Acceso a Base de Datos desde PHP 8.4

Conexión, CRUD y Operaciones Avanzadas

Aprende a conectar y trabajar con bases de datos relacionales desde PHP 8.4 de forma moderna y segura utilizando PDO. Ejemplo práctico: Tienda de Frutas

Índice

Introducción a Base de Datos

¿Qué es una Base de Datos?

Una base de datos es un **conjunto organizado de datos** almacenados de forma estructurada en un servidor. Permite guardar, recuperar, actualizar y eliminar información de manera eficiente y segura.

Tipos de Base de Datos

- **Relacionales:** Organizan datos en tablas con filas y columnas (MySQL, PostgreSQL, MariaDB)
- **NoSQL:** Almacenan datos en documentos JSON, clave-valor (MongoDB, Redis)
- **Gráficas:** Representan relaciones complejas entre datos (Neo4j)

¿Por qué son importantes?

- ✓ **Persistencia:** Los datos se mantienen incluso después de cerrar la aplicación
- ✓ **Escalabilidad:** Pueden manejar millones de registros eficientemente
- ✓ **Seguridad:** Controlan el acceso y protegen los datos
- ✓ **Integridad:** Garantizan que los datos sean consistentes
- ✓ **Consultas:** Permiten recuperar información específica rápidamente

En este curso

Nos enfocamos en **bases de datos relacionales** (MySQL, MariaDB, PostgreSQL) que son las más comunes en aplicaciones web.

Instalación de Base de Datos

1 XAMPP (Recomendado para principiantes)

Paquete todo en uno que incluye Apache, MySQL, PHP y Perl. Ideal para desarrollo local rápido.

- ✓ Instalación sencilla en Windows, Mac y Linux
- ✓ Panel de control gráfico (XAMPP Control Panel)
- ✓ Incluye phpMyAdmin para gestionar BD

2 MySQL Vanilla (Instalación manual)

Instalar MySQL directamente desde mysql.com sin dependencias adicionales. Mayor control y personalización.

- ✓ Control total sobre la configuración
- ✓ Acceso a las últimas versiones
- ✓ Requiere conocimientos de línea de comandos

3 Docker (Profesional)

Contenedores que encapsulan la BD con todas sus dependencias. Ideal para desarrollo profesional y producción.

- ✓ Entorno consistente en cualquier máquina
- ✓ Fácil de compartir y reproducir
- ✓ Requiere aprender Docker

4 Servicios en la nube

Amazon RDS, Google Cloud SQL, Azure Database. BD gestionada sin necesidad de instalar nada localmente.

- ✓ Sin mantenimiento local requerido
- ✓ Escalabilidad automática
- ✓ Requiere cuenta y configuración en la nube

PDO - PHP Data Objects

¿Qué es PDO?

PDO es una **interfaz de abstracción** que proporciona un método consistente para acceder a diferentes bases de datos desde PHP. Funciona como una capa intermedia entre tu código y la BD, permitiendo cambiar de base de datos sin modificar el código.

Ventajas de PDO

- ✓ **Multiplataforma:** Soporta MySQL, PostgreSQL, SQLite, Oracle, etc.
- ✓ **Seguridad:** Prepared statements previenen SQL injection
- ✓ **Consistencia:** Misma sintaxis para diferentes BD
- ✓ **Moderno:** Recomendado en PHP 8.4
- ✓ **Excepciones:** Manejo de errores integrado

Comparación con otras formas

❌ **mysql_* (Deprecado)**

Eliminado en PHP 7.0. No usar en nuevos proyectos.

⚠️ **mysqli (Específico)**

Funciona bien pero solo para MySQL. Menos flexible que PDO.

✓ **PDO (Recomendado)**

Flexible, seguro, soporta múltiples BD. El estándar en PHP moderno.

Drivers PDO soportados

- PDO_MYSQL - MySQL y MariaDB
- PDO_PGSQL - PostgreSQL
- PDO_SQLITE - SQLite
- PDO_OCI - Oracle

Conexión a Base de Datos

¿Cómo conectarse?

Para conectarse a una base de datos desde PHP 8.4, usamos **PDO (PHP Data Objects)**. PDO proporciona una interfaz consistente para trabajar con diferentes tipos de bases de datos.

Parámetros necesarios

- **DSN:** Data Source Name (tipo de BD, host, puerto, nombre)
- **Usuario:** Credenciales de acceso a la BD
- **Contraseña:** Credenciales de acceso a la BD

Manejo de errores

Siempre envuelve la conexión en un bloque **try-catch** para capturar excepciones de PDO.

Conexión a MySQL/MariaDB

```
try { $pdo = new PDO( 'mysql:host=localhost;port=3306;dbname=tienda_frutas;charset=utf8mb4', 'usuario', 'contraseña' ); // Configurar modo de error $pdo->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION ); echo 'Conectado exitosamente'; } catch (PDOException $e) { echo 'Error de conexión: ' . $e->getMessage(); die(); }
```

Conexión a PostgreSQL

```
$pdo = new PDO( 'pgsql:host=localhost;port=5432;dbname=tienda_frutas', 'usuario', 'contraseña' );
```

Crear Tablas - Tienda de Frutas

Tablas de la Tienda

Para la tienda de frutas necesitamos crear varias tablas que almacenen información sobre categorías, productos y usuarios.

Estructura

- **categorias:** Tipos de frutas (Cítricos, Berries, etc.)
- **productos:** Frutas individuales con precio y stock
- **usuarios:** Clientes de la tienda

Tipos de Datos

- **INT:** Números enteros
- **VARCHAR:** Texto de longitud variable
- **DECIMAL:** Números decimales (precios)
- **DATETIME:** Fecha y hora

Tabla: categorias

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

Tabla: productos

```
CREATE TABLE productos ( id INT PRIMARY KEY AUTO_INCREMENT, nombre VARCHAR(150) NOT NULL, categoria_id INT NOT NULL, precio DECIMAL(10, 2) NOT NULL, stock INT DEFAULT 0, FOREIGN KEY (categoria_id) REFERENCES categorias(id) );
```

Tabla: usuarios

```
CREATE TABLE usuarios ( id INT PRIMARY KEY AUTO_INCREMENT, nombre VARCHAR(100) NOT NULL, email VARCHAR(100) NOT NULL, telefono VARCHAR(20), fecha_registro DATETIME DEFAULT CURRENT_TIMESTAMP, UNIQUE (email) );
```

Relaciones en Base de Datos

1 : N

One-to-Many

Una categoría puede tener **muchos productos**. Un producto pertenece a una sola categoría.

Ejemplo Práctico

La categoría "Cítricos" contiene múltiples productos: Naranjas, Limones, Pomelos. Cada producto está en una sola categoría.

```
categorias (1) ↓ ├─ Naranjas ├─ Limones └─ Pomelos (N)
```

N : 1

Many-to-One

Muchos productos pertenecen a una categoría. Es la misma relación vista desde el otro lado.

Ejemplo Práctico

Múltiples productos (Naranjas, Limones, Pomelos) pertenecen a la categoría "Cítricos". Perspectiva inversa de 1:N.

```
Naranjas ┐ Limones ├─ Cítricos Pomelos ┘ Manzanas ┐ Peras └─ Frutas Árbol
```

1 : 1

One-to-One

Un registro se relaciona con **exactamente uno** en otra tabla, y viceversa.

Ejemplo Práctico

Un usuario tiene un carrito de compras, y un carrito pertenece a un único usuario. Relación exclusiva.

```
Usuario 1 ↔ Carrito 1 Usuario 2 ↔ Carrito 2 Usuario 3 ↔ Carrito 3
```

One-to-Many (1:N)

Relación 1:N

Una categoría tiene **muchos productos**. Se implementa añadiendo una **clave foránea (FK)** en la tabla "muchos" que referencia a la tabla "uno".

Crear la relación

```
ALTER TABLE productos ADD FOREIGN KEY (categoria_id) REFERENCES
categorias(id) ON DELETE CASCADE;
```

Insertar datos

```
INSERT INTO categorias (nombre) VALUES ('Cítricos'); INSERT INTO productos
(nombre, categoria_id, precio, stock) VALUES ('Naranjas', 1, 2.50, 100);
```

Obtener todos los productos de una categoría

```
$stmt = $pdo->prepare( 'SELECT p.* FROM productos p WHERE p.categoria_id = ?' );
$stmt->execute([1]); $productos = $stmt->fetchAll( PDO::FETCH_ASSOC ); foreach
($productos as $producto) { echo $producto['nombre']; }
```

Con JOIN (más eficiente)

```
$stmt = $pdo->prepare( 'SELECT c.nombre as categoria, p.nombre, p.precio, p.stock
FROM productos p INNER JOIN categorias c ON p.categoria_id = c.id WHERE c.nombre =
?' ); $stmt->execute(['Cítricos']); $resultados = $stmt->fetchAll( PDO::FETCH_ASSOC
);
```

Contar productos por categoría

```
$stmt = $pdo->prepare( 'SELECT c.nombre, COUNT(p.id) as total FROM categorias c LEFT
JOIN productos p ON c.id = p.categoria_id GROUP BY c.id' ); $stmt->execute();
$conteos = $stmt->fetchAll( PDO::FETCH_ASSOC );
```

Many-to-One (N:1)

Relación N:1

Muchos productos pertenecen a **una categoría**. Es la misma relación One-to-Many vista desde el lado "muchos".

Ejemplo práctico

Cuando queremos obtener los detalles de un producto **incluyendo su categoría**, estamos navegando la relación Many-to-One.

Estructura de datos

```
Producto: - id: 5 - nombre: Naranjas - precio: 2.50 - categoria_id: 1 ←
Referencia Categoría: - id: 1 - nombre: Cítricos
```

Obtener un producto con su categoría

```php
$stmt = $pdo->prepare( 'SELECT p.id, p.nombre, p.precio, c.nombre as categoria FROM productos p INNER JOIN categorias c ON p.categoria_id = c.id WHERE p.id = ?' ); $stmt->execute([5]); $producto = $stmt->fetch( PDO::FETCH_ASSOC ); echo $producto['nombre']; echo $producto['categoria'];
```

Listar todos los productos con categoría

```php
$stmt = $pdo->prepare( 'SELECT p.id, p.nombre, p.precio, c.nombre as categoria FROM productos p INNER JOIN categorias c ON p.categoria_id = c.id ORDER BY c.nombre, p.nombre' ); $stmt->execute(); $productos = $stmt->fetchAll( PDO::FETCH_ASSOC ); foreach ($productos as $p) { echo $p['categoria'] . ': ' . $p['nombre']; }
```

Filtrar productos por categoría

```php
$categoria = 'Cítricos'; $stmt = $pdo->prepare( 'SELECT p.* FROM productos p INNER JOIN categorias c ON p.categoria_id = c.id WHERE c.nombre = ? AND p.precio < ?' ); $stmt->execute([$categoria, 3.00]); $baratos = $stmt->fetchAll( PDO::FETCH_ASSOC );
```

One-to-One (1:1)

Relación 1:1

Un registro en una tabla se relaciona con **exactamente un registro** en otra tabla. Cada usuario tiene su propio carrito.

Crear la relación

```
CREATE TABLE carritos ( id INT PRIMARY KEY AUTO_INCREMENT, usuario_id INT NOT NULL UNIQUE, fecha_creacion DATETIME, total DECIMAL(10, 2), FOREIGN KEY (usuario_id) REFERENCES usuarios(id) );
```

Nota importante

La clave **UNIQUE** en usuario_id garantiza que cada usuario tenga **solo un carrito**.

Obtener el carrito de un usuario

```
$stmt = $pdo->prepare( 'SELECT c.* FROM carritos c WHERE c.usuario_id = ?' ); $stmt->execute([1]); $carrito = $stmt->fetch( PDO::FETCH_ASSOC ); echo 'Total: ' . $carrito['total'];
```

Usuario con su carrito (JOIN)

```
$stmt = $pdo->prepare( 'SELECT u.nombre, u.email, c.id as carrito_id, c.total, c.fecha_creacion FROM usuarios u LEFT JOIN carritos c ON u.id = c.usuario_id WHERE u.id = ?' ); $stmt->execute([1]); $resultado = $stmt->fetch( PDO::FETCH_ASSOC );
```

Crear carrito para nuevo usuario

```
$usuario_id = 5; $stmt = $pdo->prepare( 'INSERT INTO carritos (usuario_id, fecha_creacion, total) VALUES (?, NOW(), 0)' ); $stmt->execute([$usuario_id]); $carrito_id = $pdo->lastInsertId();
```

Actualizar total del carrito

```
$nuevo_total = 125.50; $usuario_id = 1; $stmt = $pdo->prepare( 'UPDATE carritos SET total = ? WHERE usuario_id = ?' ); $stmt->execute([ $nuevo_total, $usuario_id ]);
```

JOINs - Consultas Relacionales

¿Qué es un JOIN?

Un JOIN combina datos de múltiples tablas relacionadas en una sola consulta. Es la forma más eficiente de obtener datos relacionados.

INNER JOIN

Retorna solo los registros que tienen coincidencia en **ambas tablas**. Ejemplo: Productos con categoría.

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

LEFT JOIN

Retorna **todos los registros de la tabla izquierda** más los coincidentes de la derecha. Ejemplo: Categorías con o sin productos.

```
SELECT c.*, COUNT(p.id) FROM categorias c LEFT JOIN productos p ON c.id = p.categoria_id GROUP BY c.id;
```

RIGHT JOIN

Retorna **todos los registros de la tabla derecha** más los coincidentes de la izquierda. Menos usado que LEFT JOIN.

INNER JOIN desde PHP

```
$stmt = $pdo->prepare( 'SELECT p.id, p.nombre, p.precio, c.nombre as cat FROM productos p INNER JOIN categorias c ON p.categoria_id = c.id WHERE p.precio > ? ORDER BY c.nombre' ); $stmt->execute([2.00]); $productos = $stmt->fetchAll( PDO::FETCH_ASSOC );
```

LEFT JOIN - Contar por categoría

```
$stmt = $pdo->prepare( 'SELECT c.nombre, COUNT(p.id) as total FROM categorias c LEFT JOIN productos p ON c.id = p.categoria_id GROUP BY c.id ORDER BY total DESC' ); $stmt->execute(); $stats = $stmt->fetchAll( PDO::FETCH_ASSOC ); foreach ($stats as $s) { echo $s['nombre'] . ': ' . $s['total']; }
```

JOIN múltiple (3 tablas)

```
$stmt = $pdo->prepare( 'SELECT u.nombre, p.nombre, c.total, pr.nombre as producto FROM usuarios u LEFT JOIN carritos c ON u.id = c.usuario_id LEFT JOIN carrito_items ci ON c.id = ci.carrito_id LEFT JOIN productos pr ON ci.producto_id = pr.id WHERE u.id = ?' ); $stmt->execute([1]); $compra = $stmt->fetchAll( PDO::FETCH_ASSOC );
```

SELECT I - Consultas Básicas

¿Qué es SELECT?

SELECT es la operación que permite **recuperar datos** de una tabla. Es la consulta más común en aplicaciones web.

Métodos PDO

- **fetch():** Obtiene un registro
- **fetchAll():** Obtiene todos los registros
- **fetchColumn():** Obtiene una columna

Modos de obtención

```
PDO::FETCH_ASSOC // Array asociativo PDO::FETCH_OBJ // Objeto PDO::FETCH_NUM
// Array numérico PDO::FETCH_BOTH // Ambos
```

Obtener todos los productos

```
$stmt = $pdo->query( 'SELECT * FROM productos' ); $productos = $stmt->fetchAll(
PDO::FETCH_ASSOC ); foreach ($productos as $p) { echo $p['nombre'] . ' - ' .
$p['precio']; }
```

Obtener un producto específico

```
$stmt = $pdo->prepare( 'SELECT * FROM productos WHERE id = ?' ); $stmt-
>execute([5]); $producto = $stmt->fetch( PDO::FETCH_ASSOC ); if ($producto) { echo
$producto['nombre']; } else { echo 'Producto no encontrado'; }
```

Obtener como objeto

```
$stmt = $pdo->prepare( 'SELECT * FROM productos WHERE id = ?' ); $stmt-
>execute([3]); $producto = $stmt->fetch( PDO::FETCH_OBJ ); echo $producto->nombre;
echo $producto->precio;
```

Contar registros

```
$stmt = $pdo->query( 'SELECT COUNT(*) as total FROM productos' ); $resultado =
$stmt->fetch( PDO::FETCH_ASSOC ); $total = $resultado['total']; echo 'Total de
productos: ' . $total;
```

SELECT II - Consultas Avanzadas

Cláusulas principales

WHERE: Filtra registros según condiciones
ORDER BY: Ordena los resultados
LIMIT: Limita el número de resultados

Operadores WHERE

```
= (igual) <> (distinto) > (mayor) < (menor) >= (mayor o igual) <= (menor o igual)
LIKE (búsqueda) IN (en lista) BETWEEN (rango)
```

Ejemplo: Productos

Obtener frutas baratas, ordenadas por precio, limitadas a 5 resultados.

WHERE - Filtrar por precio

```php
$stmt = $pdo->prepare( 'SELECT * FROM productos WHERE precio < ? ORDER BY precio ASC' );
$stmt->execute([3.00]); $baratos = $stmt->fetchAll( PDO::FETCH_ASSOC );
```

WHERE con múltiples condiciones

```php
$stmt = $pdo->prepare( 'SELECT * FROM productos WHERE stock > ? AND precio BETWEEN ? AND ?
ORDER BY nombre' ); $stmt->execute([10, 2.00, 5.00]); $disponibles = $stmt->fetchAll(
PDO::FETCH_ASSOC );
```

LIKE - Búsqueda por nombre

```php
$busqueda = '%naranja%'; $stmt = $pdo->prepare( 'SELECT * FROM productos WHERE nombre LIKE ?
ORDER BY nombre' ); $stmt->execute([$busqueda]); $resultados = $stmt->fetchAll(
PDO::FETCH_ASSOC );
```

IN - Múltiples valores

```php
$ids = [1, 3, 5]; $placeholders = implode(',', array_fill(0, count($ids), '?')); $stmt =
$pdo->prepare( 'SELECT * FROM productos WHERE id IN (' . $placeholders . ')' ); $stmt-
>execute($ids);
```

LIMIT - Paginación

```php
$pagina = 2; $por_pagina = 10; $offset = ($pagina - 1) * $por_pagina; $stmt = $pdo->prepare(
'SELECT * FROM productos ORDER BY nombre LIMIT ? OFFSET ?' ); $stmt->execute([$por_pagina,
$offset]);
```

SELECT III - Prepared Statements

¿Qué es un Prepared Statement?

Es una forma segura de ejecutar consultas SQL con parámetros. Los datos se envían **separados del SQL**, previniendo SQL Injection.

¿Por qué es importante?

SQL Injection: Ataque donde se inyecta código SQL malicioso a través de parámetros.
Los prepared statements **escapan automáticamente** los datos, haciendo imposible inyectar SQL.

Dos formas de usar parámetros

Posicionales (?): Parámetros sin nombre
Nombrados (:nombre): Parámetros con nombre

❌ INSEGURO - NO HACER ESTO

```
$nombre = $_GET['nombre']; $sql = 'SELECT * FROM productos WHERE nombre = ' . $nombre; $stmt = $pdo->query($sql); // Vulnerable a SQL Injection! // Si $nombre = "'; DROP TABLE productos; --"
```

✓ SEGURO - Parámetros posicionales

```
$nombre = $_GET['nombre']; $stmt = $pdo->prepare( 'SELECT * FROM productos WHERE nombre = ?' ); $stmt->execute([$nombre]); $resultado = $stmt->fetchAll( PDO::FETCH_ASSOC );
```

✓ SEGURO - Parámetros nombrados

```
$nombre = $_GET['nombre']; $precio_min = $_GET['precio']; $stmt = $pdo->prepare( 'SELECT * FROM productos WHERE nombre LIKE :nombre AND precio >= :precio' ); $stmt->execute([ ':nombre' => '%' . $nombre . '%', ':precio' => $precio_min ]);
```

Múltiples parámetros

```
$categoria_id = 1; $precio_min = 2.00; $precio_max = 5.00; $stmt = $pdo->prepare( 'SELECT * FROM productos WHERE categoria_id = ? AND precio BETWEEN ? AND ? ORDER BY precio' ); $stmt->execute([ $categoria_id, $precio_min, $precio_max ]); $productos = $stmt->fetchAll( PDO::FETCH_ASSOC );
```

INSERT I - Insertar Registros

¿Qué es INSERT?

INSERT es la operación que permite **añadir nuevos registros** a una tabla. Es fundamental para crear datos en la BD.

Métodos útiles

lastInsertId(): Obtiene el ID del último registro insertado
rowCount(): Obtiene el número de filas afectadas

Validación importante

Siempre valida los datos **antes de insertar**. Usa prepared statements para evitar SQL Injection.

Ejemplo: Tienda de Frutas

Insertar un nuevo producto: Manzanas, precio 1.50, stock 50

INSERT básico

```
$stmt = $pdo->prepare( 'INSERT INTO productos (nombre, categoria_id, precio, stock) VALUES (?, ?, ?, ?)' ); $stmt->execute([ 'Manzanas', 2, 1.50, 50 ]); echo 'Producto insertado';
```

Obtener el ID del nuevo registro

```
$stmt = $pdo->prepare( 'INSERT INTO productos (nombre, categoria_id, precio, stock) VALUES (?, ?, ?, ?)' ); $stmt->execute([ 'Peras', 2, 2.00, 30 ]); $nuevo_id = $pdo->lastInsertId(); echo 'ID del nuevo producto: ' . $nuevo_id;
```

Verificar filas afectadas

```
$stmt = $pdo->prepare( 'INSERT INTO productos (nombre, categoria_id, precio, stock) VALUES (?, ?, ?, ?)' ); $resultado = $stmt->execute([ 'Plátanos', 3, 0.80, 100 ]); $filas = $stmt->rowCount(); if ($filas > 0) { echo 'Insertado correctamente'; }
```

INSERT con validación

```
$nombre = $_POST['nombre'] ?? ''; $precio = $_POST['precio'] ?? 0; if (empty($nombre) || $precio <= 0) { echo 'Datos inválidos'; exit; } $stmt = $pdo->prepare( 'INSERT INTO productos (nombre, categoria_id, precio, stock) VALUES (?, ?, ?, ?)' ); $stmt->execute([ trim($nombre), 1, $precio, 0 ]);
```

INSERT II - Insertar Múltiples

¿Por qué insertar múltiples?

A menudo necesitamos **insertar varios registros** a la vez. Usar loops es más eficiente que hacer una consulta por cada registro.

Formas de hacerlo

Opción 1: Loop con INSERT individual (más lento)
Opción 2: INSERT múltiple en una sola consulta (más rápido)

Ejemplo: Importar productos

Cargar 100 frutas nuevas desde un array de datos.

Opción 1: Loop con INSERT individual

```
$productos = [ ['Naranjas', 1, 2.50, 100], ['Limones', 1, 2.00, 80], ['Manzanas', 2, 1.50, 120] ]; $stmt = $pdo->prepare( 'INSERT INTO productos (nombre, categoria_id, precio, stock) VALUES (?, ?, ?, ?)' ); foreach ($productos as $p) { $stmt->execute($p); } echo 'Insertados: ' . count($productos);
```

Opción 2: INSERT múltiple (más rápido)

```
$productos = [ ['Naranjas', 1, 2.50, 100], ['Limones', 1, 2.00, 80], ['Manzanas', 2, 1.50, 120] ]; $placeholders = []; $valores = []; foreach ($productos as $p) { $placeholders[] = '(?, ?, ?, ?)'; $valores = array_merge($valores, $p); } $sql = 'INSERT INTO productos (nombre, categoria_id, precio, stock) VALUES ' . implode(',', $placeholders); $stmt = $pdo->prepare($sql); $stmt->execute($valores);
```

Con validación y transacciones

```
$productos = $_POST['productos'] ?? []; try { $pdo->beginTransaction(); $stmt = $pdo->prepare( 'INSERT INTO productos (nombre, categoria_id, precio, stock) VALUES (?, ?, ?, ?)' ); $insertados = 0; foreach ($productos as $p) { if ($p['precio'] > 0) { $stmt->execute([ $p['nombre'], $p['categoria_id'], $p['precio'], $p['stock'] ]); $insertados++; } } $pdo->commit(); echo 'Insertados: ' . $insertados; } catch (Exception $e) { $pdo->rollBack(); echo 'Error: ' . $e->getMessage(); }
```

UPDATE - Actualizar Datos

¿Qué es UPDATE?

UPDATE permite **modificar registros existentes** en una tabla. Siempre debe incluir una cláusula WHERE para especificar qué registros actualizar.

⚠️ Advertencia importante

NUNCA olvides el WHERE. Sin WHERE, se actualizarán **todos los registros** de la tabla.

Casos de uso

Cambiar precio de un producto, actualizar stock, modificar datos de usuario, etc.

UPDATE básico

```
$stmt = $pdo->prepare( 'UPDATE productos SET precio = ? WHERE id = ?' ); $stmt->execute([2.99, 5]); $filas = $stmt->rowCount(); echo 'Actualizados: ' . $filas;
```

UPDATE múltiples columnas

```
$stmt = $pdo->prepare( 'UPDATE productos SET precio = ?, stock = ? WHERE id = ?' ); $stmt->execute([ 3.50, // nuevo precio 75, // nuevo stock 10 // id del producto ]);
```

UPDATE con condiciones múltiples

```
$stmt = $pdo->prepare( 'UPDATE productos SET stock = stock - ? WHERE id = ? AND stock >= ?' ); $cantidad = 10; $producto_id = 5; $stmt->execute([ $cantidad, $producto_id, $cantidad // Verificar stock ]); if ($stmt->rowCount() == 0) { echo 'Stock insuficiente'; }
```

UPDATE con cálculos

```
$stmt = $pdo->prepare( 'UPDATE productos SET precio = precio * ? WHERE categoria_id = ?' ); $descuento = 0.90; // 10% descuento $categoria = 1; $stmt->execute([$descuento, $categoria]); echo 'Productos rebajados: ' . $stmt->rowCount();
```

UPDATE con validación

```
$nuevo_precio = $_POST['precio'] ?? 0; $producto_id = $_POST['id'] ?? 0; if ($nuevo_precio <= 0 || $producto_id <= 0) { echo 'Datos inválidos'; exit; } $stmt = $pdo->prepare( 'UPDATE productos SET precio = ? WHERE id = ?' ); $stmt->execute([$nuevo_precio, $producto_id]);
```

DELETE - Eliminar Datos

¿Qué es DELETE?

DELETE permite **eliminar registros** de una tabla. Es la operación más peligrosa, por eso siempre requiere WHERE.

⚠️ ADVERTENCIA CRÍTICA

NUNCA ejecutes DELETE sin WHERE. Borraría **todos los registros** de la tabla. **No hay vuelta atrás**.

Alternativa recomendada

En lugar de DELETE, considera usar un campo **"eliminado" (soft delete)** para marcar registros como inactivos.

Casos de uso

Eliminar un producto específico, limpiar datos antiguos, etc.

DELETE básico

```
$stmt = $pdo->prepare( 'DELETE FROM productos WHERE id = ?' ); $stmt->execute([5]);
$filas = $stmt->rowCount(); echo 'Eliminados: ' . $filas;
```

DELETE con condiciones

```
$stmt = $pdo->prepare( 'DELETE FROM productos WHERE stock = 0 AND precio < ?' );
$stmt->execute([1.00]); echo 'Productos sin stock ' . 'eliminados: ' . $stmt-
>rowCount();
```

DELETE con validación

```
$producto_id = $_POST['id'] ?? 0; if ($producto_id <= 0) { echo 'ID inválido'; exit;
} // Verificar que existe $check = $pdo->prepare( 'SELECT id FROM productos WHERE id
= ?' ); $check->execute([$producto_id]); if (!$check->fetch()) { echo 'Producto no
existe'; exit; } // Ahora sí, eliminar $stmt = $pdo->prepare( 'DELETE FROM productos
```

Soft Delete (recomendado)

```
// En lugar de DELETE, marcar como inactivo $stmt = $pdo->prepare( 'UPDATE productos
SET eliminado = 1, fecha_eliminacion = NOW() WHERE id = ?' ); $stmt->execute([5]);
// Luego, en SELECT, filtrar: $stmt = $pdo->prepare( 'SELECT * FROM productos WHERE
```

DELETE en cascada (con relaciones)

```
// Eliminar categoría y sus productos try { $pdo->beginTransaction(); // Primero
eliminar productos $stmt = $pdo->prepare( 'DELETE FROM productos WHERE categoria_id
= ?' ); $stmt->execute([1]); // Luego eliminar categoría $stmt = $pdo->prepare(
'DELETE FROM categorias WHERE id = ?' ); $stmt->execute([1]); $pdo->commit(); }
```

Transacciones y Seguridad

¿Qué es una Transacción?

Una transacción es un **conjunto de operaciones** que se ejecutan como una unidad. O se completan todas, o ninguna.

Estados de una transacción

BEGIN: Inicia la transacción
COMMIT: Confirma los cambios
ROLLBACK: Deshace los cambios

Caso de uso: Transferencia

Restar dinero de una cuenta Y sumar a otra. Si falla una, ambas se revierten.

Seguridad: SQL Injection

Siempre usa prepared statements. Nunca concatenes variables en SQL.

Transacción básica

```
try { $pdo->beginTransaction(); // Operaciones $stmt = $pdo->prepare( 'INSERT INTO ...' ); $stmt->execute([...]); $pdo->commit(); echo 'Éxito'; } catch (Exception $e) { $pdo->rollBack(); echo 'Error: ' . $e->getMessage(); }
```

Transacción: Compra en tienda

```
try { $pdo->beginTransaction(); // 1. Crear pedido $stmt = $pdo->prepare( 'INSERT INTO pedidos (usuario_id, total) VALUES (?, ?)' ); $stmt->execute([1, 50.00]); $pedido_id = $pdo->lastInsertId(); // 2. Reducir stock $stmt = $pdo->prepare( 'UPDATE productos SET stock = stock - ? WHERE id = ?' ); $stmt->execute([5, 3]); $pdo->commit(); echo 'Compra exitosa'; } catch (Exception $e) { $pdo->rollBack(); echo 'Compra cancelada'; }
```

Savepoints (puntos de guardado)

```
try { $pdo->beginTransaction(); // Operación 1 $pdo->exec( 'INSERT INTO logs ...' ); // Punto de guardado $pdo->exec( 'SAVEPOINT sp1' ); // Operación 2 (puede fallar) $stmt = $pdo->prepare( 'UPDATE usuarios SET ...' ); $stmt->execute([...]); $pdo->commit(); } catch (Exception $e) { // Volver al savepoint $pdo->exec( 'ROLLBACK TO sp1' ); }
```

Validación antes de transacción

```
// Validar datos ANTES de transacción $usuario_id = $_POST['usuario'] ?? 0; $cantidad = $_POST['cantidad'] ?? 0; $producto_id = $_POST['producto'] ?? 0; if ($usuario_id <= 0 || $cantidad <= 0 || $producto_id <= 0) { echo 'Datos inválidos'; exit; } try { $pdo->beginTransaction(); // Operaciones seguras $pdo->commit(); } catch (Exception $e) { $pdo->rollBack(); }
```

Conectar a Diferentes BD

Ventaja de PDO

La principal ventaja de PDO es que **el código PHP es idéntico** para diferentes bases de datos. Solo cambia el **DSN (Data Source Name)**.

¿Qué es el DSN?

El DSN es una cadena que especifica:

- Tipo de BD (mysql, pgsql, sqlite)
- Host y puerto
- Nombre de la BD
- Opciones adicionales

Ejemplo

Cambiar de MySQL a PostgreSQL requiere solo cambiar el DSN, el resto del código permanece igual.

MySQL / MariaDB

```
$dsn = 'mysql:host=localhost;port=3306;dbname=tienda_frutas;charset=utf8mb4';
$usuario = 'root'; $contraseña = 'password'; $pdo = new PDO( $dsn, $usuario,
$contraseña );
```

PostgreSQL

```
$dsn = 'pgsql:host=localhost;port=5432;dbname=tienda_frutas'; $usuario = 'postgres';
$contraseña = 'password'; $pdo = new PDO( $dsn, $usuario, $contraseña ); // El resto
del código es IDÉNTICO
```

SQLite (archivo local)

```
$dsn = 'sqlite:/home/usuario/bd/tienda.db'; $pdo = new PDO($dsn); // SQLite no
requiere usuario/contraseña // Perfecto para desarrollo local
```

Configuración dinámica

```
$config = [ 'driver' => $_ENV['DB_DRIVER'], 'host' => $_ENV['DB_HOST'], 'port' =>
$_ENV['DB_PORT'], 'dbname' => $_ENV['DB_NAME'], 'user' => $_ENV['DB_USER'],
'password' => $_ENV['DB_PASSWORD'] ]; $dsn = $config['driver'] . ':host=' .
$config['host'] . ';port=' . $config['port'] . ';dbname=' . $config['dbname']; $pdo
```

Función reutilizable

```
function conectarBD( $driver, $host, $port, $dbname, $user, $pass ) { $dsn = $driver
. ':host=' . $host . ';port=' . $port . ';dbname=' . $dbname; return new PDO( $dsn,
$user, $pass ); } // Uso: $pdo = conectarBD( 'pgsql', 'localhost', '5432',
'tienda_frutas', 'postgres', 'password' );
```

Manejo de Errores

¿Por qué manejar errores?

Los errores de BD son **inevitables**: conexión fallida, tabla no existe, datos duplicados, etc. Debes **anticiparlos y manejarlos**.

Modos de error en PDO

PDO::ERRMODE_SILENT: Sin excepciones (no recomendado)
PDO::ERRMODE_WARNING: Warnings de PHP
PDO::ERRMODE_EXCEPTION: Lanza excepciones (recomendado)

Métodos útiles

errorCode(): Código de error SQL
errorInfo(): Información detallada del error
getMessage(): Mensaje del error

Configurar modo de error

```
$pdo = new PDO( $dsn, $usuario, $contraseña ); // Configurar para lanzar excepciones $pdo->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );
```

Try-Catch básico

```
try { $stmt = $pdo->prepare( 'SELECT * FROM productos WHERE id = ?' ); $stmt->execute([5]); $resultado = $stmt->fetch(); } catch (PDOException $e) { echo 'Error: ' . $e->getMessage(); }
```

Información detallada del error

```
try { $stmt = $pdo->prepare( 'INSERT INTO usuarios ...' ); $stmt->execute([...]); } catch (PDOException $e) { $errorInfo = $e->errorInfo; echo 'SQLSTATE: ' . $errorInfo[0]; echo 'Código: ' . $errorInfo[1]; echo 'Mensaje: ' . $errorInfo[2]; }
```

Múltiples excepciones

```
try { $pdo->beginTransaction(); $stmt = $pdo->prepare( 'INSERT INTO ...' ); $stmt->execute([...]); $pdo->commit(); } catch (PDOException $e) { $pdo->rollBack(); echo 'Error BD: ' . $e->getMessage(); } catch (Exception $e) { echo 'Error general: ' . $e->getMessage(); }
```

Logging de errores

```
try { $stmt = $pdo->prepare( 'SELECT * FROM ...' ); $stmt->execute([...]); } catch (PDOException $e) { // Guardar error en log $log = '[' . date('Y-m-d H:i:s') . '] ' . $e->getMessage() . "\n"; file_put_contents( '/var/log/errores.log', $log, FILE_APPEND ); // Mostrar mensaje genérico al usuario echo 'Error en la BD. Intenta más tarde.'; }
```

Ejercicios Prácticos I

Ejercicio 1: Crear la BD de Tienda de Frutas

Crea una base de datos llamada **"tienda_frutas"** con las siguientes tablas:

• **categorias** (id, nombre, descripción)

• **productos** (id, nombre, categoria_id, precio, stock)

• **usuarios** (id, nombre, email, contraseña)

• **pedidos** (id, usuario_id, fecha, total)

> 💡 *Usa PRIMARY KEY, FOREIGN KEY y NOT NULL donde sea necesario*

Ejercicio 2: Insertar datos iniciales

Inserta al menos 3 categorías (Cítricos, Frutas Rojas, Tropicales) y 10 productos diferentes con sus precios y stock.

> 💡 *Usa INSERT múltiple para hacerlo más eficiente*

Ejercicio 3: Consultas SELECT básicas

Escribe consultas PHP para:

a) Obtener todos los productos ordenados por precio (menor a mayor)

b) Obtener productos de una categoría específica

c) Obtener productos con stock menor a 20

d) Contar cuántos productos hay en total

> 💡 *Usa prepared statements con parámetros*

Ejercicio 4: JOIN - Productos con categoría

Escribe una consulta que obtenga el nombre del producto, su precio y el nombre de su categoría. Usa INNER JOIN.

Luego, ordena los resultados por categoría y dentro de cada categoría por precio.

> 💡 *SELECT p.nombre, p.precio, c.nombre FROM productos p INNER JOIN categorias c...*

Ejercicio 5: UPDATE - Cambiar precios

Crea un script PHP que:

a) Aumente el precio de todos los productos de una categoría en un 10%

b) Reduzca el stock de un producto específico cuando se realiza una compra

c) Valide que el stock no sea negativo antes de actualizar

> 💡 *Usa transacciones para garantizar que ambas operaciones se completen*

Ejercicios Prácticos II

Ejercicio 6: DELETE - Eliminar productos

Crea un script que elimine productos sin stock (stock = 0). Pero antes, implementa un **soft delete** añadiendo una columna "eliminado" en la tabla productos.

Luego, modifica tus consultas SELECT para no mostrar productos eliminados.

> 💡 *Usa UPDATE en lugar de DELETE para marcar como eliminado*

Ejercicio 7: Simulación de compra

Crea un script que simule una compra:

1. Crear un nuevo pedido para un usuario
2. Reducir el stock del producto
3. Calcular el total del pedido
4. Usar transacciones para garantizar consistencia
5. Manejar errores (stock insuficiente, usuario no existe, etc.)

> 💡 *Usa try-catch con PDOException y beginTransaction()*

Ejercicio 8: Reportes y análisis

Crea consultas que generen reportes:

a) Productos más vendidos (requiere tabla de detalles de pedidos)

b) Ingresos totales por categoría

c) Productos con bajo stock (< 10 unidades)

d) Usuarios con más compras

> 💡 *Usa GROUP BY, SUM(), COUNT() y ORDER BY para análisis*

Desafío Extra: Sistema completo

Integra todos los conceptos aprendidos para crear un sistema de tienda online básico:

- Gestión de productos (CRUD completo)
- Gestión de usuarios y autenticación
- Carrito de compras con transacciones
- Reportes de ventas y estadísticas
- Manejo robusto de errores