta elimina **TODOS** los registros de la tabla (la tabla sigue existiendo, pero vacía).

**TIP Profesional:** Antes de ejecutar un DELETE, es buena práctica hacer primero un SELECT con la misma condición para verificar qué registros se eliminarán.

# Conexión entre PHP y MySQL

Ahora que conocemos lo básico de MySQL, vamos a ver cómo **conectar nuestras aplicaciones PHP** con la base de datos. Esta es la pieza clave que une el backend con los datos.

## ¿Qué es una conexión a base de datos?

Es un **enlace** entre tu script PHP y el servidor MySQL que permite enviar consultas y recibir resultados de manera bidireccional.

## ¿Para qué sirve?

Para que tu aplicación pueda leer y escribir **datos persistentes** que se mantienen incluso cuando el usuario cierra la página o reinicia el servidor.

## ¿Cómo se hace?

PHP ofrece dos extensiones principales para conectar con MySQL: **MySQLi** y **PDO**. Veremos ambas en detalle para que elijas la que mejor se adapte a tus necesidades.

# MySQLi vs PDO: ¿Cuál elegir?

PHP ofrece dos formas principales de conectarse a MySQL: **MySQLi** y **PDO**. Cada una tiene sus ventajas específicas.

## PDO (PHP Data Objects)

### MySQLi (MySQL Improved)

- ✓ Específico para bases de datos **MySQL** ✓
- Soporta procedimientos almacenados ✓ Ofrece estilo orientado a objetos y procedural ✓ Más simple para proyectos solo con MySQL ✓ Mejor rendimiento en algunas operaciones

```
$conexion = new mysqli(  
    "localhost",  
    "usuario",  
    "contraseña",  
    "basededatos"  
)
```

- ✓ Compatible con **12+ tipos** de bases de datos ✓ Solo estilo orientado a objetos
- ✓ Manejo de errores mediante excepciones ✓ Mayor portabilidad entre bases de datos ✓ Sentencias preparadas más intuitivas

```
$conexion = new PDO(  
    "mysql:host=localhost;dbname=bd",  
    "usuario",  
    "contraseña"  
)
```

Ambas opciones son **seguras y eficientes**. En este curso veremos las dos, pero nos centraremos más en **MySQLi** por su simplicidad para

principiantes.

# Primera conexión con MySQLi (Orientado a Objetos)

Vamos a crear nuestro primer script PHP para conectar con MySQL usando MySQLi en estilo orientado a objetos. **Paso 1: Crear archivo conexión.php**

```
<?php  
// Parámetros de conexión  
$servidor = "localhost";  
$usuario = "root"; // Usuario predeterminado en XAMPP  
$password = ""; // Sin contraseña en XAMPP  
$basedatos = "mi_primera_bd";  
  
// Crear conexión  
$conexion = new mysqli(  
    $servidor,  
    $usuario,  
    $password,
```

```
$basedatos  
);  
  
// Verificar conexión  
if ($conexion->connect_error) {  
    die("Error de conexión: " .  
        $conexion->connect_error);  
}  
  
echo "Conexión exitosa a la base de datos";  
  
// No cerramos la conexión aquí para usarla  
// en otros scripts  
?>
```

01

## Guardar y probar

Guarda este archivo como "conexion.php" en la carpeta htdocs de XAMPP (por ejemplo, en C:\xampp\htdocs\miproyecto)

### ¿Qué hace cada parte?

- new mysqli(): Crea un objeto de conexión a la base de datos
- \$conexion->connect\_error: Verifica si hay un error de conexión
- die(): Detiene la ejecución del script y muestra un mensaje

02

**Abrir en navegador**

Visita: <http://localhost/miproyecto/conexion.php>

**TIP:** En producción, nunca muestres los errores específicos al usuario. En su lugar, regístralos en un archivo de log.

## Conexión con MySQLi (Estilo Procedural)

Algunos desarrolladores

prefieren el estilo procedural. Veamos cómo es la misma conexión usando este enfoque alternativo.

**Código de conexión estilo procedural** Principales diferencias con el estilo orientado a objetos:

```
<?php  
// Parámetros de conexión $servidor = "localhost";  
$usuario = "root";  
$password = "";
```

```
$basedatos = "mi_primera_bd";
```

```
// Crear conexión
```

```
$conexion = mysqli_connect( $servidor,  
$usuario,  
$password,  
$basedatos  
);
```

```
// Verificar conexión
```

```
if (!$conexion) {  
die("Error de conexión: " . mysqli_connect_error()); }
```

```
echo "Conexión exitosa"; ?>
```

Usamos `mysqli_connect()` en lugar de `new mysqli()` Verificamos el error con `!$conexion` y `mysqli_connect_error()`

Todas las funciones comienzan con `mysqli_` y reciben la conexión como primer parámetro

**¿Cuál estilo elegir?** El estilo orientado a objetos es más moderno y recomendado para nuevos proyectos. El estilo procedural puede ser más familiar si vienes de versiones antiguas de PHP.

Por coherencia, usaremos principalmente el **estilo orientado a objetos** en este curso.

## Conexión con PDO

PDO ofrece mayor **flexibilidad y portabilidad** entre diferentes tipos de bases de datos. Veamos cómo implementar una conexión segura.

```
<?php
// Parámetros de conexión
$servidor = "localhost";
$usuario = "root";
$password = "";
$basedatos = "mi_primera_bd";

try {
    // Crear conexión PDO
    $pdo = new PDO(
        "mysql:host=$servidor;dbname=$basedatos;charset=utf8mb4",
        $usuario,
        $password,
        [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION]
```

```
);

echo "Conexión exitosa con PDO";

} catch(PDOException $e) {
    // Capturar y manejar excepciones
    echo "Error de conexión: " . $e->getMessage();
}

?>
```

## Características importantes de PDO

Usa bloques **try-catch** para manejar errores de forma estructurada

El DSN (Data Source Name) especifica la conexión:  
"mysql:host=servidor;dbname=basedatos"

Podemos configurar opciones como  
PDO::ATTR\_ERRMODE para determinar cómo manejar errores

PDO::ERRMODE\_EXCEPTION hace que PDO lance excepciones cuando ocurren errores

## Ventajas de PDO

Cambiar a otra base de datos (PostgreSQL, SQLite) solo requiere modificar el DSN

Manejo de errores más estructurado mediante excepciones  
Sentencias preparadas más intuitivas con marcadores de posición con nombre

# Configuración en archivos separados

Es una **buenas prácticas profesionales** separar la configuración de la base de datos en archivos independientes. Esto mejora la seguridad y facilita el mantenimiento.

**1**

## **config.php**

Archivo de configuración con constantes

### **1. config.php**

```
<?php
// Configuración de la BD
define('DB_HOST', 'localhost');
define('DB_USER', 'root');
define('DB_PASS', '');
define('DB_NAME', 'mi_primera_bd'); define('DB_CHARSET', 'utf8mb4');
?>
```

**2**

## conexion.php

Funciones de conexión

### 2. conexion.php

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

function getConexionMysqli() { $conn = new mysqli(
    DB_HOST, DB_USER,
    DB_PASS, DB_NAME
);

if ($conn->connect_error) {
    die("Error: " .
    $conn->connect_error);
}

$conn->set_charset(DB_CHARSET); return $conn;
```

```
}
```

```
?>
```

**3**

## Uso en scripts

Incluir y usar

### 3. uso.php

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

```
// Obtener conexión  
$conexion =  
getConexionMysqli();
```

```
// Usar la conexión...  
?>
```

Esta estructura mantiene tu código **organizado, seguro y fácil de mantener.**

## CRUD Básico: CREATE - Insertar registros

Vamos a implementar la

primera operación CRUD: **CREATE** (Crear) registros en la base de datos desde PHP de forma segura.

### Con MySQLi (Orientado a Objetos)

```
<?php  
require_once 'conexion.php';  
$conexion = getConexionMysqli();
```

```
// Datos a insertar
$nombre = "Carlos Rodríguez";
$email = "carlos@ejemplo.com";
```

```
// Preparar sentencia SQL
$sql = "INSERT INTO usuarios
(nombre, email)
VALUES (?, ?);
```

```
// Crear sentencia preparada
$stmt = $conexion->prepare($sql);
```

```
// Vincular parámetros (s=string)
$stmt->bind_param("ss",
$nombre, $email);
```

```
// Ejecutar la sentencia
if ($stmt->execute()) {
echo "Registro creado";
```

```
// Obtener el ID insertado
```

```
$id_insertado = $stmt->insert_id;  
echo "ID: " . $id_insertado;  
} else {  
    echo "Error: " . $stmt->error;  
}  
  
$stmt->close();
```

```
$conexion->close();
```

```
?>
```

## Con PDO

```
<?php  
require_once 'conexion.php'; $pdo = getConexionPDO();  
  
// Datos a insertar  
$nombre = "Carlos Rodríguez"; $email = "carlos@ejemplo.com";  
  
try {  
    // Preparar sentencia SQL $sql = "INSERT INTO usuarios (nombre, email)  
    VALUES (:nombre, :email)";  
    // Crear sentencia preparada $stmt = $pdo->prepare($sql);  
    // Vincular parámetros
```

```
$stmt->bindParam(':nombre', $nombre);
$stmt->bindParam(':email', $email);

// Ejecutar
$stmt->execute();

// Obtener ID insertado
$id = $pdo->lastInsertId(); echo "Registro creado con ID: " . $id;

} catch(PDOException $e) { echo "Error: " .
$e->getMessage();
}

?>
```

**Importante:** Usamos **sentencias preparadas** con marcadores (?) o (:nombre) para prevenir inyecciones SQL. Esta es la forma más segura de insertar datos.

## CRUD Básico: READ - Consultar registros

Ahora implementaremos la

operación **READ** para consultar y mostrar datos desde PHP de forma estructurada.

## Con MySQLi (Orientado a Objetos)

```
<?php
require_once 'conexion.php';
$conexion = getConexionMysqli();

// Consultar todos los registros
$sql = "SELECT id, nombre, email
FROM usuarios";
$resultado = $conexion->query($sql);

// Verificar si hay resultados
if ($resultado->num_rows > 0) {
    // Iterar sobre los resultados
    echo "<table border='1'>";
    echo "<tr><th>ID</th>
<th>Nombre</th>
<th>Email</th></tr>";

    while ($fila =
        $resultado->fetch_assoc()) {
        echo "<tr>";
```

```
echo "<td>" . $fila["id"] .  
"</td>";  
echo "<td>" .  
$fila["nombre"] . "</td>";  
echo "<td>" . $fila["email"]  
. "</td>";  
echo "</tr>";  
}  
echo "</table>";  
} else {  
echo "No se encontraron  
registros";  
}
```

```
$resultado->free();  
$conexion->close();
```

```
?>
```

## Con PDO

```
<?php  
require_once 'conexion.php'; $pdo = getConexionPDO();
```

```
try {
    // Consultar todos los registros $sql = "SELECT id, nombre, email  FROM usuarios";
    $stmt = $pdo->query($sql);
    // Mostrar resultados
    echo "<table border='1'>";
    echo "<tr><th>ID</th>
<th>Nombre</th>
<th>Email</th></tr>";

    // Iterar sobre resultados
    while ($fila = $stmt->fetch()) { echo "<tr>";
        echo "<td>" . $fila["id"] .
        "</td>";
        echo "<td>" .
        $fila["nombre"] . "</td>"; echo "<td>" .
        $fila["email"] . "</td>";
        echo "</tr>";
    }
    echo "</table>";

} catch(PDOException $e) {
```

```
echo "Error: " .  
$e->getMessage();  
}  
?>
```

**TIP:** PDO ofrece modos de obtención como PDO::FETCH\_ASSOC (array asociativo), PDO::FETCH\_OBJ (objeto) o PDO::FETCH\_NUM (array numérico).

## Consultar un registro específico

A menudo necesitamos consultar un registro específico por su ID u otra condición. Siempre usa **sentencias preparadas** para esto.

### Con MySQLi

```
<?php  
require_once 'conexion.php'; $conexion = getConexionMysqli();
```

```
// ID a buscar
$id = 1;

// Preparar consulta
$sql = "SELECT * FROM usuarios WHERE id = ?";
$stmt = $conexion->prepare($sql);

// i = integer
$stmt->bind_param("i", $id); $stmt->execute();
$resultado = $stmt->get_result();

if ($resultado->num_rows > 0) { $usuario =
    $resultado->fetch_assoc(); echo "Usuario encontrado: " . $usuario["nombre"];
    echo "<br>Email: " .
    $usuario["email"];
} else {
    echo "Usuario no encontrado"; }

$stmt->close();
$conexion->close();
?>
```

## Con PDO

```
<?php
require_once 'conexion.php'; $pdo = getConexionPDO();

// ID a buscar
$id = 1;

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

    // Verificar si existe
    if ($usuario = $stmt->fetch()) { echo "Usuario encontrado: " . $usuario["nombre"];
        echo "<br>Email: " .
        $usuario["email"];
    } else {
        echo "Usuario no encontrado"; }

} catch(PDOException $e) {
```

```
echo "Error: " .  
$e->getMessage();  
}  
?>
```

**Seguridad:** Siempre usa sentencias preparadas cuando incluyas variables en tus consultas SQL, especialmente si provienen de entradas de usuario.

## CRUD Básico: UPDATE - Actualizar registros

Ahora veremos

cómo **actualizar registros existentes** desde PHP de forma segura usando sentencias preparadas.

### Con MySQLi

```
<?php  
require_once 'conexion.php'; $conexion = getConexionMysqli();  
  
// Datos para actualizar  
$id = 1;  
$nuevoNombre = "Carlos Pérez"; $nuevoEmail =
```

```
"carlos.perez@ejemplo.com";  
  
// Preparar sentencia SQL  
$sql = "UPDATE usuarios  
SET nombre = ?, email = ? WHERE id = ?";  
  
// Crear sentencia preparada $stmt = $conexion->prepare($sql);  
  
// s=string, i=integer  
$stmt->bind_param("ssi",  
$nuevoNombre,  
$nuevoEmail, $id);  
  
// Ejecutar  
if ($stmt->execute()) {  
    if ($stmt->affected_rows > 0) { echo "Registro actualizado"; } else {  
        echo "No se encontró el ID: " . $id;  
    }  
}  
} else {  
    echo "Error: " . $stmt->error; }  
  
$stmt->close();
```

```
$conexion->close();
?>
Con PDO

<?php
require_once 'conexion.php'; $pdo = getConexionPDO();

// Datos para actualizar
$id = 1;
$nuevoNombre = "Carlos Pérez"; $nuevoEmail =
"carlos.perez@ejemplo.com";

try {
// Preparar sentencia SQL $sql = "UPDATE usuarios
SET nombre = :nombre,
email = :email
WHERE id = :id";

// Crear sentencia preparada $stmt = $pdo->prepare($sql);

// Vincular parámetros
$stmt->bindParam(':nombre', $nuevoNombre);
```

```
$stmt->bindParam(':email', $nuevoEmail);
$stmt->bindParam(':id', $id, PDO::PARAM_INT);

// Ejecutar
$stmt->execute();

if ($stmt->rowCount() > 0) { echo "Registro actualizado"; } else {
echo "No se encontró el ID: " . $id;
}
} catch(PDOException $e) { echo "Error: " .
$e->getMessage();
}
?>
```

## CRUD Básico: DELETE - Eliminar registros

Por último,

implementaremos la operación **DELETE** para eliminar registros desde PHP. Recuerda que esta operación es **permanente**.

### Con MySQLi

```
<?php
require_once 'conexion.php'; $conexion = getConexionMysqli();
```

```
// ID del registro a eliminar
$id = 3;

// Preparar sentencia SQL
$sql = "DELETE FROM usuarios WHERE id = ?";

// Crear sentencia preparada $stmt = $conexion->prepare($sql);

// Vincular parámetros
$stmt->bind_param("i", $id);

// Ejecutar
if ($stmt->execute()) {
    if ($stmt->affected_rows > 0) { echo "Registro eliminado"; } else {
        echo "No se encontró el ID: " . $id;
    }
} else {
    echo "Error: " . $stmt->error;
}

$stmt->close();
$conexion->close();
```

?>

## Con PDO

```
<?php
require_once 'conexion.php'; $pdo = getConexionPDO();

// ID del registro a eliminar $id = 3;

try {
    // Preparar sentencia SQL $sql = "DELETE FROM usuarios WHERE id = :id";

    // Crear sentencia preparada $stmt = $pdo->prepare($sql);
    // Vincular parámetros
    $stmt->bindParam(':id', $id, PDO::PARAM_INT);

    // Ejecutar
    $stmt->execute();

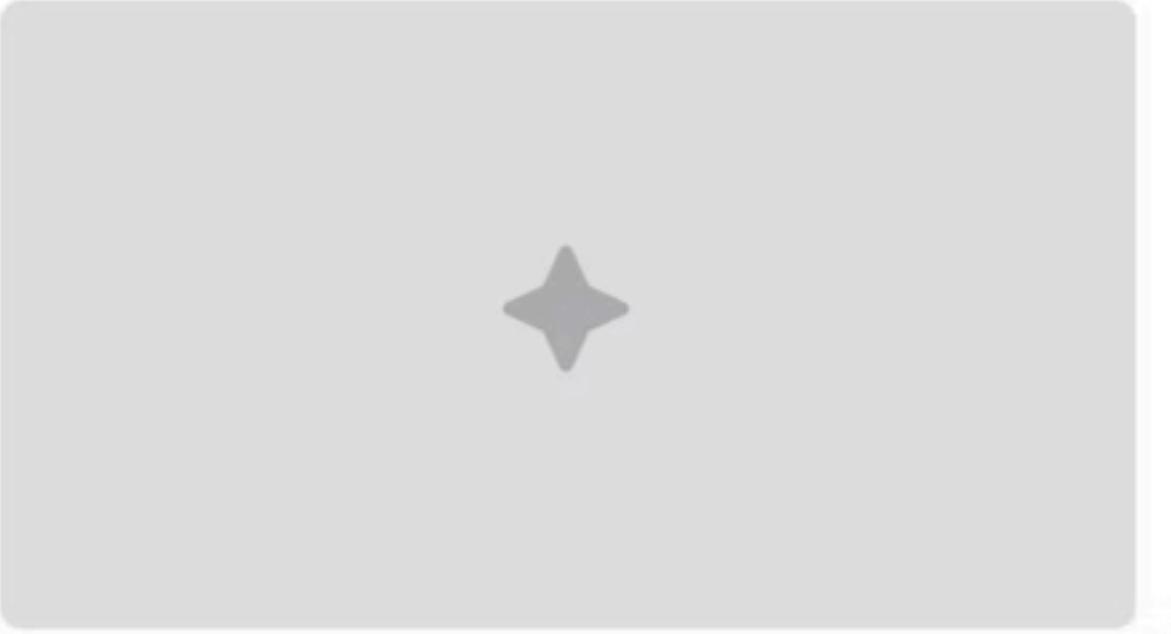
    if ($stmt->rowCount() > 0) { echo "Registro eliminado"; } else {
        echo "No se encontró el ID: " . $id;
    }
}
```

```
 } catch(PDOException $e) { echo "Error: " .  
 $e->getMessage();  
}  
?>
```

▲ **CUIDADO:** La eliminación es permanente. Considera implementar una "**eliminación lógica**" (marcar como inactivo) en lugar de eliminar físicamente los registros importantes.

## Seguridad: ¿Qué es la inyección SQL?

La inyección SQL es una de las **vulnerabilidades más comunes y peligrosas** en aplicaciones web. Entenderla es crucial para crear aplicaciones seguras.



## ¿Qué es?

Es una técnica de ataque donde un usuario malintencionado **inserta código SQL malicioso** en las entradas de la aplicación, alterando las consultas que se envían a la base de datos.

## Consulta resultante

```
SELECT * FROM usuarios  
WHERE nombre = ''  
OR 1=1 --'
```

Esto devolvería **TODOS** los usuarios porque 1=1 siempre es verdadero, y -- comenta el resto.

### Ejemplo vulnerable

```
$usuario = $_POST["usuario"];  
$sql = "SELECT * FROM usuarios  
WHERE nombre = '$usuario';"
```

```
$resultado =  
$conexion->query($sql);
```

Si un atacante escribe: ' OR 1=1 --

### Consecuencias posibles

Acceso no autorizado a datos sensibles Eludir autenticación (login sin contraseña) Modificación o eliminación masiva de datos Ejecución de comandos en el servidor Comprometer todo el sistema

## Prevención de inyecciones SQL

La prevención de inyecciones SQL es **crucial** para la seguridad de cualquier aplicación que use bases de datos. Aquí están las mejores prácticas.

### 1. Utilizar sentencias preparadas

Las sentencias preparadas **separan el SQL de los datos**, evitando que los parámetros sean interpretados como código SQL.

```
$stmt = $conexion->prepare(  
    "SELECT * FROM usuarios  
    WHERE nombre = ?"  
);  
$stmt->bind_param("s",  
    $_POST["usuario"]);  
$stmt->execute();
```

### 3. Validar y filtrar entradas

Valida tipo y formato de los datos antes de procesarlos.

```
// Validar que un ID sea numérico  
if (!is_numeric($_POST["id"])) {  
    die("ID no válido");  
}
```

### 2. Escapar datos con funciones específicas

Si no puedes usar sentencias preparadas, utiliza funciones para escapar caracteres especiales.

```
$usuario =
```

```
$conexion->real_escape_string  
$_POST["usuario"]  
);  
$sql = "SELECT * FROM usuarios  
WHERE nombre = '$usuario';
```

## 4. Principio de mínimo privilegio

Utiliza cuentas de base de datos con **permisos limitados** para cada operación.

```
// Usuario con permisos limitados  
$conexionLectura = new mysqli(  
"localhost",  
"usuario_lectura",  
"clave", "db"  
);
```

**IMPORTANTE:** Las [sentencias preparadas](#) son la mejor defensa contra inyecciones SQL. Úsalas **siempre** que sea posible.

# Sentencias preparadas en detalle

Las sentencias preparadas son fundamentales para la seguridad. Entender cómo funcionan te ayudará a usarlas correctamente.

## Ejecución

## Preparación

Se envía la **estructura de la consulta** con marcadores de posición a la base de datos, que la compila y optimiza.

Se envían los **valores** para los marcadores por separado y la base de datos los inserta de forma segura.

La clave es que los datos **nunca se interpretan como parte del código SQL**, sino como valores puros.

"**s**" string (cadena de texto) "**i**" integer (número entero) "**d**" double

## Marcadores en MySQLi

(número decimal)

más legible:

## Marcadores nombrados en PDO

PDO permite usar marcadores con nombre, lo que hace el código

```
"b" blob (datos binarios) $stmt = $pdo->prepare( "INSERT INTO productos
$stmt =
$stmt =
$conn->prepare(
"INSERT INTO productos
(nombre, precio, cantidad)
VALUES (?, ?, ?)"
);
$stmt->bind_param("sdi",
$nombre, $precio, $cantidad);

VALUES (:nombre, :precio,
(nombre, precio, cantidad)
:nombre" );
$stmt->execute([
':nombre' => $nombre,
':precio' => $precio,
':cantidad' => $cantidad
]);
```

## Gestión segura de contraseñas

Almacenar contraseñas de forma segura es **esencial** para proteger las cuentas de usuario. Nunca guardes contraseñas en texto plano.

## **NUNCA hagas esto**

```
// NUNCA ALMACENES ASÍ
$clave = $_POST['clave'];
$sql = "INSERT INTO usuarios
(clave)
VALUES ('$clave');
```

Si la base de datos se ve comprometida, **todas las cuentas quedan expuestas.**

## **✓ Usar password\_hash() para encriptar**

```
// Crear hash seguro
$clave = $_POST['clave'];
$hash = password_hash(
$clave,
PASSWORD_DEFAULT
);

$sql = "INSERT INTO usuarios
(clave) VALUES (?)";
```

```
$stmt = $conexion->prepare($sql); $stmt->bind_param("s", $hash);
```

## ✓ Usar password\_verify() para verificar

```
// Verificar contraseña
```

```
$clave = $_POST["clave"];
```

```
$sql = "SELECT clave FROM usuarios WHERE correo = ?";
```

```
$stmt = $conexion->prepare($sql); $stmt->bind_param("s",
$_POST["correo"]);
```

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

```
$resultado = $stmt->get_result(); $usuario =
```

```
$resultado->fetch_assoc();
```

```
if (password_verify($clave,
```

```
$usuario["clave"])) {
```

```
echo "Contraseña correcta";
```

```
} else {
```

```
echo "Contraseña incorrecta"; }
```

**Ventajas de password\_hash():** Genera automáticamente una "sal" (salt) única, utiliza algoritmos seguros (actualmente bcrypt), se

actualiza automáticamente a algoritmos más seguros, y es resistente a ataques de fuerza bruta.

# Validación y sanitización de datos

Validar y sanitizar datos de entrada es una práctica esencial para la **seguridad y calidad** de los datos en tu aplicación.

## Sanitización

Es el proceso de **limpiar** los datos de entrada para eliminar caracteres potencialmente peligrosos o no deseados.

```
// Sanitizar datos
$nombre = htmlspecialchars(
    trim($_POST["nombre"])
);

$correo = filter_var(
    $_POST["correo"],
    FILTER_SANITIZE_EMAIL
);

$url = filter_var(
    $_POST["sitio"],
```

```
FILTER_SANITIZE_URL  
);
```

## Funciones útiles

`filter_var()` - Valida o sanitiza según el filtro

`is_numeric()` - Comprueba si es un número

`preg_match()` - Validaciones con expresiones regulares `trim()` - Elimina espacios al inicio y final

`htmlspecialchars()` - Convierte caracteres especiales

## Validación

Es el proceso de **verificar** que los datos cumplen con ciertos criterios o reglas antes de procesarlos.

```
// Validar datos  
if (!filter_var($correo,  
FILTER_VALIDATE_EMAIL)) {  
echo "Email no válido";  
}
```

```
if (!filter_var($url,
```

```
FILTER_VALIDATE_URL)) {  
    echo "URL no válida";  
}  
  
if (!is_numeric($_POST["edad"])  
    || $_POST["edad"] < 18) {  
    echo "Edad no válida";  
}
```

## Validación con expresiones regulares

```
// Solo letras y espacios  
if (!preg_match("/^([a-zA-Z ])*$/",
$nombre)) {  
    echo "Solo letras y espacios";  
}  
  
// Contraseña fuerte  
if (!preg_match(
"/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)
```

```
[a-zA-Z\d]{8,}$/",
```

```
$clave) {
```

```
echo "Contraseña débil";
```

```
}
```

## Ejemplo práctico: Formulario de registro

Vamos a crear un **formulario de registro completo** con validación y procesamiento seguro. Este ejemplo integra todo lo aprendido.

### HTML del formulario (registro.html)

```
<!DOCTYPE html>
<html>
<head>
<title>Registro de Usuario</title>
</head>
<body>
<h2>Crear una cuenta</h2>
<form action="registrar.php" method="post">
<div>
<label for="nombre">Nombre:</label>
```

```
<input type="text" name="nombre" id="nombre" required>
</div>
<div>
<label for="email">Correo electrónico:</label>
<input type="email" name="email" id="email" required>
</div>
<div>
<label for="clave">Contraseña:</label>
<input type="password" name="clave" id="clave"
required minlength="8">
<small>Mínimo 8 caracteres, incluyendo mayúsculas,
minúsculas y números</small>
</div>
<div>
<label for="clave2">Confirmar contraseña:</label>
<input type="password" name="clave2" id="clave2" required>
</div>
<button type="submit">Registrarse</button>
</form>
<p>¿Ya tienes cuenta? <a href="login.php">Iniciar sesión</a></p>
</body>
</html>
```

# Procesamiento del formulario de registro

Ahora implementaremos el procesamiento PHP para el formulario de registro, con **validación completa** y seguridad.

```
<?php
require_once 'conexion.php';
$conexion = getConexionMysqli();

// Inicializar array de errores
$errores = [];

// Procesar formulario enviado
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Sanitizar entradas
    $nombre = htmlspecialchars(trim($_POST['nombre']));
    $email = filter_var($_POST['email'], FILTER_SANITIZE_EMAIL);
    $clave = $_POST['clave'];
    $clave2 = $_POST['clave2'];

    // Validar nombre
    if (empty($nombre)) {
```

```
$errores[] = "El nombre es obligatorio";
} elseif (strlen($nombre) < 2) {
$errores[] = "El nombre debe tener al menos 2 caracteres";
}

// Validar email
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
$errores[] = "Email no válido";
}

// Verificar si el email ya existe
$stmt = $conexion->prepare("SELECT id FROM usuarios WHERE email = ?");
$stmt->bind_param("s", $email);
$stmt->execute();
$stmt->store_result();
if ($stmt->num_rows > 0) {
$errores[] = "Este email ya está registrado";
}
$stmt->close();

// Validar contraseña
if (strlen($clave) < 8) {
```

```
$errores[] = "La contraseña debe tener al menos 8 caracteres";
} elseif (!preg_match("/[A-Z]/", $clave)) {
$errores[] = "La contraseña debe incluir al menos una letra mayúscula";
} elseif (!preg_match("/[a-z]/", $clave)) {
$errores[] = "La contraseña debe incluir al menos una letra minúscula";
} elseif (!preg_match("/[0-9]/", $clave)) {
$errores[] = "La contraseña debe incluir al menos un número";
}

// Verificar que las contraseñas coincidan
if ($clave !== $clave2) {
$errores[] = "Las contraseñas no coinciden";
}

// Si no hay errores, proceder con el registro
if (empty($errores)) {
// Hashear la contraseña
$hash = password_hash($clave, PASSWORD_DEFAULT);

// Insertar el nuevo usuario
$stmt = $conexion->prepare(
"INSERT INTO usuarios (nombre, email, clave) VALUES (?, ?, ?)"
```

```
);

$stmt->bind_param("sss", $nombre, $email, $hash);

if ($stmt->execute()) {
    // Registro exitoso
    $usuario_id = $stmt->insert_id;

    // Iniciar sesión automáticamente
    session_start();
    $_SESSION['usuario_id'] = $usuario_id;
    $_SESSION['usuario_nombre'] = $nombre;

    // Redireccionar
    header("Location: perfil.php");
    exit();
} else {
    $errores[] = "Error al registrar usuario: " . $stmt->error;
}
$stmt->close();
}
```

```
// Si hay errores, mostrarlos
if (!empty($errores)) {
    echo "<div class='errores'>";
    foreach ($errores as $error) {
        echo "<p>" . $error . "</p>";
    }
    echo "</div>";
    echo "<p><a href='javascript:history.back()>Volver</a></p>";
}
$conexion->close();
?>
```

## Ejemplo práctico: Sistema de inicio de sesión

Ahora crearemos

un **sistema completo de inicio de sesión** con verificación de credenciales, manejo de sesiones y seguridad avanzada. **HTML del**

### formulario (login.html)

```
<!DOCTYPE html>
```



```
<html>
<head>
<title>Iniciar Sesión</title>
</head>
<body>
<h2>Iniciar Sesión</h2>
<form action="login.php"
method="post">
<div>
<label for="email">
Correo electrónico:
</label>
<input type="email"
name="email"
id="email"
required>
</div>
<div>
<label for="clave">
Contraseña:
</label>
<input type="password"
```

```
name="clave"
id="clave"
required>
</div>
<div>
<label>
<input type="checkbox"
name="recordar"
value="1">
Recordarme
</label>
</div>
<button type="submit">
Iniciar Sesión
</button>
</form>
<p>
<a href="recuperar.php">
¿Olvidaste tu contraseña?
</a>
</p>
<p>
```

```
¿No tienes cuenta?  
<a href="registro.php">  
Regístrate  
</a>  
</p>  
</body>  
</html>
```

## Procesamiento del inicio de sesión

Implementaremos el procesamiento del login con **verificación segura de contraseñas**, manejo de intentos fallidos y prevención de ataques de fuerza bruta.

```
<?php  
// Iniciar sesión  
session_start();  
  
require_once 'conexion.php';  
$conexion = getConexionMysqli();  
  
$error = ";
```

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {  
    // Sanitizar entradas  
    $email = filter_var($_POST['email'], FILTER_SANITIZE_EMAIL);  
    $clave = $_POST['clave'];  
    $recordar = isset($_POST['recordar']) ? true : false;  
  
    // Validar campos  
    if (empty($email) || empty($clave)) {  
        $error = "Por favor, completa todos los campos";  
    } else {  
        // Buscar usuario por email  
        $stmt = $conexion->prepare(  
            "SELECT id, nombre, email, clave, estado, intentos_fallidos  
            FROM usuarios WHERE email = ?"  
        );  
        $stmt->bind_param("s", $email);  
        $stmt->execute();  
        $resultado = $stmt->get_result();  
  
        if ($resultado->num_rows === 1) {  
            $usuario = $resultado->fetch_assoc();
```

```
// Verificar estado
if ($usuario['estado'] != 'activo') {
    $error = "Esta cuenta ha sido suspendida o desactivada";
}

// Verificar intentos fallidos
elseif ($usuario['intentos_fallidos'] >= 5) {
    $error = "Cuenta bloqueada por múltiples intentos fallidos";
}

// Verificar contraseña
elseif (password_verify($clave, $usuario['clave'])) {
    // Contraseña correcta

    // Actualizar último acceso e intentos fallidos
    $stmt = $conexion->prepare(
        "UPDATE usuarios
        SET ultimo_acceso = NOW(), intentos_fallidos = 0
        WHERE id = ?"
    );
    $stmt->bind_param("i", $usuario['id']);
    $stmt->execute();
}
```

```
// Guardar datos en sesión
$_SESSION['usuario_id'] = $usuario['id'];
$_SESSION['usuario_nombre'] = $usuario['nombre'];

// Si eligió "recordarme", crear cookie
if ($recordar) {
    $token = bin2hex(random_bytes(32));
    $expiracion = time() + 60*60*24*30; // 30 días
    $hash_token = password_hash($token, PASSWORD_DEFAULT);

    $stmt = $conexion->prepare(
        "UPDATE usuarios
        SET token_recuperacion = ?, expiracion_token = FROM_UNIXTIME(?)
        WHERE id = ?"
    );
    $stmt->bind_param("sii", $hash_token, $expiracion, $usuario['id']);
    $stmt->execute();

    setcookie("recordar", $usuario['id'].'.'.$token,
        $expiracion, "/", "", true, true);
}
```

```
// Redireccionar
header("Location: perfil.php");
exit();
} else {
// Contraseña incorrecta
$intentos = $usuario['intentos_fallidos'] + 1;
$stmt = $conexion->prepare(
"UPDATE usuarios SET intentos_fallidos = ? WHERE id = ?"
);
$stmt->bind_param("ii", $intentos, $usuario['id']);
$stmt->execute();

$error = "Correo o contraseña incorrectos";
}
} else {
$error = "Correo o contraseña incorrectos";
}
$stmt->close();
}
}
```

```
// Mostrar error si existe
if (!empty($error)) {
    echo "<div class='error'><p>" . $error . "</p></div>";
}

$conexion->close();
?>
```

## Gestión de sesiones seguras

Las sesiones permiten mantener el estado y recordar quién es el usuario entre diferentes páginas. Es fundamental configurarlas correctamente.

### Configuración segura de sesiones

```
<?php
// Configuración segura
ini_set('session.cookie_httponly', 1); ini_set('session.use_only_cookies', 1); ini_set('session.cookie_secure', 1);

// Iniciar sesión
session_start();
```

?>

## Verificar sesión activa

```
<?php
function verificarSesion() {
if (!isset($_SESSION['usuario_id'])) { header("Location: login.php");
exit();
}

// Verificar tiempo de inactividad
$tiempo_inactivo = 1800; // 30 min if (isset($_SESSION['ultimo_acceso']) && (time() -
$_SESSION['ultimo_acceso'])
> $tiempo_inactivo) {
session_unset();
session_destroy();
header("Location: login.php?expirado=1"); exit();
}
```

```
$_SESSION['ultimo_acceso'] = time(); return true;  
}  
?>
```

## Almacenar datos en sesión

```
<?php  
// Después de verificar credenciales $_SESSION['usuario_id'] =  
$usuario['id'];  
$_SESSION['usuario_nombre'] = $usuario['nombre'];  
$_SESSION['usuario_rol'] =  
$usuario['rol'];  
$_SESSION['ultimo_acceso'] = time();  
?>
```

## Cerrar sesión (logout.php)

```
<?php  
session_start();
```

```
// Destruir todas las variables $_SESSION = array();

// Borrar cookie de sesión
if (ini_get("session.use_cookies")) { $params =
    session_get_cookie_params(); setcookie(session_name(), "", time() - 42000,
    $params["path"],
    $params["domain"],
    $params["secure"],
    $params["httponly"]
);
}

// Destruir la sesión
session_destroy();

// Eliminar cookie "recordarme" if (isset($_COOKIE['recordar'])) { setcookie("recordar", "", time() - 3600, "/");
}

header("Location: login.php"); exit();
?>
```

# Transacciones en MySQL

Las transacciones permiten agrupar múltiples operaciones en una **unidad atómica**, garantizando la integridad de los datos.

## Atomicidad

Una transacción se ejecuta **completamente** o no se ejecuta

## Consistencia

La base de datos pasa de un **estado válido** a otro

## Aislamiento

Las transacciones se ejecutan de manera **aislada**

## Durabilidad

Una vez completada, la transacción **persiste**

## **Casos de uso comunes:**

**Transferencias bancarias** (débito y crédito simultáneos)

**Registro de usuarios** con datos en múltiples tablas

**Procesamiento de pedidos** (inventario y ventas)

Cualquier operación que modifique múltiples tablas relacionadas

# **100 Ejercicios Prácticos MySQL + PHP: Guía Completa Paso a Paso**

Esta es una colección completa de ejercicios progresivos, desde nivel básico hasta avanzado, cada uno con explicaciones detalladas línea por línea, preámbulos explicativos y ejemplos de despliegue local.

## **Ejercicio 1: Crear tu Primera Base de Datos ¿Qué es y para qué sirve?**

Una base de datos es un contenedor organizado donde almacenamos información relacionada. Es como crear una carpeta principal que contendrá todas nuestras tablas de datos.

## **¿Cómo funciona?**

MySQL organiza la información en bases de datos, que a su vez contienen tablas. Cada base de datos es independiente y puede tener sus propias configuraciones.

### **Paso a paso:**

Abrir phpMyAdmin (<http://localhost/phpmyadmin>)1.

Hacer clic en "Nueva" en el panel izquierdo2.

Escribir el nombre: "tienda\_online"3.

Seleccionar cotejamiento: utf8mb4\_general\_ci4.

Hacer clic en "Crear"5.

### **Código SQL equivalente:**

```
CREATE DATABASE tienda_online  
CHARACTER SET utf8mb4  
COLLATE utf8mb4_general_ci;
```

## Explicación línea por línea:

CREATE DATABASE: Comando para crear una nueva base de datos

tienda\_online: Nombre de nuestra base de datos

CHARACTER SET utf8mb4: Permite caracteres especiales y emojis

COLLATE utf8mb4\_general\_ci: Reglas de comparación de texto

## Despliegue local:

Verificar que aparece "tienda\_online" en la lista de bases de datos del panel izquierdo.

## Ejercicio 2: Crear Tabla de Usuarios con Campos Completos

### ¿Qué es y para qué sirve?

Una tabla es la estructura donde almacenamos datos organizados en filas y columnas. Cada columna representa un tipo de información (nombre, email, etc.) y cada fila un registro completo.

## **¿Cómo funciona?**

Las tablas tienen campos (columnas) con tipos de datos específicos. Cada campo tiene propiedades como longitud máxima, si puede estar vacío, valores por defecto, etc.

## **Paso a paso:**

Seleccionar la base de datos "tienda\_online"1.

Hacer clic en "Nueva tabla"2.

Nombre: "usuarios"3.

Número de campos: 84.

Configurar cada campo según la tabla5.

## **Estructura de campos:**

id INT, AUTO\_INCREMENT, PRIMARY KEY

nombre VARCHAR(50), NOT NULL

apellido VARCHAR(50), NOT NULL

email VARCHAR(100), NOT NULL, UNIQUE

password VARCHAR(255), NOT NULL

telefono VARCHAR(15), NULL

fecha\_registro TIMESTAMP, DEFAULT CURRENT\_TIMESTAMP

activo BOOLEAN, DEFAULT TRUE

## Código SQL:

```
CREATE TABLE usuarios (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL,
    apellido VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    telefono VARCHAR(15),
    fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    activo BOOLEAN DEFAULT TRUE
```

);

## Explicación línea por línea:

AUTO\_INCREMENT: Genera números consecutivos automáticamente

PRIMARY KEY: Identifica únicamente cada registro

NOT NULL: El campo es obligatorio

UNIQUE: No permite valores duplicados

VARCHAR(n): Texto variable hasta n caracteres

TIMESTAMP: Fecha y hora

DEFAULT: Valor automático si no se especifica

## Ejercicio 3: Insertar Datos con INSERT - Múltiples Métodos

### ¿Qué es y para qué sirve?

INSERT es el comando SQL que nos permite agregar nuevos registros (filas) a una tabla. Es la operación "CREATE" del CRUD.

## ¿Cómo funciona?

Especificamos la tabla destino, las columnas que queremos llenar y los valores correspondientes. MySQL insertará una nueva fila con esa información.

### Método 1: INSERT básico

```
INSERT INTO usuarios (nombre, apellido, email, password)  
VALUES ('Juan', 'Pérez', 'juan@email.com', 'mi_password_123');
```

### Método 2: INSERT múltiple

```
INSERT INTO usuarios (nombre, apellido, email, password)  
VALUES  
('María', 'García', 'maria@email.com', 'password456'),  
('Carlos', 'López', 'carlos@email.com', 'clave789'),  
('Ana', 'Martínez', 'ana@email.com', 'secreto321');
```

### Método 3: INSERT con todos los campos

```
INSERT INTO usuarios (nombre, apellido, email, password, telefono, activo)
VALUES ('Pedro', 'Rodríguez', 'pedro@email.com', 'pass123', '555-1234', TRUE);
```

## Explicación línea por línea:

INSERT INTO: Comando para insertar datos

usuarios: Nombre de la tabla destino

(nombre, apellido...): Columnas que vamos a llenar

VALUES: Palabra clave que indica los valores

('Juan', 'Pérez'...): Valores en el mismo orden que las columnas

Strings van entre comillas simples

Números y booleanos sin comillas

## Paso a paso en phpMyAdmin:

Seleccionar tabla "usuarios"1.

Clic en pestaña "Insertar"2.

Llenar los campos del formulario3.

Clic en "Continuar"4.

## Despliegue local:

Verificar en la pestaña "Examinar" que los registros se insertaron correctamente.

# Ejercicio 4: Consultas SELECT - Desde Básico hasta Avanzado

## ¿Qué es y para qué sirve?

SELECT es el comando más usado en SQL. Nos permite recuperar y mostrar datos de las tablas según criterios específicos. **¿Cómo funciona?**

SELECT examina las tablas, aplica filtros si los hay, y devuelve las filas que coinciden con nuestros criterios. **Consulta 1: Seleccionar todo**

```
SELECT * FROM usuarios;
```

\*: Significa "todas las columnas"

FROM usuarios: De la tabla usuarios

## Consulta 2: Columnas específicas

```
SELECT nombre, apellido, email FROM usuarios;
```

Solo muestra las columnas especificadas

Más eficiente que SELECT \*

## Consulta 3: Con condición WHERE

```
SELECT * FROM usuarios WHERE activo = TRUE;
```

WHERE: Filtra registros que cumplan la condición

Solo usuarios activos

## Consulta 4: Múltiples condiciones

```
SELECT nombre, email FROM usuarios  
WHERE activo = TRUE AND nombre LIKE 'J%';
```

AND: Ambas condiciones deben cumplirse

LIKE 'J%': Nombres que empiecen con J

%: Comodín que representa cualquier texto

## Consulta 5: Ordenamiento

```
SELECT * FROM usuarios  
ORDER BY fecha_registro DESC;
```

ORDER BY: Ordena los resultados

DESC: Descendente (más reciente primero)

ASC: Ascendente (por defecto)

## Consulta 6: Limitar resultados

```
SELECT * FROM usuarios
```

```
ORDER BY fecha_registro DESC  
LIMIT 5;
```

LIMIT 5: Solo los primeros 5 resultados

## Despliegue local:

Ejecutar cada consulta en la pestaña "SQL" de phpMyAdmin y observar los diferentes resultados.

# Ejercicio 5: UPDATE - Actualizar Registros de Forma Segura

## ¿Qué es y para qué sirve?

UPDATE modifica datos existentes en una tabla. Es fundamental para mantener la información actualizada en nuestras aplicaciones.

## ¿Cómo funciona?

UPDATE localiza registros que cumplan una condición y modifica los campos especificados con nuevos valores.

## **REGLA DE ORO: SIEMPRE usar WHERE**

Sin WHERE, UPDATE modificará TODOS los registros de la tabla.

### **Actualización 1: Un campo específico**

UPDATE usuarios

SET telefono = '555-9999'

WHERE id = 1;

SET: Especifica qué campo cambiar

WHERE id = 1: Solo el usuario con ID 1

### **Actualización 2: Múltiples campos**

UPDATE usuarios

SET nombre = 'Juan Carlos',

telefono = '555-8888',

activo = TRUE

WHERE email = 'juan@email.com';

Múltiples campos separados por comas

WHERE con email como condición

## Actualización 3: Con condiciones complejas

UPDATE usuarios

SET activo = FALSE

WHERE fecha\_registro < '2023-01-01'

AND telefono IS NULL;

Desactivar usuarios antiguos sin teléfono

IS NULL: Campo vacío

Múltiples condiciones con AND

## Actualización 4: Usando CASE (condicional)

UPDATE usuarios

SET activo = CASE

WHEN fecha\_registro < '2023-01-01' THEN FALSE

```
ELSE TRUE  
END;
```

CASE: Estructura condicional

WHEN-THEN-ELSE: Si-entonces-sino

## Verificación antes de actualizar:

-- Primero ver qué se va a cambiar

```
SELECT * FROM usuarios WHERE id = 1;
```

-- Luego hacer el UPDATE

```
UPDATE usuarios SET telefono = '555-9999' WHERE id = 1;
```

-- Verificar el cambio

```
SELECT * FROM usuarios WHERE id = 1;
```

## Despliegue local:

Ejecutar SELECT para ver datos actuales1.

Ejecutar UPDATE2.

Verificar cambios con otro SELECT3.

## Ejercicio 6: DELETE - Eliminar Registros con Precaución ¿Qué es y para qué sirve?

DELETE elimina registros completos de una tabla. Es una operación PERMANENTE que no se puede deshacer fácilmente. [¿Cómo funciona?](#)

DELETE localiza registros que cumplan la condición WHERE y los elimina completamente de la tabla.

### ADVERTENCIA CRÍTICA

Sin WHERE elimina TODOS los registros

La operación es IRREVERSIBLE

Siempre hacer backup antes de DELETE masivos

### Eliminación 1: Por ID específico

```
DELETE FROM usuarios WHERE id = 5;
```

Elimina solo el usuario con ID 5

La forma más segura de eliminar

## Eliminación 2: Por condición

```
DELETE FROM usuarios  
WHERE activo = FALSE  
AND fecha_registro < '2022-01-01';
```

Elimina usuarios inactivos y antiguos

Múltiples condiciones para mayor precisión

## Eliminación 3: Con LIMIT (seguridad extra)

```
DELETE FROM usuarios  
WHERE activo = FALSE  
LIMIT 1;
```

**LIMIT 1:** Solo elimina un registro

Previene eliminaciones masivas accidentales

### **Proceso seguro de eliminación:**

-- PASO 1: Ver qué se va a eliminar

```
SELECT * FROM usuarios WHERE id = 5;
```

-- PASO 2: Contar cuántos registros

```
SELECT COUNT(*) FROM usuarios WHERE id = 5;
```

-- PASO 3: Si está correcto, eliminar

```
DELETE FROM usuarios WHERE id = 5;
```

-- PASO 4: Verificar que se eliminó

```
SELECT * FROM usuarios WHERE id = 5;
```

### **Alternativa: Eliminación lógica**

En lugar de DELETE, marcar como eliminado:

```
-- Agregar campo deleted_at a la tabla
```

```
ALTER TABLE usuarios ADD COLUMN deleted_at TIMESTAMP NULL;
```

```
-- "Eliminar" marcando fecha
```

```
UPDATE usuarios
```

```
SET deleted_at = NOW()
```

```
WHERE id = 5;
```

```
-- Consultar solo registros no eliminados
```

```
SELECT * FROM usuarios WHERE deleted_at IS NULL;
```

## Despliegue local:

Crear registros de prueba1.

Usar SELECT para verificar antes de eliminar2.

Ejecutar DELETE con WHERE específico3.

Confirmar eliminación con SELECT4.

# Ejercicio 7: Primera Conexión PHP-MySQL con

# MySQLi

## ¿Qué es y para qué sirve?

La conexión PHP-MySQL es el puente que permite a nuestras aplicaciones web comunicarse con la base de datos para leer y escribir información.

## ¿Cómo funciona?

PHP establece una conexión TCP/IP con el servidor MySQL usando credenciales. Una vez conectado, puede enviar consultas SQL y recibir resultados.

### Paso a paso:

01

#### 1. Crear archivo conexion.php

Código básico de conexión:

```
servidor = "localhost" (dirección del servidor MySQL)  
usuario = "root" (usuario de MySQL por defecto en XAMPP)  
password = "" (contraseña vacía por defecto en XAMPP)
```

basedatos = "tienda\_online" (nombre de nuestra base de datos)

02

## **2. Crear objeto MySQLi**

new mysqli() crea la conexión con los parámetros dados

03

## **3. Verificar conexión**

connect\_error contiene errores si los hay

die() detiene ejecución y muestra mensaje de error

04

## **4. Configurar charset**

set\_charset("utf8mb4") permite caracteres especiales y emojis

## **Explicación línea por línea:**

Parámetros de conexión se almacenan en variables

```
new mysqli() intenta conectar al servidor  
if (connect_error) verifica si hubo errores  
die() termina el script si hay problemas  
set_charset() configura la codificación  
echo muestra mensaje de éxito
```

## Versión mejorada con try-catch:

Usar try-catch permite mejor manejo de errores y código más limpio. **Despliegue local:**

Asegurar que XAMPP esté ejecutándose (Apache y MySQL)1. Crear archivo conexion.php en carpeta htdocs2.

Abrir navegador en <http://localhost/conexion.php>3.

Verificar mensaje "Conexión exitosa a la base de datos"4.

## Ejercicio 8: Insertar Datos desde PHP con Sentencias Preparadas

### ¿Qué es y para qué sirve?

Las sentencias preparadas son la forma más segura de insertar datos desde PHP. Separan el código SQL de los datos, previniendo inyecciones SQL.

## ¿Cómo funciona?

01

### 1. Preparamos la consulta con marcadores (?)

03

### 3. Ejecutamos la consulta

**Código completo - insertar\_usuario.php:** Incluir conexión:

```
require_once 'conexion.php';
```

Datos a insertar:

```
$nombre = "Laura";  
$apellido = "González";
```

```
$email = "laura@email.com";
$password = password_hash("mi_clave_123", PASSWORD_DEFAULT); $telefono = "555-1234";
```

Preparar la consulta:

02

## **2. Vinculamos los parámetros con sus tipos**

04

## **4. MySQL procesa datos y consulta por separado**

```
$sql = "INSERT INTO usuarios (nombre, apellido, email, password, telefono) VALUES (?, ?, ?, ?, ?)"; $stmt = $conexion->prepare($sql);
```

Vincular parámetros:

```
$stmt->bind_param("sssss", $nombre, $apellido, $email, $password, $telefono); Ejecutar y verificar:
```

```
if ($stmt->execute()) {  
    echo "Usuario insertado correctamente";  
    echo "ID del nuevo usuario: " . $conexion->insert_id;  
} else {  
    echo "Error: " . $stmt->error;  
}
```

Cerrar recursos:

```
$stmt->close();  
$conexion->close();
```

## Explicación línea por línea:

prepare(): Prepara la consulta con marcadores ?

bind\_param(): Vincula variables a los marcadores

"sssss": Tipos de datos (s=string, i=integer, d=double, b=blob)

`execute()`: Ejecuta la consulta preparada

`insert_id`: ID del último registro insertado

`close()`: Libera recursos de memoria

## Tipos de parámetros:

`s`: string (texto)

`i`: integer (números enteros)

`d`: double (números decimales)

`b`: blob (datos binarios)

## Despliegue local:

Crear archivo `insertar_usuario.php`.

Ejecutar desde navegador.

Verificar en phpMyAdmin que se insertó el usuario.

Observar que la contraseña está encriptada.

# Ejercicio 9: Consultar y Mostrar Datos en HTML ¿Qué

## **es y para qué sirve?**

Consultar datos desde PHP nos permite mostrar información de la base de datos en páginas web dinámicas, creando contenido que cambia según los datos almacenados.

## **¿Cómo funciona?**

**PHP ejecuta una consulta SELECT Recibe los resultados Los procesa para mostrarlos en formato HTML**

### **Código completo - mostrar\_usuarios.php:**

Incluir conexión y consultar:

```
require_once 'conexion.php';

$sql = "SELECT id, nombre, apellido, email, telefono, fecha_registro, activo FROM usuarios ORDER BY fecha_registro DESC"; $resultado =
$conexion->query($sql);
```

Estructura HTML básica:

```
<!DOCTYPE html>
<html>
<head>
<title>Lista de Usuarios</title>
<style>
table { border-collapse: collapse; width: 100%; }
th, td { border: 1px solid #ddd; padding: 8px; text-align: left; }
th { background-color: #f2f2f2; }
.activo { color: green; }
.inactivo { color: red; }
</style>
</head>
<body>
<h1>Lista de Usuarios Registrados</h1>
```

Verificar si hay resultados:

```
if ($resultado->num_rows > 0) {
echo "<p>Total de usuarios: " . $resultado->num_rows . "</p>";
echo "<table>";
```

```
echo "<tr><th>ID</th><th>Nombre</th><th>Apellido</th><th>Email</th><th>Teléfono</th><th>Fecha
```

Registro</th><th>Estado</th></tr>"; Iterar sobre los resultados:

```
while($fila = $resultado->fetch_assoc()) {  
    echo "<tr>";  
    echo "<td>" . $fila["id"] . "</td>";  
    echo "<td>" . $fila["nombre"] . "</td>";  
    echo "<td>" . $fila["apellido"] . "</td>";  
    echo "<td>" . $fila["email"] . "</td>";  
    echo "<td>" . ($fila["telefono"] ? $fila["telefono"] : "No especificado") . "</td>";  
    echo "<td>" . date("d/m/Y H:i", strtotime($fila["fecha_registro"])) . "</td>";  
  
    $estado = $fila["activo"] ? "Activo" : "Inactivo";  
    $clase = $fila["activo"] ? "activo" : "inactivo";  
    echo "<td class='$clase'>$estado</td>";  
    echo "</tr>";  
}  
echo "</table>";  
} else {  
    echo "<p>No hay usuarios registrados.</p>";
```

}

## Explicación línea por línea:

query(): Ejecuta consulta SELECT directa

num\_rows: Número de filas devueltas

fetch\_assoc(): Obtiene fila como array asociativo

while(): Itera sobre cada resultado

strtotime(): Convierte fecha a timestamp

date(): Formatea fecha legible

Operador ternario ?: para valores condicionales

## Despliegue local:

Crear archivo mostrar\_usuarios.php1.

Abrir en navegador2.

Ver tabla con todos los usuarios3.

Verificar formato de fechas y estados4.

# Ejercicio 10: Actualizar Registros desde PHP

## ¿Qué es y para qué sirve?

Actualizar registros desde PHP permite modificar información existente en la base de datos a través de formularios web o procesos automáticos.

## ¿Cómo funciona?

Usamos sentencias preparadas con UPDATE para modificar campos específicos de registros que cumplan ciertas condiciones.

**Código completo - actualizar\_usuario.php:** Incluir conexión:

```
require_once 'conexion.php';
```

Datos para actualizar:

```
$id_usuario = 1; // ID del usuario a actualizar  
$nuevo_nombre = "Juan Carlos";
```

```
$nuevo_apellido = "Pérez Rodríguez";
$nuevo_telefono = "555-9876";
$nuevo_activo = true;
```

Preparar consulta UPDATE:

```
$sql = "UPDATE usuarios SET nombre = ?, apellido = ?, telefono = ?, activo = ? WHERE id = ?";
$stmt = $conexion->prepare($sql);
```

Verificar preparación:

```
if (!$stmt) {
    die("Error en prepare: " . $conexion->error);
}
```

Vincular parámetros:

```
$stmt->bind_param("sssi", $nuevo_nombre, $nuevo_apellido, $nuevo_telefono, $nuevo_activo, $id_usuario);
```

Ejecutar y verificar:

```
if ($stmt->execute()) {  
    if ($stmt->affected_rows > 0) {  
        echo "Usuario actualizado correctamente";  
        echo "<br>Registros afectados: " . $stmt->affected_rows;  
    } else {  
        echo "No se encontró el usuario con ID: " . $id_usuario;  
    }  
} else {  
    echo "Error al actualizar: " . $stmt->error;  
}
```

Cerrar recursos:

```
$stmt->close();  
$conexion->close();
```

Versión con verificación previa:

```
// Primero verificar que el usuario existe  
$sql_check = "SELECT id, nombre, apellido FROM usuarios WHERE id = ?"; $stmt_check = $conexion->prepare($sql_check);
```

```
$stmt_check->bind_param("i", $id_usuario);
$stmt_check->execute();
$resultado = $stmt_check->get_result();

if ($resultado->num_rows > 0) {
    $usuario_actual = $resultado->fetch_assoc();
    echo "Actualizando usuario: " . $usuario_actual['nombre'] . " " . $usuario_actual['apellido'];

    // Proceder con la actualización...
} else {
    echo "Usuario no encontrado";
}
```

## Explicación línea por línea:

bind\_param("sssi"): s=string para nombre, apellido, teléfono; i=integer para activo e id

affected\_rows: Número de registros modificados

get\_result(): Obtiene resultado de consulta preparada

Verificación previa evita actualizaciones innecesarias

## Despliegue local:

Crear archivo actualizar\_usuario.php1.

Modificar variables con datos de prueba2.

Ejecutar desde navegador3.

Verificar cambios en phpMyAdmin4.

## Ejercicio 11: Eliminar Registros desde PHP de Forma Segura

### ¿Qué es y para qué sirve?

Eliminar registros desde PHP permite remover información obsoleta o no deseada de la base de datos. Es una operación crítica que requiere máxima precaución.

### ¿Cómo funciona?

Usamos sentencias preparadas con DELETE para eliminar registros específicos. Siempre verificamos antes de eliminar y confirmamos después.

### Paso a paso:

01

## **Incluir conexión y definir ID**

require\_once 'conexion.php'

Definir variable con ID del usuario a eliminar

02

## **Verificar existencia del usuario**

Consulta SELECT para confirmar que existe

Mostrar información del usuario antes de eliminar

Si no existe, detener ejecución con mensaje de error

03

## **Preparar consulta DELETE**

Usar sentencias preparadas para seguridad

Vincular parámetro ID con bind\_param

Ejecutar consulta y verificar resultado

## Confirmar eliminación

Verificar affected\_rows para confirmar eliminación

Mostrar mensaje de éxito o error

Cerrar recursos de memoria

## Explicación línea por línea:

get\_result(): Obtiene resultados de consulta preparada

affected\_rows: Número de registros afectados por la operación die(): Detiene ejecución si no existe el usuario

Verificación previa evita errores y confirma existencia

bind\_param("i", variable): Vincula entero de forma segura

## Versión con confirmación adicional:

Agregar parámetro GET 'confirmar' para evitar eliminaciones accidentales. Mostrar enlace de confirmación antes de proceder.

## **Medidas de seguridad:**

Siempre verificar existencia antes de eliminar

Usar sentencias preparadas

Mostrar información del registro antes de eliminar

Implementar confirmación de usuario

Registrar eliminaciones en log de auditoría

## **Despliegue local:**

Crear archivo eliminar\_usuario.php1.

Modificar variable con ID válido de usuario existente2.

Ejecutar desde navegador3.

Verificar eliminación en phpMyAdmin4.

Comprobar que el registro ya no existe5.

# **Ejercicio 12: Crear Formulario HTML para Registro de Usuarios**

## ¿Qué es y para qué sirve?

Un formulario HTML es la interfaz que permite a los usuarios ingresar datos que serán procesados por PHP y almacenados en la base de datos. ¿Cómo funciona?

El formulario captura datos del usuario, los envía vía POST a un script PHP que los procesa, valida y almacena en MySQL. Código

## completo - registro\_form.html:

### Estructura HTML básica:

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Registro de Usuario</title>
<style>
```

```
body { font-family: Arial, sans-serif; max-width: 600px; margin: 50px auto; padding: 20px; }

.form-group { margin-bottom: 15px; }

label { display: block; margin-bottom: 5px; font-weight: bold; }

input, select, textarea { width: 100%; padding: 8px; border: 1px solid #ddd; border-radius: 4px; }

button { background: #007bff; color: white; padding: 10px 20px; border: none; border-radius: 4px; cursor: pointer; } button:hover { background: #0056b3; }

.error { color: red; font-size: 14px; }

.required { color: red; }

</style>

</head>

<body>
```

## Formulario principal:

```
<h2>Crear Nueva Cuenta</h2>

<form action="procesar_registro.php" method="POST" id="formRegistro">

    <div class="form-group">
        <label for="nombre">Nombre <span class="required">*</span></label>
        <input type="text" id="nombre" name="nombre" required maxlength="50" placeholder="Ingrese su nombre"> </div>
```

```
<div class="form-group">
<label for="apellido">Apellido <span class="required">*</span></label>
<input type="text" id="apellido" name="apellido" required maxlength="50" placeholder="Ingrese su apellido"> </div>

<div class="form-group">
<label for="email">Correo Electrónico <span class="required">*</span></label>
<input type="email" id="email" name="email" required maxlength="100" placeholder="ejemplo@correo.com"> </div>

<div class="form-group">
<label for="password">Contraseña <span class="required">*</span></label>
<input type="password" id="password" name="password" required minlength="6" placeholder="Mínimo 6 caracteres"> </div>

<div class="form-group">
<label for="password2">Confirmar Contraseña <span class="required">*</span></label>
<input type="password" id="password2" name="password2" required placeholder="Repita la contraseña"> </div>

<div class="form-group">
<label for="telefono">Teléfono</label>
<input type="tel" id="telefono" name="telefono" maxlength="15" placeholder="555-1234">
</div>

<div class="form-group">
```

```
<button type="submit">Registrar Usuario</button>
<button type="reset">Limpiar Formulario</button>
</div>

</form>
```

## JavaScript para validación:

```
<script>
document.getElementById('formRegistro').addEventListener('submit', function(e) {
  const password = document.getElementById('password').value;
  const password2 = document.getElementById('password2').value;

  if (password !== password2) {
    e.preventDefault();
    alert('Las contraseñas no coinciden');
    return false;
  }

  if (password.length < 6) {
    e.preventDefault();
```

```
        alert('La contraseña debe tener al menos 6 caracteres');
        return false;
    }
});
</script>
```

## Explicación línea por línea:

method="POST": Envía datos de forma segura (no visible en URL)

action="procesar\_registro.php": Archivo que procesará los datos

required: Campo obligatorio (validación HTML5)

maxlength: Longitud máxima permitida

type="email": Validación automática de formato email

type="password": Oculta el texto ingresado

addEventListener: Escucha evento submit del formulario

preventDefault(): Detiene envío si hay errores

## Despliegue local:

Crear archivo registro\_form.html en htdocs1.

Abrir [http://localhost/registro\\_form.html](http://localhost/registro_form.html)2.

Probar validaciones ingresando datos incorrectos3.

Verificar que funciona antes de crear el procesador PHP4.

## Ejercicio 13: Procesar Formulario con Validación Completa

### ¿Qué es y para qué sirve?

El procesamiento de formularios es el corazón de las aplicaciones web dinámicas. Valida, sanitiza y almacena datos de forma segura.

### ¿Cómo funciona?

PHP recibe datos POST, los valida, sanitiza, verifica duplicados y los inserta en la base de datos usando sentencias preparadas.

### Código completo - `procesar_registro.php`:

## Inicialización y conexión:

```
<?php
session_start();
require_once 'conexion.php';

// Array para almacenar errores
$errores = array();
$exito = false;

// Verificar que se envió el formulario
if ($_SERVER['REQUEST_METHOD'] !== 'POST') {
    header('Location: registro_form.html');
    exit();
}
```

## Sanitización de datos:

```
// Sanitizar datos de entrada
$nombre = trim(htmlspecialchars($_POST['nombre'] ?? ""));
$apellido = trim(htmlspecialchars($_POST['apellido'] ?? ""));
```

```
$email = trim(filter_var($_POST['email'] ?? "", FILTER_SANITIZE_EMAIL));
$password = $_POST['password'] ?? "";
$password2 = $_POST['password2'] ?? "";
$telefono = trim(htmlspecialchars($_POST['telefono'] ?? ""));
```

## Validación de campos:

```
// Validar nombre
if (empty($nombre)) {
    $errores[] = "El nombre es obligatorio";
} elseif (strlen($nombre) < 2) {
    $errores[] = "El nombre debe tener al menos 2 caracteres";
} elseif (strlen($nombre) > 50) {
    $errores[] = "El nombre no puede exceder 50 caracteres";
}

// Validar apellido
if (empty($apellido)) {
    $errores[] = "El apellido es obligatorio";
} elseif (strlen($apellido) < 2) {
    $errores[] = "El apellido debe tener al menos 2 caracteres";
```

```
}

// Validar email
if (empty($email)) {
    $errores[] = "El email es obligatorio";
} elseif (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
    $errores[] = "El formato del email no es válido";
}

// Validar contraseñas
if (empty($password)) {
    $errores[] = "La contraseña es obligatoria";
} elseif (strlen($password) < 6) {
    $errores[] = "La contraseña debe tener al menos 6 caracteres";
} elseif ($password !== $password2) {
    $errores[] = "Las contraseñas no coinciden";
}

// Validar teléfono (opcional)
if (!empty($telefono) && !preg_match('/^([0-9]+(\-[0-9]+){0,3})$/i', $telefono)) {
    $errores[] = "El formato del teléfono no es válido";
}
```

## **Verificar email duplicado:**

```
if (empty($errores)) {  
    // Verificar si el email ya existe  
    $sql_verificar = "SELECT id FROM usuarios WHERE email = ?";  
    $stmt_verificar = $conexion->prepare($sql_verificar);  
    $stmt_verificar->bind_param("s", $email);  
    $stmt_verificar->execute();  
    $resultado = $stmt_verificar->get_result();  
  
    if ($resultado->num_rows > 0) {  
        $errores[] = "Este email ya está registrado";  
    }  
    $stmt_verificar->close();  
}
```

## **Insertar usuario si no hay errores:**

```
if (empty($errores)) {
```

```
// Encriptar contraseña
$password_hash = password_hash($password, PASSWORD_DEFAULT);

// Insertar nuevo usuario
$sql_insertar = "INSERT INTO usuarios (nombre, apellido, email, password, telefono) VALUES (?, ?, ?, ?, ?); $stmt_insertar =
$conexion->prepare($sql_insertar);
$stmt_insertar->bind_param("sssss", $nombre, $apellido, $email, $password_hash, $telefono);

if ($stmt_insertar->execute()) {
$exito = true;
$nuevo_id = $conexion->insert_id;
} else {
$errores[] = "Error al registrar usuario: " . $stmt_insertar->error;
}
$stmt_insertar->close();
}

$conexion->close();
```

## Explicación línea por línea:

`htmlspecialchars()`: Convierte caracteres especiales a entidades HTML

`filter_var()`: Sanitiza y valida datos según filtros específicos

`trim()`: Elimina espacios en blanco al inicio y final

`preg_match()`: Verifica patrones con expresiones regulares

`password_hash()`: Encripta contraseña de forma segura

`?? ''`: Operador null coalescing, valor por defecto si es null

## **Despliegue local:**

Crear archivo procesar\_registro.php1.

Probar con datos válidos e inválidos2.

Verificar mensajes de error y éxito3.

Comprobar inserción en phpMyAdmin4.

# **Ejercicio 14: Sistema de Login con Verificación de Contraseñas**

**¿Qué es y para qué sirve?**

Un sistema de login autentica usuarios verificando sus credenciales contra la base de datos y estableciendo sesiones para mantener el estado de autenticación.

## ¿Cómo funciona?

Compara email y contraseña ingresados con los almacenados en la base de datos, usando `password_verify()` para verificar contraseñas encriptadas.

## Código completo - login.php:

### Inicialización:

```
<?php
session_start();

// Si ya está logueado, redirigir
if (isset($_SESSION['usuario_id'])) {
    header('Location: dashboard.php');
    exit();
}
```

```
require_once 'conexion.php';
```

```
$error = ";
```

```
$email = ";
```

## Procesamiento del formulario:

```
if ($_SERVER['REQUEST_METHOD'] === 'POST') {  
    // Sanitizar datos  
    $email = trim(filter_var($_POST['email'] ?? "", FILTER_SANITIZE_EMAIL));  
    $password = $_POST['password'] ?? "";  
    $recordar = isset($_POST['recordar']);  
  
    // Validar campos básicos  
    if (empty($email) || empty($password)) {  
        $error = 'Por favor complete todos los campos';  
    } elseif (!filter_var($email, FILTER_VALIDATE_EMAIL)) {  
        $error = 'Formato de email inválido';  
    } else {  
        // Buscar usuario en la base de datos
```

```
$sql = "SELECT id, nombre, apellido, email, password, activo FROM usuarios WHERE email = ?";
$stmt = $conexion->prepare($sql);
$stmt->bind_param("s", $email);
$stmt->execute();
$resultado = $stmt->get_result();

if ($resultado->num_rows === 1) {
    $usuario = $resultado->fetch_assoc();

    // Verificar si el usuario está activo
    if (!$usuario['activo']) {
        $error = 'Cuenta desactivada. Contacte al administrador';
    }

    // Verificar contraseña
    elseif (password_verify($password, $usuario['password'])) {
        // Login exitoso
        $_SESSION['usuario_id'] = $usuario['id'];
        $_SESSION['usuario_nombre'] = $usuario['nombre'];
        $_SESSION['usuario_apellido'] = $usuario['apellido'];
        $_SESSION['usuario_email'] = $usuario['email'];
        $_SESSION['login_time'] = time();
    }
}
```

```
// Actualizar último acceso
$sql_update = "UPDATE usuarios SET ultimo_acceso = NOW() WHERE id = ?";
$stmt_update = $conexion->prepare($sql_update);
$stmt_update->bind_param("i", $usuario['id']);
$stmt_update->execute();
$stmt_update->close();

// Cookie "recordar" (opcional)
if ($recordar) {
    $token = bin2hex(random_bytes(32));
    setcookie('remember_token', $token, time() + (30 * 24 * 60 * 60), '/', '', false, true);
}

// Guardar token en base de datos
$sql_token = "UPDATE usuarios SET remember_token = ? WHERE id = ?";
$stmt_token = $conexion->prepare($sql_token);
$stmt_token->bind_param("si", $token, $usuario['id']);
$stmt_token->execute();
$stmt_token->close();
}

// Redirigir al dashboard
header('Location: dashboard.php');
```

```
exit();
} else {
$error = 'Email o contraseña incorrectos';
}
} else {
$error = 'Email o contraseña incorrectos';
}

$stmt->close();
}
}

$connexion->close();
```

## Formulario HTML:

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8">
<title>Iniciar Sesión</title>
```

```
<style>
body { font-family: Arial, sans-serif; max-width: 400px; margin: 100px auto; padding: 20px; }
.form-group { margin-bottom: 15px; }
label { display: block; margin-bottom: 5px; }
input { width: 100%; padding: 10px; border: 1px solid #ddd; border-radius: 4px; }
button { width: 100%; padding: 12px; background: #007bff; color: white; border: none; border-radius: 4px; } .error { color: red; margin-bottom: 15px; padding: 10px; background: #ffe6e6; border-radius: 4px; }
.checkbox { width: auto; margin-right: 5px; }
</style>
</head>
<body>
<h2>Iniciar Sesión</h2>

<?php if ($error): ?>
<div class="error"><?php echo htmlspecialchars($error); ?></div>
<?php endif; ?>

<form method="POST">
<div class="form-group">
<label for="email">Email:</label>
<input type="email" id="email" name="email" value="<?php echo htmlspecialchars($email); ?>" required> </div>
```

```
<div class="form-group">
<label for="password">Contraseña:</label>
<input type="password" id="password" name="password" required>
</div>

<div class="form-group">
<input type="checkbox" id="recordar" name="recordar" class="checkbox">
<label for="recordar" style="display: inline;">Recordarme</label>
</div>

<button type="submit">Iniciar Sesión</button>
</form>

<p><a href="registro_form.html">¿No tienes cuenta? Regístrate</a></p>
</body>
</html>
```

## Explicación línea por línea:

password\_verify(): Verifica contraseña contra hash almacenado

`$_SESSION`: Array superglobal para datos de sesión  
`bin2hex(random_bytes())`: Genera token seguro aleatorio  
`setcookie()`: Establece cookie en el navegador  
`time()`: Timestamp actual en segundos  
`NOW()`: Función MySQL para fecha/hora actual

## Despliegue local:

1. Crear archivo login.php1.
2. Probar con credenciales válidas e inválidas2.
3. Verificar que establece sesión correctamente3.
4. Comprobar redirección después del login4.

# Ejercicio 15: Dashboard Protegido con Control de Sesiones

## ¿Qué es y para qué sirve?

Un dashboard es la página principal después del login, protegida por verificación de sesiones. Muestra información personalizada del usuario autenticado.

## ¿Cómo funciona?

Verifica que existe una sesión válida, muestra datos del usuario y proporciona funcionalidades exclusivas para usuarios autenticados.

## Código completo - dashboard.php:

Verificación de sesión:

```
<?php
session_start();

// Función para verificar sesión
function verificarSesion() {
    if (!isset($_SESSION['usuario_id'])) {
        header('Location: login.php');
        exit();
    }

    // Verificar tiempo de sesión (opcional)
    if (isset($_SESSION['login_time'])) {
```

```
$tiempo_limite = 2 * 60 * 60; // 2 horas
if (time() - $_SESSION['login_time'] > $tiempo_limite) {
    session_destroy();
    header('Location: login.php?mensaje=sesion_expirada');
    exit();
}
}

}

}

verificarSesion();

require_once 'conexion.php';
```

Obtener datos del usuario:

```
// Obtener información actualizada del usuario
$sql = "SELECT nombre, apellido, email, telefono, fecha_registro, ultimo_acceso FROM usuarios WHERE id = ?"; $stmt =
$conexion->prepare($sql);
$stmt->bind_param("i", $_SESSION['usuario_id']);
$stmt->execute();
$resultado = $stmt->get_result();
```

```
$usuario = $resultado->fetch_assoc();
$stmt->close();

// Estadísticas adicionales (ejemplo)
$sql_stats = "SELECT
(SELECT COUNT(*) FROM usuarios) as total_usuarios,
(SELECT COUNT(*) FROM usuarios WHERE DATE(fecha_registro) = CURDATE()) as usuarios_hoy";
$resultado_stats = $conexion->query($sql_stats);
$stats = $resultado_stats->fetch_assoc();

$conexion->close();
```

HTML del Dashboard:

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Dashboard - Panel de Control</title>
<style>
```

```
* { margin: 0; padding: 0; box-sizing: border-box; }
body { font-family: Arial, sans-serif; background: #f4f4f4; }
.header { background: #007bff; color: white; padding: 1rem; display: flex; justify-content: space-between; align-items: center; } .container
{ max-width: 1200px; margin: 2rem auto; padding: 0 1rem; }
.welcome { background: white; padding: 2rem; border-radius: 8px; margin-bottom: 2rem; box-shadow: 0 2px 4px rgba(0,0,0,0.1); } .stats {
display: grid; grid-template-columns: repeat(auto-fit, minmax(250px, 1fr)); gap: 1rem; margin-bottom: 2rem; } .stat-card { background:
white; padding: 1.5rem; border-radius: 8px; text-align: center; box-shadow: 0 2px 4px rgba(0,0,0,0.1); } .stat-number { font-size: 2rem;
font-weight: bold; color: #007bff; }
.user-info { background: white; padding: 2rem; border-radius: 8px; box-shadow: 0 2px 4px rgba(0,0,0,0.1); } .info-row { display: flex;
justify-content: space-between; padding: 0.5rem 0; border-bottom: 1px solid #eee; } .logout-btn { background: #dc3545; color: white;
padding: 0.5rem 1rem; text-decoration: none; border-radius: 4px; } .logout-btn:hover { background: #c82333; }
.nav-links a { color: white; text-decoration: none; margin-left: 1rem; }
</style>
</head>
<body>
<header class="header">
<h1>Panel de Control</h1>
<nav>
<span>Bienvenido, <?php echo htmlspecialchars($usuario['nombre']); ?></span>
<a href="perfil.php" class="nav-links">Mi Perfil</a>
<a href="usuarios.php" class="nav-links">Usuarios</a>
<a href="logout.php" class="logout-btn">Cerrar Sesión</a>
```

```
</nav>
</header>

<div class="container">
<div class="welcome">
<h2>¡Bienvenido de vuelta, <?php echo htmlspecialchars($usuario['nombre'] . ' ' . $usuario['apellido']); ?>!</h2> <p>Último acceso: <?php
echo date('d/m/Y H:i:s', strtotime($usuario['ultimo_acceso'])); ?></p>
</div>

<div class="stats">
<div class="stat-card">
<div class="stat-number"><?php echo $stats['total_usuarios']; ?></div>
<div>Total de Usuarios</div>
</div>
<div class="stat-card">
<div class="stat-number"><?php echo $stats['usuarios_hoy']; ?></div>
<div>Registros Hoy</div>
</div>
<div class="stat-card">
<div class="stat-number"><?php echo date('H:i'); ?></div>
<div>Hora Actual</div>
</div>
```

```
</div>

<div class="user-info">
<h3>Información de tu Cuenta</h3>
<div class="info-row">
<strong>Email:</strong>
<span><?php echo htmlspecialchars($usuario['email']); ?></span>
</div>
<div class="info-row">
<strong>Teléfono:</strong>
<span><?php echo htmlspecialchars($usuario['telefono'] ?: 'No especificado'); ?></span>
</div>
<div class="info-row">
<strong>Fecha de Registro:</strong>
<span><?php echo date('d/m/Y', strtotime($usuario['fecha_registro'])); ?></span>
</div>
<div class="info-row">
<strong>ID de Usuario:</strong>
<span><?php echo $_SESSION['usuario_id']; ?></span>
</div>
</div>
</div>
```

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

## Explicación línea por línea:

verificarSesion(): Función personalizada para validar sesión activa

time() - \$\_SESSION['login\_time']: Calcula tiempo transcurrido desde login

session\_destroy(): Elimina todos los datos de sesión

CURDATE(): Función MySQL para fecha actual

htmlspecialchars(): Previene XSS escapando caracteres especiales

??: Operador ternario para valores por defecto

## Despliegue local:

Crear archivo dashboard.php1.

Acceder sin login (debe redirigir a login.php)2.

Hacer login y verificar acceso al dashboard3.

Probar navegación y funcionalidades4.

Verificar que muestra datos correctos del usuario5.

# Ejercicio 16: Sistema de Logout y Destrucción de Sesiones

## ¿Qué es y para qué sirve?

El logout es el proceso de cerrar sesión de forma segura, eliminando todos los datos de sesión y cookies para proteger la cuenta del usuario.

## Código completo - logout.php:

Verificación inicial:

```
<?php  
session_start();  
  
// Verificar que hay una sesión activa  
if (!isset($_SESSION['usuario_id'])) {  
    header('Location: login.php');  
    exit();
```

```
}
```

```
require_once 'conexion.php';
```

Limpiar token "recordar" de la base de datos:

```
// Si existe token de "recordar", eliminarlo de la base de datos if (isset($_COOKIE['remember_token'])) {  
    $sql = "UPDATE usuarios SET remember_token = NULL WHERE id = ?"; $stmt = $conexion->prepare($sql);  
    $stmt->bind_param("i", $_SESSION['usuario_id']);  
    $stmt->execute();  
    $stmt->close();
```

```
// Eliminar cookie del navegador  
setcookie('remember_token', '', time() - 3600, '/', false, true); }
```

Registrar logout en log (opcional):

```
// Opcional: Registrar logout en tabla de logs
```

**¿Cómo funciona?**

Destruye la sesión activa, elimina cookies de "recordar", limpia datos temporales y redirige al usuario a la página de login.



```
$conexion->prepare($sql_log);
$ip = $_SERVER['REMOTE_ADDR'] ?? 'unknown';
$user_agent = $_SERVER['HTTP_USER_AGENT'] ?? 'unknown';
$stmt_log->bind_param("iss", $_SESSION['usuario_id'], $ip, $user_agent);
$stmt_log->execute();
$stmt_log->close();
```

Destruir sesión completamente:

```
// Guardar nombre para mensaje de despedida
$nombre_usuario = $_SESSION['usuario_nombre'] ?? 'Usuario';
```

```
// Eliminar todas las variables de sesión
$_SESSION = array();
```

```
// Eliminar cookie de sesión si existe
if (ini_get("session.use_cookies")) {
    $params = session_get_cookie_params();
    setcookie(session_name(), "", time() - 42000,
        $params["path"], $params["domain"],
        $params["secure"], $params["httponly"])
```

```
);  
}  
  
// Destruir la sesión  
session_destroy();  
  
$conexion->close();
```

## Página de confirmación:

```
<!DOCTYPE html>  
<html lang="es">  
<head>  
<meta charset="UTF-8">  
<meta name="viewport" content="width=device-width, initial-scale=1.0">  
<title>Sesión Cerrada</title>  
<style>  
body {  
font-family: Arial, sans-serif;  
background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
```

```
min-height: 100vh;  
display: flex;  
align-items: center;  
justify-content: center;  
margin: 0;  
}  
.logout-container {  
background: white;  
padding: 3rem;  
border-radius: 10px;  
box-shadow: 0 10px 25px rgba(0,0,0,0.2);  
text-align: center;  
max-width: 400px;  
width: 90%;  
}  
.logout-icon {  
font-size: 4rem;  
color: #28a745;  
margin-bottom: 1rem;  
}  
h1 {  
color: #333;
```

```
margin-bottom: 1rem;
}
p {
color: #666;
margin-bottom: 2rem;
line-height: 1.6;
}
.btn-group {
display: flex;
gap: 1rem;
justify-content: center;
}
.btn {
padding: 0.75rem 1.5rem;
border: none;
border-radius: 5px;
text-decoration: none;
font-weight: bold;
transition: background-color 0.3s;
}
.btn-primary {
background: #007bff;
```

```
color: white;  
}  
.btn-primary:hover {  
background: #0056b3;  
}  
.btn-secondary {  
background: #6c757d;  
color: white;  
}  
.btn-secondary:hover {  
background: #545b62;  
}  
.security-note {  
background: #f8f9fa;  
border-left: 4px solid #28a745;  
padding: 1rem;  
margin-top: 2rem;  
text-align: left;  
font-size: 0.9rem;  
}  
  
</head>
```

```
<body>
<div class="logout-container">
<div class="logout-icon">✓ </div>
<h1>Sesión Cerrada</h1>
<p>¡Hasta luego, <?php echo htmlspecialchars($nombre_usuario); ?>!</p>
<p>Tu sesión ha sido cerrada de forma segura. Todos los datos temporales han sido eliminados.</p>
<div class="btn-group">
<a href="login.php" class="btn btn-primary">Iniciar Sesión</a>
<a href="index.html" class="btn btn-secondary">Página Principal</a>
</div>

<div class="security-note">
<strong>Nota de Seguridad:</strong><br>
• Tu sesión ha sido completamente eliminada<br>
• Las cookies de "recordar" han sido borradas<br>
• Es seguro cerrar el navegador
</div>
</div>

<script>
// Opcional: Redirigir automáticamente después de 10 segundos
setTimeout(function() {
```

```
window.location.href = 'login.php';
}, 10000);

// Prevenir uso del botón "atrás" del navegador
history.pushState(null, null, location.href);
window.onpopstate = function () {
    history.go(1);
};
</script>
</body>
</html>
```

## Explicación línea por línea:

`$_SESSION = array();`: Vacía todas las variables de sesión

`session_get_cookie_params();`: Obtiene parámetros de cookie de sesión

`session_name();`: Nombre de la cookie de sesión

`time() - 42000;`: Fecha pasada para expirar cookie

`session_destroy();`: Destruye completamente la sesión

`history.pushState();`: Manipula historial del navegador

window.onpopstate: Previene navegación hacia atrás

### **Despliegue local:**

Crear archivo logout.php1.

Hacer login en el sistema2.

Acceder a logout.php desde dashboard3.

Verificar que redirige y muestra mensaje4.

Intentar acceder a dashboard (debe redirigir a login)5.

## **Ejercicio 17: Crear Tablas Relacionadas con Claves Foráneas**

### **¿Qué es y para qué sirve?**

Las claves foráneas establecen relaciones entre tablas, manteniendo la integridad referencial. Es como crear "enlaces" entre diferentes tipos de información.

### **¿Cómo funciona?**

Una clave foránea en una tabla hace referencia a la clave primaria de otra tabla, creando una relación padre-hijo que MySQL puede validar automáticamente.

## Ejemplo práctico: Sistema de Tienda Online

### Paso 1: Crear tabla de categorías (tabla padre)

```
CREATE TABLE categorias (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL UNIQUE,
    descripcion TEXT,
    activa BOOLEAN DEFAULT TRUE,
    fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

### Paso 2: Insertar categorías de ejemplo

```
INSERT INTO categorias (nombre, descripcion) VALUES
('Electrónicos', 'Dispositivos electrónicos y gadgets'),
('Ropa', 'Vestimenta y accesorios'),
```

('Hogar', 'Artículos para el hogar y decoración'),  
('Deportes', 'Equipamiento deportivo y fitness');

### Paso 3: Crear tabla de productos (tabla hija)

```
CREATE TABLE productos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(200) NOT NULL,
    descripcion TEXT,
    precio DECIMAL(10,2) NOT NULL,
    stock INT DEFAULT 0,
    categoria_id INT NOT NULL,
    fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    -- Definir clave foránea
    FOREIGN KEY (categoria_id) REFERENCES categorias(id)
    ON DELETE RESTRICT
    ON UPDATE CASCADE
);
```

### Paso 4: Insertar productos con relaciones

```
INSERT INTO productos (nombre, descripcion, precio, stock, categoria_id) VALUES
('iPhone 14', 'Smartphone Apple última generación', 999.99, 50, 1),
('Camiseta Nike', 'Camiseta deportiva 100% algodón', 29.99, 100, 2),
('Sofá 3 plazas', 'Sofá cómodo para sala de estar', 599.99, 10, 3),
('Pelota de fútbol', 'Pelota oficial FIFA', 39.99, 25, 4);
```

## Comandos especiales para claves foráneas:

### Verificar relaciones existentes:

```
SELECT
    TABLE_NAME,
    COLUMN_NAME,
    CONSTRAINT_NAME,
    REFERENCED_TABLE_NAME,
    REFERENCED_COLUMN_NAME
FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
WHERE REFERENCED_TABLE_SCHEMA = 'tienda_online';
```

## Agregar clave foránea a tabla existente:

```
ALTER TABLE productos  
ADD CONSTRAINT fk_producto_categoria  
FOREIGN KEY (categoria_id) REFERENCES categorias(id);
```

## Eliminar clave foránea:

```
ALTER TABLE productos  
DROP FOREIGN KEY fk_producto_categoria;
```

## Explicación línea por línea:

**FOREIGN KEY:** Define la columna como clave foránea

**REFERENCES:** Especifica tabla y columna referenciada

**ON DELETE RESTRICT:** Impide eliminar categoría si tiene productos

**ON UPDATE CASCADE:** Actualiza automáticamente si cambia ID de categoría

**INFORMATION\_SCHEMA:** Base de datos del sistema con metadatos

## Despliegue local:

Ejecutar comandos SQL en phpMyAdmin1.

Intentar insertar producto con categoria\_id inexistente (debe fallar)2.

Intentar eliminar categoría que tiene productos (debe fallar)3.

Verificar que las relaciones funcionan correctamente4.

# Ejercicio 18: Consultas JOIN - Unir Datos de Múltiples Tablas

## ¿Qué es y para qué sirve?

JOIN permite combinar datos de múltiples tablas relacionadas en una sola consulta. Es como "pegar" información de diferentes fuentes para obtener un resultado completo.

## ¿Cómo funciona?

JOIN busca coincidencias entre las claves foráneas y primarias, combinando las filas que cumplen la condición especificada.

## Tipos de JOIN con ejemplos prácticos:

### 1. INNER JOIN - Solo coincidencias exactas

Mostrar productos con sus categorías

-- Mostrar productos con sus categorías

SELECT

p.id,

p.nombre AS producto,

```
p.precio,  
c.nombre AS categoria  
FROM productos p  
INNER JOIN categorias c ON p.categoria_id = c.id;
```

### Resultado esperado:

id producto precio categoria

1 iPhone 14 999.99 Electrónicos

2 Camiseta 29.99 Ropa

## 2. LEFT JOIN - Todos los registros de la tabla izquierda

Mostrar todas las categorías, incluso sin productos

-- Mostrar todas las categorías, incluso sin productos

SELECT

c.nombre AS categoria,

COUNT(p.id) AS total\_productos,

```
COALESCE(AVG(p.precio), 0) AS precio_promedio  
FROM categorias c  
LEFT JOIN productos p ON c.id = p.categoria_id  
GROUP BY c.id, c.nombre;
```

### 3. RIGHT JOIN - Todos los registros de la tabla derecha

Mostrar todos los productos, incluso sin categoría válida

-- Mostrar todos los productos, incluso sin categoría válida

```
SELECT  
p.nombre AS producto,  
p.precio,  
COALESCE(c.nombre, 'Sin categoría') AS categoria  
FROM categorias c  
RIGHT JOIN productos p ON c.id = p.categoria_id;
```

### 4. FULL OUTER JOIN (simulado con UNION)

MySQL no tiene FULL OUTER JOIN, se simula con UNION

-- MySQL no tiene FULL OUTER JOIN, se simula con UNION

SELECT

p.nombre AS producto,

c.nombre AS categoria

FROM productos p

LEFT JOIN categorias c ON p.categoría\_id = c.id

UNION

SELECT

p.nombre AS producto,

c.nombre AS categoria

FROM productos p

RIGHT JOIN categorias c ON p.id = p.categoría\_id;

## Consultas JOIN avanzadas:

### JOIN con múltiples tablas:

Crear tabla de pedidos

```
-- Crear tabla de pedidos
```

```
CREATE TABLE pedidos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    usuario_id INT,
    fecha_pedido TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    total DECIMAL(10,2),
    FOREIGN KEY (usuario_id) REFERENCES usuarios(id)
);
```

```
Crear tabla de detalles de pedido
```

```
-- Crear tabla de detalles de pedido
```

```
CREATE TABLE pedido_detalles (
    id INT AUTO_INCREMENT PRIMARY KEY,
    pedido_id INT,
    producto_id INT,
    cantidad INT,
    precio_unitario DECIMAL(10,2),
    FOREIGN KEY (pedido_id) REFERENCES pedidos(id),
    FOREIGN KEY (producto_id) REFERENCES productos(id)
```

);

Consulta con 4 tablas relacionadas

-- Consulta con 4 tablas relacionadas

SELECT

u.nombre AS cliente,  
p.fecha\_pedido,  
prod.nombre AS producto,  
pd.cantidad,  
pd.precio\_unitario,  
c.nombre AS categoria

FROM usuarios u

INNER JOIN pedidos p ON u.id = p.usuario\_id

INNER JOIN pedido\_detalles pd ON p.id = pd.pedido\_id

INNER JOIN productos prod ON pd.producto\_id = prod.id

INNER JOIN categorias c ON prod.categoría\_id = c.id

ORDER BY p.fecha\_pedido DESC;

## Funciones especiales con JOIN:

### COALESCE - Manejar valores NULL:

```
SELECT
    p.nombre,
    COALESCE(c.nombre, 'Sin categoría') AS categoria,
    COALESCE(p.stock, 0) AS stock_disponible
FROM productos p
LEFT JOIN categorias c ON p.categoria_id = c.id;
```

### CASE con JOIN - Lógica condicional:

```
SELECT
    p.nombre,
    p.precio,
    c.nombre AS categoria,
    CASE
```

```
WHEN p.stock > 50 THEN 'Stock Alto'  
WHEN p.stock > 10 THEN 'Stock Medio'  
WHEN p.stock > 0 THEN 'Stock Bajo'  
ELSE 'Sin Stock'  
END AS estado_stock  
FROM productos p  
INNER JOIN categorias c ON p.categoría_id = c.id;
```

## Subconsultas con JOIN:

```
-- Productos más caros que el promedio de su categoría  
SELECT  
    p.nombre,  
    p.precio,  
    c.nombre AS categoria,  
    (SELECT AVG(precio) FROM productos WHERE categoria_id = c.id) AS precio_promedio_categoria  
FROM productos p  
INNER JOIN categorias c ON p.categoría_id = c.id  
WHERE p.precio > (  
    SELECT AVG(precio)  
    FROM productos p2
```

```
WHERE p2.categoría_id = p.categoría_id  
);
```

## Explicación línea por línea:

**INNER JOIN:** Solo filas con coincidencias en ambas tablas

**LEFT JOIN:** Todas las filas de la tabla izquierda + coincidencias

**ON:** Condición de unión entre tablas

**COALESCE:** Devuelve el primer valor no NULL

**GROUP BY:** Agrupa resultados para funciones de agregación

**UNION:** Combina resultados de múltiples consultas

## Despliegue local:

Crear las tablas con datos de ejemplo1.

Ejecutar cada tipo de JOIN por separado2.

Comparar resultados y entender las diferencias3.

Probar consultas con múltiples tablas4.

## **Ejercicio 19: Funciones de Agregación y GROUP BY Avanzado**



## ¿Qué es y para qué sirve?

Las funciones de agregación procesan grupos de filas para devolver un solo valor. Son esenciales para generar reportes, estadísticas y análisis de datos.

## ¿Cómo funciona?

GROUP BY agrupa filas con valores similares, y las funciones de agregación (COUNT, SUM, AVG, etc.) calculan valores sobre cada grupo.

## Funciones de agregación básicas:

### 1. COUNT - Contar registros

-- Contar productos por categoría

```
SELECT
    c.nombre AS categoria,
    COUNT(p.id) AS total_productos
FROM categorias c
LEFT JOIN productos p ON c.id = p.categoria_id
GROUP BY c.id, c.nombre
```

```
ORDER BY total_productos DESC;
```

## 2. SUM - Sumar valores

-- Valor total del inventario por categoría

```
SELECT  
c.nombre AS categoria,  
COUNT(p.id) AS productos,  
SUM(p.precio * p.stock) AS valor_inventario,  
SUM(p.stock) AS stock_total  
FROM categorias c  
INNER JOIN productos p ON c.id = p.categoria_id  
GROUP BY c.id, c.nombre  
ORDER BY valor_inventario DESC;
```

## 3. AVG - Promedio

-- Precio promedio por categoría

```
SELECT  
c.nombre AS categoria,
```

```
ROUND(AVG(p.precio), 2) AS precio_promedio,  
MIN(p.precio) AS precio_minimo,  
MAX(p.precio) AS precio_maximo  
FROM categorias c  
INNER JOIN productos p ON c.id = p.categoria_id  
GROUP BY c.id, c.nombre;
```

## Funciones especiales avanzadas:

### 4. GROUP\_CONCAT - Concatenar valores

-- Listar todos los productos de cada categoría

```
SELECT  
c.nombre AS categoria,  
COUNT(p.id) AS total,  
GROUP_CONCAT(p.nombre ORDER BY p.precio DESC SEPARATOR ' | ') AS productos  
FROM categorias c  
INNER JOIN productos p ON c.id = p.categoria_id  
GROUP BY c.id, c.nombre;
```

## 5. HAVING - Filtrar grupos

-- Categorías con más de 2 productos y precio promedio > 100

```
SELECT
    c.nombre AS categoria,
    COUNT(p.id) AS total_productos,
    ROUND(AVG(p.precio), 2) AS precio_promedio
FROM categorias c
INNER JOIN productos p ON c.id = p.categoria_id
GROUP BY c.id, c.nombre
HAVING COUNT(p.id) > 2 AND AVG(p.precio) > 100;
```

## Consultas con múltiples niveles de agrupación:

### 6. GROUP BY con ROLLUP - Subtotales

-- Resumen con subtotales por categoría y total general

```
SELECT
```

```
COALESCE(c.nombre, 'TOTAL GENERAL') AS categoria,  
COUNT(p.id) AS productos,  
SUM(p.stock) AS stock_total,  
ROUND(SUM(p.precio * p.stock), 2) AS valor_inventario  
FROM categorias c  
INNER JOIN productos p ON c.id = p.categoría_id  
GROUP BY c.nombre WITH ROLLUP;
```

## 7. Subconsultas con agregación

```
-- Productos con precio superior al promedio de su categoría  
SELECT  
    p.nombre,  
    p.precio,  
    c.nombre AS categoria,  
    (SELECT ROUND(AVG(precio), 2)  
     FROM productos  
    WHERE categoria_id = p.categoría_id) AS promedio_categoria,  
    ROUND(p.precio - (SELECT AVG(precio)  
                      FROM productos  
                     WHERE categoria_id = p.categoría_id), 2) AS diferencia
```

```
FROM productos p
INNER JOIN categorias c ON p.categoria_id = c.id
WHERE p.precio > (SELECT AVG(precio)
FROM productos p2
WHERE p2.categoria_id = p.categoria_id);
```

## Funciones de ventana (Window Functions):

### 8. ROW\_NUMBER y RANK

```
-- Ranking de productos por precio dentro de cada categoría
SELECT
p.nombre,
p.precio,
c.nombre AS categoria,
ROW_NUMBER() OVER (PARTITION BY c.id ORDER BY p.precio DESC) AS ranking,
RANK() OVER (PARTITION BY c.id ORDER BY p.precio DESC) AS rank_precio
FROM productos p
INNER JOIN categorias c ON p.categoria_id = c.id
ORDER BY c.nombre, ranking;
```

## **9. Análisis de percentiles**

-- Análisis estadístico por categoría

```
SELECT
    c.nombre AS categoria,
    COUNT(p.id) AS total_productos,
    MIN(p.precio) AS precio_min,
    MAX(p.precio) AS precio_max,
    ROUND(AVG(p.precio), 2) AS precio_promedio,
    ROUND(STDDEV(p.precio), 2) AS desviacion_estandar
FROM categorias c
INNER JOIN productos p ON c.id = p.categoría_id
GROUP BY c.id, c.nombre
ORDER BY precio_promedio DESC;
```

## **Consultas de análisis temporal:**

### **10. Agrupación por fechas**

-- Productos creados por mes

SELECT

```
YEAR(fecha_creacion) AS año,  
MONTH(fecha_creacion) AS mes,  
MONTHNAME(fecha_creacion) AS nombre_mes,  
COUNT(*) AS productos_creados,  
ROUND(AVG(precio), 2) AS precio_promedio_mes  
FROM productos  
GROUP BY YEAR(fecha_creacion), MONTH(fecha_creacion)  
ORDER BY año DESC, mes DESC;
```

## Funciones especiales útiles:

### COALESCE con agregación:

SELECT

```
c.nombre AS categoria,  
COALESCE(COUNT(p.id), 0) AS productos,
```

```
COALESCE(ROUND(AVG(p.precio), 2), 0) AS precio_promedio,  
COALESCE(SUM(p.stock), 0) AS stock_total  
FROM categorias c  
LEFT JOIN productos p ON c.id = p.categoría_id  
GROUP BY c.id, c.nombre;
```

## CASE con GROUP BY:

-- Clasificar productos por rango de precio

```
SELECT  
CASE  
WHEN precio < 50 THEN 'Económico'  
WHEN precio < 200 THEN 'Medio'  
WHEN precio < 500 THEN 'Premium'  
ELSE 'Lujo'  
END AS rango_precio,  
COUNT(*) AS cantidad,  
ROUND(AVG(precio), 2) AS precio_promedio,  
MIN(precio) AS precio_min,  
MAX(precio) AS precio_max  
FROM productos
```

```
GROUP BY
CASE
WHEN precio < 50 THEN 'Económico'
WHEN precio < 200 THEN 'Medio'
WHEN precio < 500 THEN 'Premium'
ELSE 'Lujo'
END
ORDER BY precio_promedio;
```

## Explicación línea por línea:

**GROUP BY:** Agrupa filas con valores similares

**HAVING:** Filtra grupos (como WHERE pero para grupos)

**ROLLUP:** Genera subtotales automáticamente

**PARTITION BY:** Divide datos en ventanas para funciones de ventana

**STDDEV:** Calcula desviación estándar

**MONTHNAME:** Devuelve nombre del mes

**ROW\_NUMBER:** Asigna número secuencial a cada fila

## **Despliegue local:**

Insertar más datos de ejemplo en las tablas1.

Ejecutar cada consulta por separado2.

Analizar los resultados y entender las diferencias3.

Modificar las consultas para experimentar4.

# **Ejercicio 20: Subconsultas y Consultas Anidadas Avanzadas**

**¿Qué es y para qué sirve?** Las subconsultas son consultas dentro de otras consultas. Permiten realizar operaciones complejas dividiendo el problema en partes más pequeñas y manejables.

**¿Cómo funciona?** Una subconsulta se ejecuta primero y su resultado se usa en la consulta principal. Pueden estar en SELECT,

WHERE, FROM o HAVING.

## Tipos de subconsultas con ejemplos prácticos:

### 1. Subconsultas en WHERE - Filtrado avanzado

-- Productos más caros que el precio promedio general

```
SELECT
    nombre,
    precio,
    (precio - (SELECT AVG(precio) FROM productos)) AS diferencia_promedio
FROM productos
WHERE precio > (SELECT AVG(precio) FROM productos)
ORDER BY precio DESC;
```

### 2. Subconsultas correlacionadas

-- Productos más caros que el promedio de su categoría

```
SELECT
    p.nombre,
    p.precio,
```

```
c.nombre AS categoria  
FROM productos p  
INNER JOIN categorias c ON p.categoría_id = c.id  
WHERE p.precio > (  
    SELECT AVG(p2.precio)  
    FROM productos p2  
    WHERE p2.categoría_id = p.categoría_id  
);
```

### 3. Subconsultas con EXISTS

-- Categorías que tienen productos en stock

```
SELECT  
    c.nombre AS categoria,  
    c.descripcion  
FROM categorias c  
WHERE EXISTS (  
    SELECT 1  
    FROM productos p  
    WHERE p.categoría_id = c.id  
    AND p.stock > 0
```

);

## 4. Subconsultas con NOT EXISTS

-- Categorías sin productos

SELECT

c.nombre AS categoria\_sin\_productos,

c.descripcion

FROM categorias c

WHERE NOT EXISTS (

SELECT 1

FROM productos p

WHERE p.categoria\_id = c.id

);

## Subconsultas con operadores especiales:

### 5. IN y NOT IN

-- Productos de categorías específicas

```
SELECT
    nombre,
    precio
FROM productos
WHERE categoria_id IN (
    SELECT id
    FROM categorias
    WHERE nombre IN ('Electrónicos', 'Deportes')
);
```

-- Productos que NO están en categorías específicas

```
SELECT
    nombre,
    precio
FROM productos
WHERE categoria_id NOT IN (
    SELECT id
    FROM categorias
    WHERE nombre IN ('Ropa', 'Hogar')
    AND id IS NOT NULL -- Importante con NOT IN
);
```

## 6. ANY y ALL

-- Productos más caros que CUALQUIER producto de electrónicos

```
SELECT
    nombre,
    precio
FROM productos
WHERE precio > ANY (
    SELECT precio
    FROM productos p
    INNER JOIN categorias c ON p.categoría_id = c.id
    WHERE c.nombre = 'Electrónicos'
);
```

-- Productos más caros que TODOS los productos de ropa

```
SELECT
    nombre,
    precio
FROM productos
WHERE precio > ALL (
```

```
SELECT precio  
FROM productos p  
INNER JOIN categorias c ON p.categoría_id = c.id  
WHERE c.nombre = 'Ropa'  
);
```

## Subconsultas en SELECT:

### 7. Campos calculados con subconsultas

-- Información detallada de productos con estadísticas

```
SELECT  
p.nombre,  
p.precio,  
c.nombre AS categoria,  
p.stock,  
(SELECT COUNT(*) FROM productos) AS total_productos,  
(SELECT AVG(precio) FROM productos) AS precio_promedio_general,  
(SELECT AVG(precio)  
FROM productos p2  
WHERE p2.categoría_id = p.categoría_id) AS precio_promedio_categoria,
```

```
(SELECT COUNT(*)
FROM productos p3
WHERE p3.categoría_id = p.categoría_id) AS productos_en_categoria
FROM productos p
INNER JOIN categorías c ON p.categoría_id = c.id;
```

## Subconsultas en FROM (Tablas derivadas):

### 8. Consultas sobre resultados agregados

```
-- Top 3 categorías por valor de inventario
SELECT
    categoría,
    productos,
    valor_inventario,
    RANK() OVER (ORDER BY valor_inventario DESC) AS ranking
FROM (
    SELECT
        c.nombre AS categoría,
        COUNT(p.id) AS productos,
        SUM(p.precio * p.stock) AS valor_inventario
```

```
FROM categorias c
INNER JOIN productos p ON c.id = p.categoria_id
GROUP BY c.id, c.nombre
) AS resumen_categorias
WHERE valor_inventario > 1000
ORDER BY valor_inventario DESC
LIMIT 3;
```

## Subconsultas complejas con múltiples niveles:

### 9. Análisis de productos por percentiles

-- Productos en el top 25% de precio de su categoría

```
SELECT
p.nombre,
p.precio,
c.nombre AS categoria,
percentil.precio_75 AS precio_percentil_75
FROM productos p
INNER JOIN categorias c ON p.categoria_id = c.id
INNER JOIN (
```

```
SELECT
categoria_id,
ROUND(
(SELECT precio
FROM productos p2
WHERE p2.categoria_id = p1.categoria_id
ORDER BY precio DESC
LIMIT 1 OFFSET (
SELECT FLOOR(COUNT(*) * 0.25)
FROM productos p3
WHERE p3.categoria_id = p1.categoria_id
)
),2
) AS precio_75
FROM productos p1
GROUP BY categoria_id
) AS percentil ON p.categoria_id = percentil.categoria_id
WHERE p.precio >= percentil.precio_75;
```

## 10. Subconsultas con UNION

-- Reporte combinado de productos y categorías

SELECT

'Producto' AS tipo,  
nombre AS item,  
CAST(precio AS CHAR) AS valor,  
'Individual' AS nivel

FROM productos

WHERE stock > 0

UNION ALL

SELECT

'Categoría' AS tipo,  
nombre AS item,  
CAST(  
(SELECT COUNT(\*))  
FROM productos p  
WHERE p.categoria\_id = c.id) AS CHAR  
) AS valor,  
'Agrupado' AS nivel  
FROM categorias c

```
WHERE EXISTS (
    SELECT 1
    FROM productos p
    WHERE p.categoría_id = c.id
)
ORDER BY tipo, item;
```

## Funciones especiales con subconsultas:

### 11. CASE con subconsultas

```
-- Clasificación dinámica de productos
SELECT
    nombre,
    precio,
    CASE
        WHEN precio > (SELECT AVG(precio) * 1.5 FROM productos) THEN 'Premium'
        WHEN precio > (SELECT AVG(precio) FROM productos) THEN 'Estándar'
        WHEN precio > (SELECT AVG(precio) * 0.5 FROM productos) THEN 'Económico'
        ELSE 'Básico'
    END AS clasificacion,
```

```
ROUND(  
    (precio / (SELECT AVG(precio) FROM productos) * 100), 2  
) AS porcentaje_vs_promedio  
FROM productos  
ORDER BY precio DESC;
```

## 12. Subconsultas con funciones de fecha

```
-- Productos creados en el último mes con más actividad  
SELECT  
    p.nombre,  
    p.fecha_creacion,  
    c.nombre AS categoria  
FROM productos p  
INNER JOIN categorias c ON p.categoria_id = c.id  
WHERE MONTH(p.fecha_creacion) = (  
    SELECT MONTH(fecha_creacion) AS mes_activo  
    FROM productos  
    GROUP BY YEAR(fecha_creacion), MONTH(fecha_creacion)  
    ORDER BY COUNT(*) DESC  
    LIMIT 1
```

```
)  
AND YEAR(p.fecha_creacion) = (  
SELECT YEAR(fecha_creacion)  
FROM productos  
GROUP BY YEAR(fecha_creacion), MONTH(fecha_creacion)  
ORDER BY COUNT(*) DESC  
LIMIT 1  
);
```

## Optimización de subconsultas:

### 13. Convertir subconsultas a JOIN (más eficiente)

-- Menos eficiente (subconsulta)

```
SELECT nombre, precio  
FROM productos  
WHERE categoria_id IN (  
SELECT id FROM categorias WHERE activa = TRUE  
);
```

-- Más eficiente (JOIN)

```
SELECT p.nombre, p.precio  
FROM productos p  
INNER JOIN categorias c ON p.categoría_id = c.id  
WHERE c.activa = TRUE;
```

## Explicación línea por línea:

**EXISTS:** Verifica si la subconsulta devuelve al menos una fila

**NOT EXISTS:** Verifica si la subconsulta no devuelve filas

**ANY:** Compara con cualquier valor de la subconsulta

**ALL:** Compara con todos los valores de la subconsulta

**OFFSET:** Salta un número específico de filas

**CAST:** Convierte un tipo de dato a otro

## Despliegue local:

Crear datos de ejemplo suficientes para probar1.

Ejecutar subconsultas simples primero2.

Probar subconsultas correlacionadas3.

Comparar rendimiento entre subconsultas y JOINs4.

# Implementación de transacciones

Veamos cómo implementar transacciones con **MySQLi** y **PDO** para garantizar la integridad de operaciones complejas.

## Con MySQLi

```
<?php  
// Iniciar transacción  
$conexion->begin_transaction();  
  
try {  
    // Primera consulta
```

```
$stmt1 = $conexion->prepare( "UPDATE cuentas
SET saldo = saldo - ?
WHERE id = ?"
);
$stmt1->bind_param("di", $monto, $cuenta_origen); $stmt1->execute();

// Segunda consulta
$stmt2 = $conexion->prepare( "UPDATE cuentas
SET saldo = saldo + ?
WHERE id = ?"
);
$stmt2->bind_param("di", $monto, $cuenta_destino); $stmt2->execute();

// Si todo está bien, confirmar $conexion->commit();
echo "Transferencia exitosa";
} catch (Exception $e) {
// Si hay error, revertir
$conexion->rollback();
echo "Error: " .
$e->getMessage();
}
?>
```

## Con PDO

```
<?php
try {
    // Iniciar transacción
    $pdo->beginTransaction();

    // Restar de cuenta origen
    $stmt1 = $pdo->prepare(
        "UPDATE cuentas
        SET saldo = saldo - :monto WHERE id = :origen
        AND saldo >= :monto"
    );
    $stmt1->execute([
        ':monto' => $monto,
        ':origen' => $cuenta_origen ]);

    // Verificar que se actualizó
    if ($stmt1->rowCount() == 0) {
        throw new Exception(
            "Saldo insuficiente"
        );
    }
}
```

```
// Sumar a cuenta destino
$stmt2 = $pdo->prepare(
"UPDATE cuentas
SET saldo = saldo + :monto WHERE id = :destino"
);
$stmt2->execute([
':monto' => $monto,
':destino' => $cuenta_destino ]);

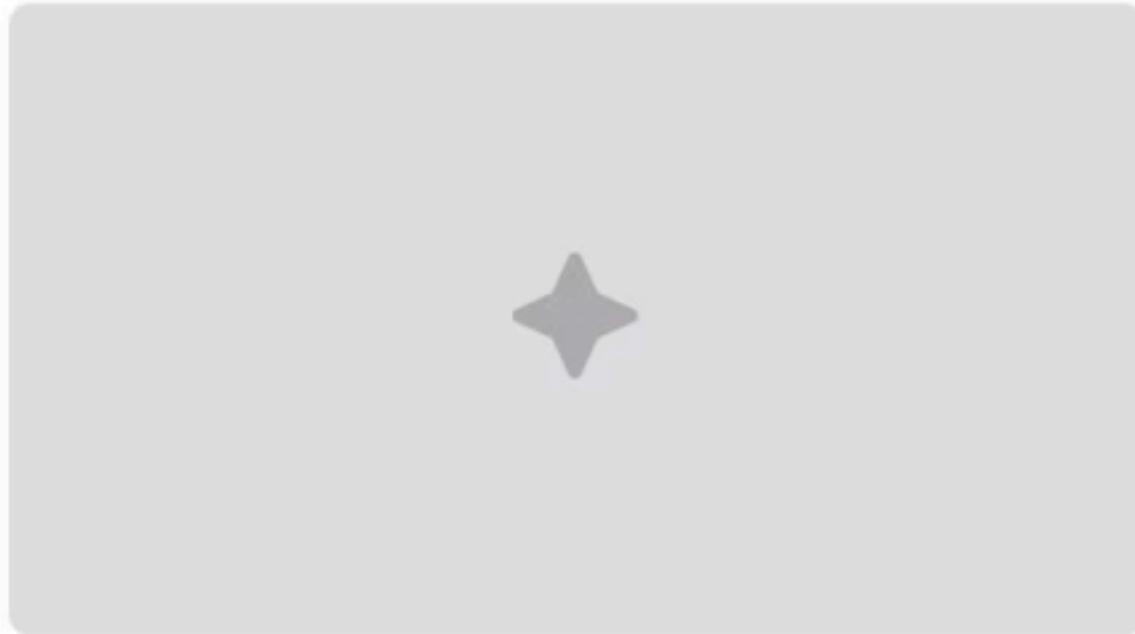
// Confirmar transacción
$pdo->commit();
echo "Transferencia completada";
} catch (Exception $e) {
// Revertir en caso de error $pdo->rollBack();
echo "Error: " .
$e->getMessage();
}
?>
```

**CUIDADO:** No hagas transacciones demasiado largas o que incluyan operaciones lentas, ya que pueden **bloquear tablas** por mucho

tiempo.

# Paginación de resultados

La paginación es esencial cuando trabajamos con grandes conjuntos de datos, mejorando el **rendimiento y la usabilidad**.



```
<?php
require_once 'conexion.php';
$conexion = getConexionMysqli();

// Parámetros de paginación
$registrosPorPagina = 10;
$paginaActual = isset($_GET['pagina']) ? (int)$_GET['pagina'] : 1;

// Asegurar que la página sea válida
if ($paginaActual < 1) {
    $paginaActual = 1;
}

// Calcular el punto de inicio
$inicio = ($paginaActual - 1) * $registrosPorPagina;

// Consulta paginada
$sql = "SELECT * FROM productos
        ORDER BY nombre ASC
        LIMIT ?, ?";
$stmt = $conexion->prepare($sql);
$stmt->bind_param("ii", $inicio, $registrosPorPagina);
```

```
$stmt->execute();
$resultado = $stmt->get_result();

// Mostrar resultados
echo "<ul>";
while ($fila = $resultado->fetch_assoc()) {
    echo "<li>" . $fila['nombre'] . " - $" . $fila['precio'] . "</li>";
}
echo "</ul>";

// Obtener el número total de registros
$totalRegistros = $conexion->query("SELECT COUNT(*) FROM productos")->fetch_row()[0];

// Calcular el número total de páginas
$totalPaginas = ceil($totalRegistros / $registrosPorPagina);

// Generar enlaces de paginación
echo "<div class='pagination'>";

if ($paginaActual > 1) {
    echo "<a href='?pagina=" . ($paginaActual - 1) . "'>Anterior</a>";
}

```

```
for ($i = 1; $i <= $totalPaginas; $i++) {  
    if ($i == $paginaActual) {  
        echo "<span class='current'>$i</span>";  
    } else {  
        echo "<a href='?pagina=$i'$>$i</a>";  
    }  
}  
  
if ($paginaActual < $totalPaginas) {  
    echo "<a href='?pagina=". ($paginaActual + 1) . "'>Siguiente</a>";  
}  
  
echo "</div>";  
?>
```

## Búsqueda y filtrado de resultados

Implementar funcionalidades de **búsqueda y filtrado** permite a los usuarios encontrar rápidamente la información que necesitan.

### Búsqueda básica

```
<?php
require_once 'conexion.php'; $conexion = getConexionMysqli();

// Obtener término de búsqueda $busqueda = isset($_GET['q']) ? trim($_GET['q']) : "";

// Consulta con búsqueda
if (!empty($busqueda)) {
    $sql = "SELECT * FROM productos WHERE nombre LIKE ?
    OR descripcion LIKE ?";
    // Comodines % para buscar // en cualquier parte
    $termino = "%$busqueda%";
    $stmt = $conexion->prepare($sql); $stmt->bind_param("ss",
    $termino, $termino);
} else {
    // Sin búsqueda, mostrar todos $sql = "SELECT * FROM productos"; $stmt = $conexion->prepare($sql); }

$stmt->execute();
$resultado = $stmt->get_result();

// Mostrar resultados
```

```
if ($resultado->num_rows > 0) { echo "<ul>";
    while ($fila =
$resultado->fetch_assoc()) { echo "<li>" .
$fila['nombre'] .
" - $" .
$fila['precio'] .
"</li>";
}
echo "</ul>";
echo $resultado->num_rows . " resultados encontrados"; } else {
echo "No se encontraron
resultados";
}
?>
```

## Búsqueda con múltiples filtros

```
<?php
// Inicializar arrays
$condiciones = [];
$parametros = [];
$tipos = "";
```

```
// Construir condiciones
if (!empty($_GET['q'])) {
    $condiciones[] = "(nombre LIKE ?
        OR descripcion LIKE ?)";
    $termino = "%" . $_GET['q'] . "%";
    $parametros[] = $termino;
    $parametros[] = $termino;
    $tipos .= 'ss';
}
```

```
if (!empty($_GET['categoria'])) {
    $condiciones[] =
        "categoria_id = ?";
    $parametros[] =
        $_GET['categoria'];
    $tipos .= 'i';
}
```

```
if (isset($_GET['precio_min'])) {
    $condiciones[] = "precio >= ?";
    $parametros[] =
```

```
$_GET['precio_min'];
$tipos .= 'd';
}

if (isset($_GET['precio_max'])) {
$condiciones[] = "precio <= ?";
$parametros[] =
$_GET['precio_max'];
$tipos .= 'd';
}

// Construir SQL
$sql = "SELECT * FROM productos";
if (count($condiciones) > 0) {
$sql .= " WHERE ".
implode(' AND ', $condiciones);
}

// Ejecutar consulta
$stmt = $conexion->prepare($sql);
if (!empty($parametros)) {
$stmt->bind_param($tipos,
```

```
...$parametros);  
}  
$stmt->execute();  
?>
```

## Relaciones entre tablas: JOIN

Las bases de datos relacionales permiten **vincular información entre tablas** mediante operaciones JOIN. Esto es fundamental para estructuras de datos complejas.

### INNER JOIN

Retorna registros cuando hay **coincidencias en ambas tablas**.

```
SELECT p.nombre, c.nombre AS categoria  
FROM productos p  
INNER JOIN categorias c  
ON p.categoria_id = c.id;
```

### LEFT JOIN

Retorna **todos los registros de la tabla izquierda** y los que coinciden de la derecha.

```
SELECT p.nombre,
```

```
IFNULL(c.nombre, 'Sin categoría') AS categoria  
FROM productos p  
LEFT JOIN categorias c  
ON p.categoria_id = c.id;
```

## **RIGHT JOIN**

Retorna todos los registros de la **tabla derecha** y los que coinciden de la izquierda.

```
SELECT p.nombre, c.nombre AS categoria  
FROM productos p  
RIGHT JOIN categorias c  
ON p.categoria_id = c.id;
```

## Estructura de tablas relacionadas

```
CREATE TABLE categorias (  
id INT AUTO_INCREMENT PRIMARY KEY,  
nombre VARCHAR(100) NOT NULL  
);
```

```
CREATE TABLE productos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    precio DECIMAL(10,2) NOT NULL,
    categoria_id INT,
    FOREIGN KEY (categoria_id) REFERENCES
    categorias(id);
```

## Consultas avanzadas: Funciones y agrupación

MySQL ofrece **funciones de agregación y agrupación** para análisis de datos más sofisticados y generación de reportes.

### Funciones de agregación

**COUNT()** - Cuenta el número de filas

**SUM()** - Suma los valores de una columna

**AVG()** - Calcula el promedio de una columna

**MIN()** - Encuentra el valor mínimo

**MAX()** - Encuentra el valor máximo

```
SELECT
    COUNT(*) AS total_productos,
    AVG(precio) AS precio_promedio,
```

```
MIN(precio) AS precio_minimo,
    MAX(precio) AS precio_maximo,
    SUM(stock) AS stock_total
```

FROM productos;

### Agrupación con GROUP BY

Permite agrupar resultados según valores de una o más columnas:

```
SELECT c.nombre AS categoria,  
       COUNT(*) AS total_productos,  
       AVG(p.precio) AS precio_promedio  
  FROM productos p  
 JOIN categorias c  
   ON p.categoria_id = c.id  
 GROUP BY p.categoria_id
```

ORDER BY total\_productos DESC;

### Filtrado de grupos con HAVING

Similar a WHERE pero para filtrar grupos después de la agrupación:

```
SELECT c.nombre AS categoria,  
       COUNT(*) AS total_productos,
```

```
       AVG(p.precio) AS precio_promedio  
  FROM productos p  
 JOIN categorias c  
   ON p.categoria_id = c.id  
 GROUP BY p.categoria_id  
 HAVING total_productos > 5  
   AND precio_promedio > 100  
 ORDER BY precio_promedio DESC;
```

## Diseño de sistema de usuarios completo

Un sistema de gestión de usuarios robusto requiere un **diseño de base de datos bien pensado** que contemple seguridad, roles y

### permisos. **Tabla principal de usuarios**

```
CREATE TABLE usuarios (  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  nombre VARCHAR(100) NOT NULL,  
  apellido VARCHAR(100) NOT NULL,  
  email VARCHAR(100) NOT NULL UNIQUE,  
  clave VARCHAR(255) NOT NULL,
```

```
telefono VARCHAR(20),
fecha_nacimiento DATE,
genero ENUM('M', 'F', 'O'),
biografia TEXT,
foto_perfil VARCHAR(255),
fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
ultimo_acceso DATETIME,
estado ENUM('activo', 'inactivo', 'suspendido') DEFAULT 'activo',
token_recuperacion VARCHAR(100),
expiracion_token DATETIME,
intentos_fallidos TINYINT DEFAULT 0
);

```

;

## Tabla de permisos

```
CREATE TABLE permisos (
id INT AUTO_INCREMENT
PRIMARY KEY,
nombre VARCHAR(50)
NOT NULL UNIQUE,
descripcion TEXT
);
```

## Tabla de roles

```
CREATE TABLE roles (
id INT AUTO_INCREMENT
PRIMARY KEY,
nombre VARCHAR(50)
NOT NULL UNIQUE,
descripcion TEXT
```

## Relación usuario-rol

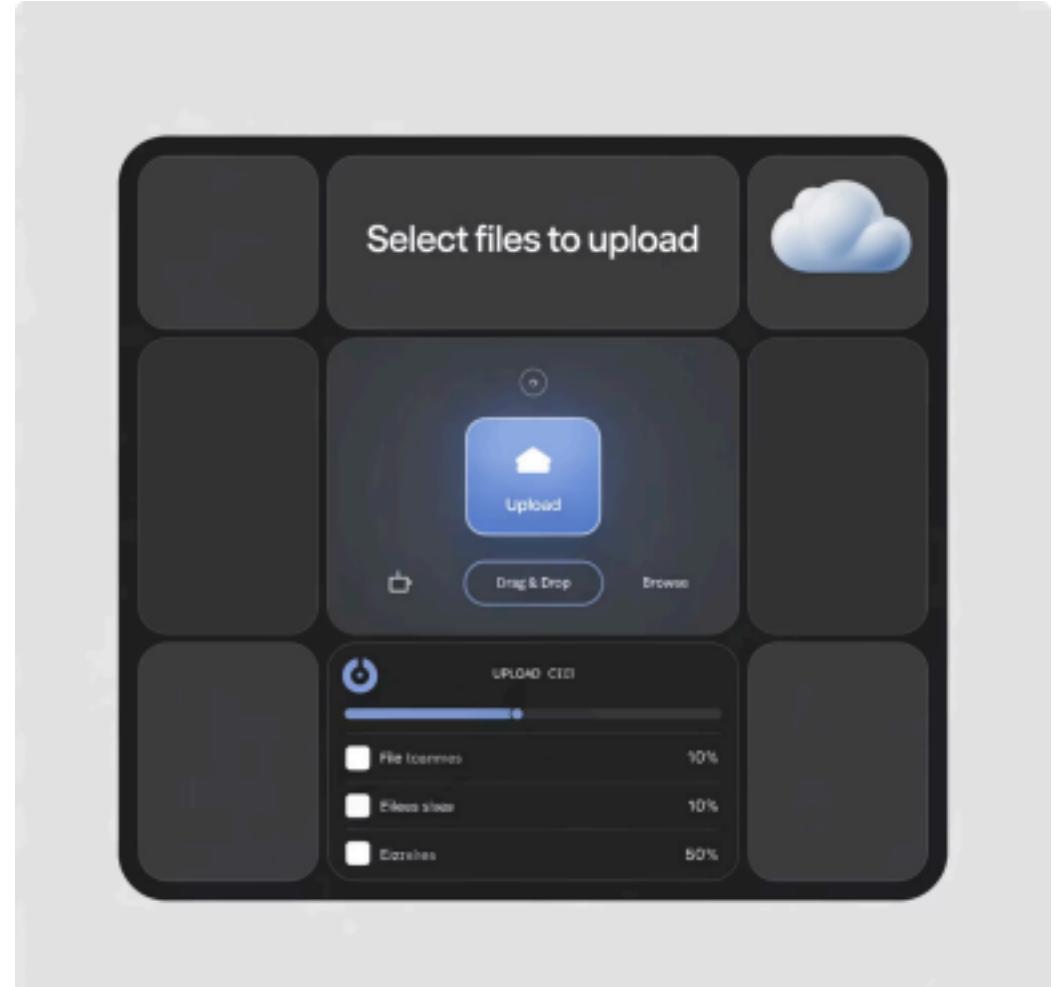
```
CREATE TABLE usuario_rol
(
usuario_id INT,
rol_id INT,
PRIMARY KEY
(usuario_id, rol_id),
FOREIGN KEY (usuario_id)
REFERENCES usuarios(id),
FOREIGN KEY (rol_id)
```

# Subida y manejo seguro de archivos

La subida de archivos es común en aplicaciones web, pero requiere un **manejo cuidadoso** para evitar vulnerabilidades de seguridad.

## Formulario HTML

```
<form action="subir.php"
method="post"
enctype="multipart/form-data">
<h3>Subir imagen de perfil</h3>
<div>
<label for="archivo">
Seleccionar imagen:
</label>
<input type="file"
name="archivo"
id="archivo"
accept="image/*">
```



```
<small>  
Formatos: JPG, PNG, GIF  
(Max: 2MB)  
</small>  
</div>  
<button type="submit">  
Subir Imagen  
</button>  
</form>
```

**Importante:** `enctype="multipart/form-data"` es **obligatorio** para subir archivos. `accept="image/*"` limita la selección pero no es una medida de seguridad real.

#### Puntos clave de seguridad:

- ✓ Verificar tamaño máximo del archivo
- ✓ Validar tipo MIME real del archivo
- ✓ Usar nombres únicos para archivos
- ✓ Almacenar fuera del directorio web público
- ✓ Establecer permisos correctos
- ✓ Procesar/optimizar imágenes antes de guardar

## Procesamiento seguro de archivos subidos

Implementaremos el procesamiento completo de archivos con **validación exhaustiva** y medidas de seguridad.

```
<?php  
session_start();
```

```
if (!isset($_SESSION['usuario_id'])) {
    header("Location: login.php");
    exit();
}

// Verificar si se envió un archivo
if (!isset($_FILES['archivo']) || 
    $_FILES['archivo']['error'] !== UPLOAD_ERR_OK) {
    echo "Error al subir el archivo";
    exit();
}

// Verificar tamaño máximo (2MB)
$max_size = 2 * 1024 * 1024; // 2MB en bytes
if ($_FILES['archivo']['size'] > $max_size) {
    echo "El archivo excede el tamaño máximo permitido (2MB)";
    exit();
}

// Verificar tipo de archivo
$allowed_types = ['image/jpeg', 'image/png', 'image/gif'];
$info = finfo_open(FILEINFO_MIME_TYPE);
```

```
$mime_type = finfo_file($finfo, $_FILES['archivo']['tmp_name']);
finfo_close($finfo);

if (!in_array($mime_type, $allowed_types)) {
    echo "Tipo de archivo no permitido. Solo se aceptan imágenes JPG, PNG y GIF";
    exit();
}

// Generar nombre único para el archivo
$extension = pathinfo($_FILES['archivo']['name'], PATHINFO_EXTENSION);
$nombre_archivo = uniqid('img_') . '.' . $extension;

// Directorio de destino
$directorio = 'uploads/perfiles/';

// Crear directorio si no existe
if (!is_dir($directorio)) {
    mkdir($directorio, 0755, true);
}

// Ruta completa del archivo
$ruta_archivo = $directorio . $nombre_archivo;
```

```
// Mover archivo del directorio temporal al destino final
if (move_uploaded_file($_FILES['archivo']['tmp_name'], $ruta_archivo)) {
    // Actualizar ruta de imagen en la base de datos
    require_once 'conexion.php';
    $conexion = getConexionMysqli();

    $stmt = $conexion->prepare(
        "UPDATE usuarios SET foto_perfil = ? WHERE id = ?"
    );
    $stmt->bind_param("si", $ruta_archivo, $_SESSION['usuario_id']);
    $stmt->execute();

    // Redireccionar
    header("Location: perfil.php?actualizado=1");
} else {
    echo "Error al guardar el archivo";
}
?>
```

## Procesamiento y optimización de imágenes

Cuando trabajamos con imágenes subidas por usuarios, es recomendable **procesarlas y optimizarlas** para mejorar rendimiento y seguridad.



```
<?php  
// Función para redimensionar imágenes con librería GD  
function redimensionarImagen($ruta_original, $ancho_max = 800, $alto_max = 600) {  
    // Obtener información de la imagen  
    list($ancho_orig, $alto_orig, $tipo) = getimagesize($ruta_original);  
  
    // Si la imagen ya es menor que el máximo, no hacer nada  
    if ($ancho_orig <= $ancho_max && $alto_orig <= $alto_max) {  
        return true;  
    }  
  
    // Calcular nuevas dimensiones manteniendo proporción  
    if ($ancho_orig > $alto_orig) {  
        $ancho_nuevo = $ancho_max;  
        $alto_nuevo = round($alto_orig * ($ancho_max / $ancho_orig));  
    } else {  
        $alto_nuevo = $alto_max;  
        $ancho_nuevo = round($ancho_orig * ($alto_max / $alto_orig));  
    }  
  
    // Crear imagen desde el original según el tipo  
    switch ($tipo) {
```

```
case IMAGETYPE_JPEG:  
$imagen = imagecreatefromjpeg($ruta_original);  
break;  
case IMAGETYPE_PNG:  
$imagen = imagecreatefrompng($ruta_original);  
break;  
case IMAGETYPE_GIF:  
$imagen = imagecreatefromgif($ruta_original);  
break;  
default:  
return false;  
}  
  
// Crear imagen nueva con las dimensiones calculadas  
$imagen_nueva = imagecreatetruecolor($ancho_nuevo, $alto_nuevo);  
  
// Preservar transparencia para PNG y GIF  
if ($tipo == IMAGETYPE_PNG || $tipo == IMAGETYPE_GIF) {  
imagealphablending($imagen_nueva, false);  
imagesavealpha($imagen_nueva, true);  
$transparente = imagecolorallocatealpha($imagen_nueva, 255, 255, 255, 127);  
imagefilledrectangle($imagen_nueva, 0, 0, $ancho_nuevo, $alto_nuevo,
```

```
$transparente);  
}  
  
// Redimensionar  
imagecopyresampled($imagen_nueva, $imagen, 0, 0, 0, 0,  
$ancho_nuevo, $alto_nuevo, $ancho_orig, $alto_orig);  
  
// Guardar la imagen según su tipo  
switch ($tipo) {  
    case IMAGETYPE_JPEG:  
        imagejpeg($imagen_nueva, $ruta_original, 85);  
        break;  
    case IMAGETYPE_PNG:  
        imagepng($imagen_nueva, $ruta_original, 8);  
        break;  
    case IMAGETYPE_GIF:  
        imagegif($imagen_nueva, $ruta_original);  
        break;  
}  
  
// Liberar memoria  
imagedestroy($imagen);
```

```
imagedestroy($imagen_nueva);

return true;
}

// Uso en el script de subida
if (move_uploaded_file($_FILES['archivo']['tmp_name'], $ruta_archivo)) {
    // Redimensionar la imagen
    redimensionarImagen($ruta_archivo, 1200, 1200);
    // Continuar con el resto del procesamiento...
}
?>
```

## Sistema CRUD completo: Interfaz de listado

Vamos a crear un

sistema CRUD completo para gestionar productos, comenzando por la interfaz de usuario con filtros y paginación.

```
<?php
require_once 'sesion.php';
require_once 'conexion.php';
$conexion = getConexionMysqli();
verificarSesion();
```

```
// Parámetros de paginación
$registrosPorPagina = 10;
$páginaActual = isset($_GET['pagina']) ? (int)$_GET['pagina'] : 1;
$inicio = ($páginaActual - 1) * $registrosPorPagina;

// Parámetros de filtrado
$categoría = isset($_GET['categoría']) ? $_GET['categoría'] : '';
$orden = isset($_GET['orden']) ? $_GET['orden'] : 'nombre_asc';

// Mapeo de opciones de ordenamiento
$ordenamientos = [
    'nombre_asc' => "ORDER BY p.nombre ASC",
    'nombre_desc' => "ORDER BY p.nombre DESC",
    'precio_asc' => "ORDER BY p.precio ASC",
    'precio_desc' => "ORDER BY p.precio DESC"
];

// Validar orden
if (!array_key_exists($orden, $ordenamientos)) {
    $orden = 'nombre_asc';
}
```

```
// Construir consulta base
$sql = "SELECT p.*, c.nombre as categoria
FROM productos p
LEFT JOIN categorias c ON p.categoria_id = c.id";
```

```
// Añadir filtro por categoría si existe
```

```
$parametros = [];
$tipos = '';
if (!empty($categoria)) {
    $sql .= " WHERE p.categoria_id = ?";
    $parametros[] = $categoria;
    $tipos .= 'i';
}

```

```
// Añadir ordenamiento
```

```
$sql .= " " . $ordenamientos[$orden];
```

```
// Añadir límite para paginación
```

```
$sql .= " LIMIT ?, ?";
$parametros[] = $inicio;
$parametros[] = $registrosPorPagina;
```

```
$tipos .= 'ii';

// Preparar y ejecutar consulta
$stmt = $conexion->prepare($sql);
if (!empty($parametros)) {
    $stmt->bind_param($tipos, ...$parametros);
}
$stmt->execute();
$resultado = $stmt->get_result();
?>
```

## Sistema CRUD: Formulario de creación/edición

Ahora implementaremos el **formulario que se usará tanto para crear como para editar** productos, con validación del lado del cliente y del servidor.

```
<?php
require_once 'sesion.php';
require_once 'conexion.php';
$conexion = getConexionMysqli();
verificarSesion();
```

```
// Determinar si es edición o creación
$edicion = false;
$producto = null;

if (isset($_GET['id'])) {
    $id = (int)$_GET['id'];
    $edicion = true;

// Obtener datos del producto
$stmt = $conexion->prepare(
    "SELECT p.*, c.nombre as categoria_nombre
    FROM productos p
    JOIN categorias c ON p.categoria_id = c.id
    WHERE p.id = ?"
);
$stmt->bind_param("i", $id);
$stmt->execute();
$resultado = $stmt->get_result();

if ($resultado->num_rows === 0) {
    header("Location: listar.php?error=no_encontrado");
}
```

```
exit();
}

$producto = $resultado->fetch_assoc();
}

// Obtener categorías para el select
$categorias = $conexion->query(
"SELECT id, nombre FROM categorias ORDER BY nombre"
);
?>

<!DOCTYPE html>
<html>
<head>
<title><?php echo $edicion ? 'Editar' : 'Crear'; ?> Producto</title>
<link rel="stylesheet" href="css/estilos.css">
</head>
<body>
<div class="contenedor">
<form method="post"
action=<?php echo $edicion ? 'actualizar.php?id='.$id : 'guardar.php'; ?>"
```

```
enctype="multipart/form-data"
class="form-producto">

<h2><?php echo $edicion ? 'Editar' : 'Crear'; ?> Producto</h2>

<div class="form-group">
<label for="nombre">Nombre:</label>
<input type="text" id="nombre" name="nombre"
value=<?php echo $edicion ? htmlspecialchars($producto['nombre']) : ">">
required>
</div>

<div class="form-group">
<label for="descripcion">Descripción:</label>
<textarea id="descripcion" name="descripcion" rows="4"><?php echo $edicion ? htmlspecialchars($producto['descripcion']) : ">"></textarea> </div>

<div class="form-row">
<div class="form-group col">
<label for="precio">Precio:</label>
<input type="number" id="precio" name="precio"
step="0.01" min="0"
```

```
value="<?php echo $edicion ? $producto['precio'] : ''; ?>"  
required>  
</div>  
  
<div class="form-group col">  
  <label for="stock">Stock:</label>  
  <input type="number" id="stock" name="stock"  
        min="0"  
        value="<?php echo $edicion ? $producto['stock'] : ''; ?>"  
        required>  
  </div>  
</div>  
  
<div class="form-group">  
  <label for="categoria_id">Categoría:</label>  
  <select id="categoria_id" name="categoria_id" required>  
    <option value="">Seleccionar categoría</option>  
    <?php while ($cat = $categorias->fetch_assoc()): ?>  
    <option value="<?php echo $cat['id']; ?>"  
          <?php echo ($edicion && $producto['categoria_id'] == $cat['id']) ? 'selected' : '' ; ?>>  
    <?php echo htmlspecialchars($cat['nombre']); ?>  
    </option>
```

```
<?php endwhile; ?>
</select>
</div>

<div class="form-actions">
<button type="submit" class="btn btn-primary">
<?php echo $edicion ? 'Actualizar' : 'Guardar'; ?> Producto
</button>
<a href="listar.php" class="btn btn-secondary">Cancelar</a>
</div>
</form>
</div>
</body>
</html>
```

## Sistema CRUD: Procesamiento de datos

Implementaremos el

**procesamiento completo** para crear, actualizar y eliminar productos con validación exhaustiva y transacciones.

### Guardar nuevo producto

```
<?php
```

```
require_once 'sesion.php';
require_once 'conexion.php';
require_once 'funciones.php';

verificarSesion();
$conexion = getConexionMysqli();

if ($_SERVER['REQUEST_METHOD'] === 'POST') { $errores = [];

// Sanitizar y validar datos
$nombre = sanitizarTexto(
$_POST['nombre']
);
$descripcion = sanitizarTexto(
$_POST['descripcion']
);
$precio = filter_var(
$_POST['precio'],
FILTER_VALIDATE_FLOAT
);
$stock = filter_var(
$_POST['stock'],
```

```
FILTER_VALIDATE_INT
);
$categoria_id = filter_var(
$_POST['categoria_id'],
FILTER_VALIDATE_INT
);

// Validar campos
if (empty($nombre)) {
$errores[] =
"El nombre es obligatorio";
}

if ($precio === false || 
$precio < 0) {
$errores[] = "Precio no válido";
}

// Procesar imagen si existe
$ruta_imagen = "";
if (isset($_FILES['imagen']) &&
$_FILES['imagen']['error']
```

```
==== UPLOAD_ERR_OK) {  
    $resultado = procesarImagen(  
        $_FILES['imagen'],  
        'uploads/productos/'  
    );  
    if ($resultado['exito']) {  
        $ruta_imagen =  
            $resultado['ruta'];  
    } else {  
        $errores[] =  
            $resultado['error'];  
    }  
}
```

## Continuación del guardado

```
// Si no hay errores, guardar  
if (empty($errores)) {  
    $conexion->begin_transaction();  
  
    try {  
        $sql = "INSERT INTO productos (nombre, descripcion,  
        ...);  
        $conexion->query($sql);  
    } catch (Exception $e) {  
        // Manejar excepción  
    }  
}
```

```
precio, stock,  
categoria_id, imagen)  
VALUES (?, ?, ?, ?, ?, ?);
```

```
$stmt =  
$conexion->prepare($sql);  
$stmt->bind_param(  
"ssdis",  
$nombre, $descripcion,  
$precio, $stock,  
$categoria_id,  
$ruta_imagen  
);
```

```
if ($stmt->execute()) {  
$producto_id =  
$stmt->insert_id;
```

```
// Confirmar transacción  
$conexion->commit();
```

```
header(
```

```
"Location: listar.php?  
exito=creado"  
);  
exit();  
} else {  
throw new Exception(  
"Error al crear: " .  
$stmt->error  
);  
}  
}  
} catch (Exception $e) {  
$conexion->rollback();  
$errores[] =  
$e->getMessage();  
}  
}  
  
// Si hay errores,  
// guardar en sesión  
if (!empty($errores)) {  
$_SESSION['errores'] =  
$errores;
```

```
header("Location: crear.php");
exit();
}
}
?>
```

# Exportación e importación de datos

Para facilitar la gestión de datos, implementaremos funcionalidades para **exportar productos a CSV e importar desde archivos CSV**.

## Exportar a CSV

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

verificarSesion();
$conexion = getConexionMysqli();

// Obtener datos a exportar
$sql = "SELECT p.id, p.nombre, p.descripcion, p.precio,
```

```
p.stock, c.nombre as categoria  FROM productos p  
JOIN categorias c  
ON p.categoria_id = c.id  
ORDER BY p.nombre";
```

```
$resultado = $conexion->query($sql);
```

```
// Configurar cabeceras
```

```
header('Content-Type: text/csv; charset=utf-8');  
header('Content-Disposition:  
attachment;  
filename=productos_'.  
date('Y-m-d') . '.csv');
```

```
// Crear recurso de salida
```

```
$output = fopen('php://output', 'w');
```

```
// Escribir BOM para UTF-8
```

```
fprintf($output,  
chr(0xEF).chr(0xBB).chr(0xBF));
```

```
// Escribir cabeceras
```

```
fputcsv($output,  
['ID', 'Nombre', 'Descripción', 'Precio', 'Stock', 'Categoría']);  
  
// Escribir datos  
while ($fila =  
$resultado->fetch_assoc()) {  
fputcsv($output, [  
$fila['id'],  
$fila['nombre'],  
$fila['descripcion'],  
$fila['precio'],  
$fila['stock'],  
$fila['categoria']]  
);  
}  
  
fclose($output);  
exit();
```

?>

## Importar desde CSV

```
<?php
```

```
require_once 'sesion.php';
require_once 'conexion.php';

verificarSesion();

if ($_SERVER['REQUEST_METHOD'] === 'POST' &&
isset($_FILES['archivo_csv'])) {

$archivo =
$_FILES['archivo_csv'];
$tipo_archivo =
mime_content_type(
$archivo['tmp_name']
);

if ($tipo_archivo !== 'text/csv'
&& $tipo_archivo !==
'text/plain') {
echo "El archivo debe ser CSV"; exit();
}

// Abrir archivo
```

```
$handle = fopen(  
$archivo['tmp_name'], 'r'  
);  
  
if ($handle !== FALSE) {  
// Leer cabeceras  
$cabeceras = fgetcsv($handle);  
  
// Inicializar contadores  
$creados = 0;  
$actualizados = 0;  
$errores = 0;  
  
$conexion =  
getConexionMysqli();  
$conexion->begin_transaction();  
  
try {  
// Leer línea por línea  
while (($datos =  
fgetcsv($handle))  
!== FALSE) {
```

```
// Procesar cada línea  
// Validar y guardar/  
// actualizar  
  
// Implementación  
// completa...  
}  
  
$conexion->commit();  
echo "Importación completada";  
  
} catch (Exception $e) {  
    $conexion->rollback();  
    echo "Error: " .  
        $e->getMessage();  
}  
  
fclose($handle);  
}  
}  
?>
```

# Mejores prácticas de seguridad

Implementar **buenas prácticas de seguridad** es fundamental para proteger tu aplicación y los datos de tus usuarios.

## Validación y sanitización

Siempre valida y sanitiza **todas las entradas de usuario** antes de procesarlas. Nunca confíes en los datos del cliente.

## Mínimo privilegio

Usa usuarios de base de datos con los **mínimos permisos necesarios** para cada operación.

## HTTPS

Configura tu sitio para usar **HTTPS** para proteger los datos en tránsito.

## Sesiones seguras

Configura adecuadamente las opciones de sesión con **httponly** y **secure**.

## Sentencias preparadas

Usa **siempre sentencias preparadas** para prevenir inyecciones SQL. Esta es la defensa más importante.

## Contraseñas seguras

Nunca almacenes contraseñas en texto plano, usa **password\_hash()** para encriptarlas.

## Protección CSRF

Implementa **tokens CSRF** en todos los formularios para prevenir ataques de falsificación.

## Manejo de errores

No muestres errores detallados en producción, **registra en logs** en su lugar.

# Optimización de rendimiento

La **optimización del rendimiento** es crucial para aplicaciones que manejan grandes volúmenes de datos o muchos usuarios simultáneos.

## Índices

Crea **índices** para campos usados frecuentemente en consultas WHERE, JOIN y ORDER BY.

```
CREATE INDEX idx_email  
ON usuarios(email);
```

```
CREATE INDEX idx_categoria  
ON productos(categoria_id);
```

## Paginación

Utiliza **LIMIT** para limitar resultados y implementa paginación en consultas grandes.

```
SELECT * FROM productos  
ORDER BY nombre  
LIMIT 10 OFFSET 20;
```

## Procesamiento de imágenes

Optimiza imágenes antes de almacenarlas: redimensiona, comprime y convierte a formatos modernos.

## Consultas eficientes

Selecciona solo las **columnas necesarias** en lugar de usar `SELECT *`.

-- Mejor  
`SELECT id, nombre, email  
FROM usuarios;`

-- Evitar  
`SELECT * FROM usuarios;`

## Caché

Implementa **caché** para consultas frecuentes y costosas usando Redis o Memcached.

## Compresión

Habilita la **compresión GZIP** para reducir el tamaño de la transferencia de datos.

## Herramientas de depuración y monitoreo

Usar las **herramientas**

**adecuadas** te ayudará a identificar y resolver problemas rápidamente en tu aplicación.

analizar estructuras de base de datos.

## phpMyAdmin

Herramienta visual para **ejecutar consultas**, inspeccionar tablas y

MySQL las ejecuta y optimizarlas.

```
EXPLAIN SELECT * FROM productos  
WHERE categoria_id = 5;
```

## Slow Query Log

Habilita el **registro de consultas lentas** en MySQL para identificar queries problemáticas.

## Error Logs

Revisa los **logs de errores** de PHP y MySQL regularmente para detectar problemas.

## Xdebug

Herramienta de **depuración** para PHP que permite hacer seguimiento paso a paso del código.

## EXPLAIN

Usa **EXPLAIN** antes de tus consultas SELECT para analizar cómo

## Monitoreo

Implementa herramientas de **monitoreo** como New Relic o Datadog para aplicaciones en producción.

# Patrones de diseño y arquitectura

Aplicar **patrones de diseño** y una arquitectura sólida hará que tu código sea más mantenable y escalable.

1

## Patrón MVC (Modelo-Vista-Controlador)

Separa la lógica de negocio, la presentación y el control de flujo en capas independientes.

**Modelo:** Maneja los datos y la lógica de negocio

**Vista:** Presenta la información al usuario

**Controlador:** Gestiona las peticiones y coordina modelo y vista

2

## Patrón Repositorio

Abstactra el acceso a datos, facilitando cambios en la capa de persistencia sin afectar el resto de la aplicación.

3

## Inyección de dependencias

Proporciona las dependencias necesarias a una clase desde el exterior en lugar de crearlas internamente.

4

## Principios SOLID

Conjunto de principios de diseño orientado a objetos que mejoran la mantenibilidad del código.

# ¿Qué sigue después de este curso?

¡Felicitaciones por completar esta guía de MySQL con PHP! Aquí tienes algunas sugerencias para **continuar tu aprendizaje** y convertirte en un desarrollador profesional.



## Frameworks PHP

Aprende **Laravel**, Symfony o CodeIgniter para desarrollo estructurado y profesional con herramientas modernas.

## Arquitectura MVC

Organiza tu código con el patrón **Modelo-Vista-Controlador** para proyectos más complejos.

## Pruebas automatizadas

**PHPUnit** para pruebas unitarias y de integración que garanticen la calidad del código.

## DevOps

**Docker**, CI/CD, despliegue automatizado y gestión de infraestructura como código.

## APIs RESTful

Crea **servicios web** para aplicaciones móviles o frontend moderno usando arquitectura REST.

## Autenticación avanzada

**OAuth**, JWT, autenticación de dos factores y sistemas de permisos granulares.

## Frontend moderno

JavaScript, **Vue.js**, React o Angular para interfaces dinámicas y experiencias de usuario ricas.

## Bases de datos NoSQL

Explora **MongoDB**, Redis o Elasticsearch para casos de uso específicos.

Recuerda que el aprendizaje es un **proceso continuo**. Practica creando proyectos personales, participa en comunidades de desarrollo y mantente actualizado con las últimas tendencias.