Ejercicios Prácticos: Acceso a Datos

Departamento de Informática

Enero 2026

Reto 1: El Puntero de Archivo

Enunciado

Imagina que el archivo `datos.csv` tiene una cabecera y 5 filas de productos. ¿Cuántas veces en total hay que ejecutar la función `fgetcsv()` para leer todos los productos (incluyendo la cabecera)?

Pista para los apuntes: Revisa cómo funciona el bucle `while` y la línea que está justo antes.

Reto 2: Modos de Apertura

Enunciado

Un alumno se equivoca y escribe: `$gestor = fopen($csvFile, "w");`. ¿Qué le ocurrirá a su archivo CSV de origen antes de empezar la importación?

- A) No pasa nada, funciona igual.
- B) El archivo se borra (queda vacío) al abrirlo en modo escritura.
- C) PHP da un error de sintaxis.

Reto 3: El misterio del ID

Enunciado

Si borramos todos los productos con `DELETE` y luego insertamos uno nuevo, su ID será el 51. ¿Qué comando debemos usar en `$pdo->exec()` para que el primer producto insertado tenga siempre el **ID 1**?

```
// Escribe la linea de codigo necesaria:
$pdo->exec("_____________________________");
```

Reto 4: Seguridad y Marcadores

Enunciado

Completa el código para insertar un producto de forma segura usando marcadores de posición (?).

```
$nombre = "Raton"; $precio = 15.50; $stock = 10;

$stmt = $pdo->prepare("INSERT INTO productos (nombre, precio, stock)
                       VALUES (____, ____, ____)");

$stmt->execute([____, ____, ____]);
```

Reto 5: El Botón del Pánico

Enunciado

Estamos importando 1.000 filas. En la fila 500 el servidor se apaga. Si hemos usado `beginTransaction()` pero **NO** se ha llegado a ejecutar el `commit()`...

Pregunta: ¿Cuántos productos habrá en la base de datos cuando el servidor vuelva a arrancar? Explica por qué.

Reto 6: Diccionario de Errores

Enunciado

A un compañero le sale este error:
`SQLSTATE[HY000] [2002] Connection refused`

Tareas:

1. ¿Qué dos cosas debe revisar en su panel de control de XAMPP?
2. Si el puerto es el problema, ¿cómo debe quedar su variable `$host`?

Reto 7: El Mapeo de Datos (Arrays)

Escenario

El archivo CSV ha cambiado de proveedor. Ahora el orden de las columnas en el archivo de texto es: `PRECIO (50.00), STOCK (10), NOMBRE (Teclado)`

Pregunta: El código actual es:

```
$stmt->execute([ $datos[0], $datos[1], $datos[2] ]);
```

¿Cómo debes cambiar los índices `[0, 1, 2]` para que el nombre, precio y stock se guarden en la columna correcta de la base de datos?

Reto 8: Filtrado de Datos (IF)

Desafío de Código

El jefe te pide que **NO** importes los productos que están agotados (es decir, los que tienen Stock 0 en el CSV).

Tarea: ¿Dónde colocarías un `if` y qué condición escribirías?

```
while (($datos = fgetcsv($gestor...)) !== FALSE) {

    // Escribe aqui tu IF para saltar la linea si stock es 0
    if ( ___________ ) {
        continue; // Salta a la siguiente vuelta del bucle
    }

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

Reto 9: Encuentra el Error de Rendimiento

Caso

Un alumno ha escrito este código. Funciona, pero tarda 10 veces más que el tuyo. ¿Por qué?

```
while (($datos = fgetcsv($gestor...)) !== FALSE) {

    // EL ALUMNO HA PUESTO ESTO DENTRO DEL BUCLE:
    $stmt = $pdo->prepare("INSERT INTO productos (...) VALUES (...)");

    $stmt->execute([ $datos[0], ... ]);
}
```

Respuesta esperada: Al poner el `prepare` dentro, estás obligando a MySQL a compilar la consulta mil veces. Debe ir fuera.

Reto 10: Contador de Éxitos

Enunciado

El script actual solo dice 'Éxito' al final. Modifícalo para que nos diga exactamente **cuántos productos se han insertado**.

```php
$contador = 0; // 1. Inicializar variable

while (...) {
    $stmt->execute(...);

    // 2. ¿Qué linea falta aqui?
    ------------- ;
}

// 3. Mostrar resultado
echo "Se han cargado $contador productos.";
```

Reto 11 (Nivel Pro): Cambiar a UPDATE

Enunciado

Imagina que el script no es para crear productos nuevos, sino para **actualizar el precio** de los que ya existen.

Escribe cómo cambiaría la línea del `PREPARE`:

```
    // CAMBIA EL INSERT POR UN UPDATE:
    $sql = "UPDATE productos SET precio = ? WHERE nombre = ?";
```

Nota: Asumimos que buscamos por nombre para este ejercicio.

Reto 12: El Puerto Maldito

Situación

Tu compañero intenta conectar pero XAMPP le ha asignado el puerto **3307** en lugar del 3306 habitual. Su script falla.

Tarea: Escribe cómo debe quedar la línea de la variable `$host`.

```
// CÓDIGO ACTUAL (FALLA):
$host = 'localhost';

// TU SOLUCIÓN:
$host = '____________________';
```

Reto 13: El Chivato de Errores

Situación

El script de María falla silenciosamente (pantalla en blanco). Necesitamos que PHP lance una **Excepción** si MySQL da error.

Tarea: Escribe la línea de configuración del objeto PDO que falta.

```
$pdo = new PDO($dsn, $user, $pass);

// ESCRIBE LA LÍNEA QUE FALTA AQUÍ:
$pdo->setAttribute(_________________, _________________);
```

Reto 14: Evitar el Crash

Situación

Cada vez que recargamos la página, el script intenta crear la tabla ‘productos‘ y falla porque ya existe.

Tarea: Corrige la sentencia SQL para que sea **Idempotente**.

```
// CÓDIGO ACTUAL (ROMPE SI YA EXISTE):
$pdo->exec("CREATE TABLE productos (...)");

// TU SOLUCIÓN:
$pdo->exec("CREATE TABLE _________ productos (...)");
```

Reto 15: Reinicio de Fábrica

Situación

Queremos borrar todos los datos antes de importar el CSV. Si usamos 'DELETE', los IDs siguen creciendo (51, 52...). Queremos que vuelvan a empezar en 1.

Tarea: Escribe el comando SQL exacto para vaciar y resetear.

```php
// ESCRIBE EL COMANDO PHP COMPLETO:
$pdo->exec("__________________________");
```

Reto 16: Mapeo de Columnas (El Cruce)

Datos

CSV (Archivo): [0]=Precio, [1]=Nombre, [2]=Stock
BD (Tabla): (nombre, precio, stock)

Tarea: Rellena los índices del array `$datos` en el orden correcto.

```php
$stmt = $pdo->prepare("INSERT INTO productos (nombre, precio, stock) VALUES (?, ?, ?)");

while (($datos = fgetcsv($gestor)) !== FALSE) {
    // Orden esperado por BD: Nombre, Precio, Stock
    $stmt->execute([
        $datos[__],  // Nombre
        $datos[__],  // Precio
        $datos[__]   // Stock
    ]);
}
```

Reto 17: Filtrado Lógico (El IF)

Situación

El jefe dice: "No quiero importar productos con precio 0 o negativo".

Tarea: Implementa la lógica dentro del bucle.

```
while (($datos = fgetcsv($gestor)) !== FALSE) {
    $precio = $datos[1];

    // ESCRIBE TU IF AQUÍ PARA SALTAR LA LÍNEA:
    if ( ______________ ) {
        continue;
    }

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

Reto 18: Atomicidad

Situación

Queremos que la importación sea "Todo o Nada".

Tarea: Coloca los comandos `commit()` y `rollBack()` en su sitio.

```
try {
    $pdo->beginTransaction();
    // ... (insertamos 1000 productos) ...

    // 1. SI TODO VA BIEN, GUARDAMOS:
    $pdo->________();

} catch (Exception $e) {

    // 2. SI HAY ERROR, DESHACEMOS:
    $pdo->________();
    echo "Error: " . $e->getMessage();
}
```

Reto 19: Implementación del Singleton

Objetivo

Evitar el error 'Too many connections' asegurando una instancia única.

Tarea: Completa la lógica del método estático para que solo cree la conexión si no existe previamente.

```
class Database {
    private static $instance = null;

    // 1. ¿Qué visibilidad debe tener el constructor?
    ---------- function __construct() {}

    public static function conectar() {
        // 2. Patrón Singleton: Lógica de control
        if (self::$instance === ----------) {

            $opciones = [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION];
            self::$instance = new PDO(...$opciones);
        }
        // 3. Retorno
        return --------------------;
    }
}
```

Reto 20: Seguridad Frontend (XSS)

Escenario

Un atacante ha logrado guardar este nombre de producto en la BD: `<script>alert('Hacked')</script>`.

Tarea: Tu código PHP es vulnerable al mostrarlo. Arréglalo usando la función nativa correcta.

```php
// CÓDIGO VULNERABLE (Ejecuta el JS del atacante):
echo "<h1>Producto: " . $producto->nombre . "</h1>";

// TU SOLUCIÓN SEGURA:
// Pista: Convierte caracteres especiales en entidades HTML

echo "<h1>Producto: " . _________________($producto->nombre) . "</h1>";
```

Reto 21: Trampa de Binding (Teoría)

Pregunta de Entrevista

Analiza el siguiente código con `bindParam` (vinculación por referencia).

```php
$estado = "Activo";
$stmt = $pdo->prepare("INSERT INTO logs (msg) VALUES (:m)");

// OJO: bindParam vincula la VARIABLE, no el valor actual
$stmt->bindParam(':m', $estado);

$estado = "Eliminado"; // Cambiamos la variable DESPUÉS del bind
$stmt->execute();      // Ejecutamos AHORA
```

¿Qué valor exacto se guarda en la base de datos?

- A) Activo
- B) Eliminado
- C) Error de sintaxis

Reto 22: Integridad Referencial Manual

Caso Real (Biblioteca)

Al borrar un Autor, decidimos **desvincular** sus libros (poner autor_id a NULL) en lugar de borrarlos. Esto requiere una transacción atómica.

Tarea: Ordena las operaciones lógicamente.

```
try {
    $pdo->beginTransaction();
    // PASO 1: Desvincular libros (UPDATE autor_id = NULL)
    $stmt1 = $pdo->prepare("UPDATE libros SET ... WHERE autor_id = ?");
    $stmt1->execute([$id]);
    // PASO 2: Borrar al autor
    $stmt2 = $pdo->prepare("DELETE FROM autores WHERE id = ?");
    $stmt2->execute([$id]);
    // PASO 3: ¿Cómo guardamos los cambios definitivamente?
    $pdo->___________();
} catch (Exception $e) {
    // PASO 4: ¿Cómo deshacemos si falla el paso 2?
    $pdo->___________();
}
```

Reto 23: FETCH_CLASS

Objetivo

Queremos que PDO nos devuelva instancias de la clase `Producto`, no arrays asociativos, para poder usar métodos como `calcularIVA()`.

Tarea: Escribe la configuración correcta del fetchAll.

```
class Producto { public $precio; ... }

$stmt = $pdo->query("SELECT * FROM productos");

// COMPLETA PARA OBTENER OBJETOS 'Producto':
$misProductos = $stmt->fetchAll(PDO::_____________, '_____________');

// Comprobación:
echo $misProductos[0]->precio; // Funciona con flecha ->
```

Reto 24: DDL Avanzado (Infraestructura)

Requisito de Infraestructura

Debes crear una tabla `pedidos` con Clave Foránea y motor compatible con transacciones.

Tarea: Rellena los huecos SQL críticos.

```
$sql = "CREATE TABLE pedidos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    usuario_id INT,

    -- Definición de FK
    CONSTRAINT fk_user
    FOREIGN KEY (__________) REFERENCES usuarios(id)

) ENGINE=__________ CHARACTER SET utf8mb4";
// Pista: El motor debe soportar ACID (No es MyISAM)

$pdo->exec($sql);
```

Reto 25: Infraestructura y Herramientas

Objetivo del Módulo

Antes de escribir una sola línea de PHP, un desarrollador Backend debe saber preparar el terreno en la base de datos.

¿Qué vamos a practicar aquí?

- **Configuración Técnica:** Diferencia entre `utf8mb4` (Emojis) y Motores `InnoDB` (Transacciones).
- **Seguridad de Servidor:** Creación de usuarios con privilegios limitados para evitar desastres.
- **Productividad:** Uso de DBeaver para generar código SQL automáticamente (Ingeniería Inversa).
- **Diagnóstico:** Cómo identificar errores de conexión comunes (Puertos y Credenciales).

Misión 1: La Base de Datos Moderna

Concepto Teórico: Motores y Caracteres

- **Cotejamiento (Collation):** Define qué caracteres se pueden guardar. El antiguo `utf8_general_ci` solo soporta 3 bytes y **rompe los Emojis**. El estándar moderno es `utf8mb4` (4 bytes).
- **Motor (Engine):** Define cómo se guardan los datos. **InnoDB** es el único que soporta Transacciones (ACID) y Claves Foráneas. **MyISAM** es obsoleto y propenso a errores.

Tu Tarea (phpMyAdmin / DBeaver)

Crea una base de datos llamada `social_media` asegurando que:

1. Pueda guardar emojis (usando `utf8mb4_unicode_ci`).
2. Soporte transacciones bancarias (usando `InnoDB`).

Evidencia: *Captura de la pestaña 'Operaciones' mostrando ambas configuraciones.*

Misión 2: Seguridad y Permisos

Concepto Teórico: Principio de Mínimo Privilegio

Nunca uses el usuario `root` en tu código PHP. Si un hacker entra, podría borrar todas las bases de datos del servidor. Debemos crear usuarios limitados que solo puedan **LEER y ESCRIBIR** datos (DML), pero no **BORRAR TABLAS** (DDL).

Tu Tarea

Crea un usuario `app_user` con contraseña. En la matriz de privilegios:

- **MARCA:** Datos (SELECT, INSERT, UPDATE, DELETE).
- **DESMARCA:** Estructura (DROP, TRUNCATE) y Administración.

Evidencia: *Captura de pantalla de los permisos asignados.*

Misión 3: Ingeniería Inversa con DBeaver

Concepto Teórico: DDL y Portabilidad

Las interfaces gráficas son cómodas, pero en una empresa se comparte código. La **Ingeniería Inversa** consiste en obtener el código fuente (SQL) a partir de algo que ya está construido (la tabla visual).

Tu Tarea

1. Crea visualmente una tabla `usuarios` (id, email, password).
2. Usa DBeaver para generar su código SQL automáticamente.
(*Pista: Clic derecho en la tabla > Generar SQL > DDL*).

Evidencia (Pega el código generado):

```
-- Tu código SQL aquí:
CREATE TABLE usuarios ( ... );
```

Misión 4: Diagnóstico de Conexión

Concepto Teórico: Error 1045

El error `Access denied for user` (Código 1045) significa que el servidor MySQL funciona, pero las credenciales (usuario/clave) son incorrectas. En XAMPP, la configuración por defecto es usuario `root` y **contraseña vacía**.

Análisis del Caso

Un alumno tiene este código en su `config.php` y le da error 1045: `$pass = 'admin123';`

Preguntas de evidencia:

1. ¿Por qué falla si él jura que quiere usar esa contraseña?
2. ¿Qué tendría que hacer en **phpMyAdmin** primero para que esa contraseña funcionase?

Reto 26: Arquitectura de Software

Concepto Teorico: Patrones de Diseno

En el desarrollo profesional no escribimos codigo "suelto". Utilizamos estructuras probadas llamadas Patrones de Diseno. Esto permite que el codigo sea mantenible, escalable y que varios programadores trabajen en el mismo proyecto sin estorbarse.

¿Que vamos a practicar en este bloque?

- **Agnosticismo:** Aprender a conectar a diferentes bases de datos (MySQL o SQLite) sin cambiar la logica de la aplicacion.
- **Patron Singleton:** Crear un "guardian"de la conexion para evitar abrir cientos de procesos innecesarios en el servidor.
- **Patron Repository:** Separar las consultas SQL de la vista HTML para que el codigo este ordenado.
- **ORM Nativo:** Dejar de usar arrays asociativos y empezar a trabajar con objetos reales mediante `FETCH_CLASS`.

Misión 1: El Camaleón (Multi-Motor)

Concepto Teórico: Agnosticismo

Una buena aplicación no debe çasarseçon una sola marca de base de datos. Usamos **PDO (PHP Data Objects)** porque nos permite cambiar de MySQL (Servidor) a SQLite (Archivo local) cambiando solo una línea de configuración (el DSN), sin reescribir todo el código.

Tu Tarea (config.php)

Completa el `switch` para que la aplicación sepa conectarse a una base de datos SQLite (que no usa usuario ni contraseña, solo una ruta de archivo).

```php
$motor = "sqlite"; // Variable de control
switch ($motor) {
    case 'mysql':
        $dsn = "mysql:host=$host;dbname=$db;charset=utf8mb4";
        break;
    case 'sqlite':
        // SQLite usa el prefijo "sqlite:" seguido de la ruta absoluta
        $dsn = "_____________________________________";
        break;
}
```

Misión 2: El Guardián de la Conexión (Singleton)

Concepto Teórico: Eficiencia de Recursos

Abrir una conexión a la BD es un proceso lento y costoso para la CPU. Si tienes 1.000 visitas y cada una abre una conexión nueva, el servidor colapsará ('Too many connections'). El **Patrón Singleton** garantiza que solo exista **una única instancia** de la conexión reutilizable para todo el ciclo de vida.

Tu Tarea (Database.php)

Implementa la lógica de control para evitar crear conexiones duplicadas.

Evidencia (Completa los huecos):

```
public static function conectar() {
    // Si la propiedad estática $instance es NULL, significa que
    // es la primera vez. Creamos la conexión.
    if (self::$instance === __________) {
        self::$instance = new PDO(...);
    }

    // Si ya existía, devolvemos la que ya estaba creada (Reutilización)
    return ____________________;
}
```

Misión 3: Separación de Poderes (Repository)

Concepto Teórico: MVC y Repositorios

Nunca escribas consultas SQL (`SELECT...`) dentro de tus archivos HTML o Controladores (`index.php`). Eso es 'Código Espagueti'. El **Patrón Repository** crea una clase dedicada exclusivamente a hablar con la base de datos. El resto de la web solo pide datos a esta clase.

Tu Tarea (ProductoRepository.php)

Crea un método estático que busque un producto por ID y devuelva un Objeto.

Evidencia (Código):

```
public static function getById($id) {
    $pdo = Database::conectar();
    $stmt = $pdo->prepare("SELECT * FROM productos WHERE id = :id");
    $stmt->execute(['id' => $id]);

    // NO queremos un array. Queremos un OBJETO de la clase Producto.
    return $stmt->fetchObject('_____________');
}
```

Misión 4: Mapeo de Objetos (ORM Nativo)

Concepto Teórico: Objetos Inteligentes

Los arrays (`$p['precio']`) son "tontos", solo guardan datos. Los objetos (`$p->precio`) son inteligentes, pueden tener métodos (ej: `$p->calcularIVA()`). PDO tiene un modo llamado `FETCH_CLASS` que inyecta los datos de la tabla directamente en las propiedades de tu Clase.

Tu Tarea

Tienes una clase `Usuario`. Escribe la línea necesaria para obtener todos los usuarios de la BD convertidos automáticamente en instancias de esa clase.

Evidencia (Una sola línea):

```
// Completa la constante de PDO necesaria:
$usuarios = $stmt->fetchAll(PDO::_____________, 'Usuario');
```

Reto 27: Seguridad y Datos Avanzados

Concepto Teorico: Integridad y Defensa

La seguridad no es una opcion, es una obligacion. Debemos proteger la base de datos de ataques externos (Inyeccion SQL) y el navegador de los usuarios de codigos maliciosos (XSS). Ademas, aprenderemos que los datos criticos no pueden guardarse 'a medias'.

¿Que vamos a practicar en este bloque?

- **Sentencias Preparadas:** Blindar nuestras consultas contra el 'Enemigo Publico Numero 1': la Inyeccion SQL.
- **Transacciones ACID:** Implementar el concepto de "Todo o Nada"para asegurar que los datos nunca queden inconsistentes.
- **Higiene de Salida:** Usar `htmlspecialchars` para evitar que un atacante inyecte JavaScript en nuestra web (ataques XSS).
- **Gestion de Fechas:** Dominar la conversion entre el formato de la base de datos (ISO) y el formato humano.

Misión 1: El Caos Temporal (Fechas)

Concepto Teórico: ISO vs Humano

- **Base de Datos:** SIEMPRE guarda en formato ISO (`YYYY-MM-DD`). Es el único que permite ordenar cronológicamente.
- **Vista (HTML):** El usuario español espera `DD/MM/YYYY`.
- **Solución:** Nunca guardes fechas en formato español. Guárdalas en ISO y conviértelas al mostrarlas usando la clase `DateTime`.

Tu Tarea

Recibes la fecha `'2025-12-31'` de la BD. Completa el código para mostrarla como "31-12-2025".

Evidencia (Código PHP):

```php
$fechaBD = '2025-12-31'; // Dato crudo

// 1. Crear objeto DateTime
$fechaObj = new DateTime($fechaBD);

// 2. Formatear para el usuario (Día-Mes-Año)
echo $fechaObj->format('_____');
```

Misión 2: El Botón del Pánico (Transacciones)

Concepto Teórico: Atomicidad

En una operación compleja (ej: Transferencia Bancaria), si falla el segundo paso (Ingresar dinero), debemos deshacer el primero (Restar dinero).

- **commit():** 'Guardar partida' (Confirmar cambios).
- **rollBack():** 'Cargar partida anterior' (Deshacer todo).

Tu Tarea

Completa la estructura de seguridad para una operación de compra.

```
try {
    $pdo->beginTransaction(); // Congelamos el guardado
    $stmt1->execute(); // Restar Stock
    $stmt2->execute(); // Crear Pedido
    $pdo->___________(); // SI TODO VA BIEN: Confirmar
} catch (Exception $e) {
    $pdo->___________(); // SI ALGO FALLA: Deshacer
    die("Error en la compra");
}
```

Misión 3: Defensa contra XSS (Frontend)

Concepto Teórico: Cross-Site Scripting

Un hacker puede guardar código JavaScript malicioso en tu base de datos (ej: un comentario con `<script>roboCookies()</script>`). Si imprimes ese dato tal cual en tu HTML, el navegador de tus usuarios ejecutará el virus. **Solución:** Debes 'escapar' los datos al mostrarlos.

Tu Tarea

Tienes una variable `$mensaje` que viene de la BD y puede ser peligrosa. Imprímela de forma segura.

Evidencia (Función nativa):

```
// MAL (Peligroso):
echo $mensaje;

// BIEN (Seguro):
// Completa la función que convierte < en &lt;
echo ________________($mensaje);
```

Misión 4: El Arquitecto de Software

Concepto Teórico: Orden de Directorios

En un proyecto profesional, no tiramos todos los archivos en la raíz.

- **/public:** Lo único que ve el navegador (index.php, css, img).
- **/src:** El código fuente PHP (Clases, Modelos).
- **/config:** Credenciales sensibles.

Tu Tarea

Clasifica estos archivos. ¿En qué carpeta deberían ir?

Evidencia (Une con flechas o escribe al lado):

1. `Database.php` (La clase de conexión) → _____
2. `estilos.css` (Diseño web) → _____
3. `db_passwords.php` (Claves secretas) → _____

Reto 28: Diseno y Despliegue

Concepto Teorico: El Ciclo de Vida del Dato

Un buen diseno de base de datos ahorra meses de trabajo en programacion. Aprenderemos a definir reglas que la base de datos ejecutara por nosotros (como borrar datos en cascada) y como documentar y mover nuestro trabajo a un servidor real.

¿Que vamos a practicar en este bloque?

- **Integridad Referencial:** Configurar que ocurre con los 'hijos' cuando borramos a un 'padre' (Reglas `ON DELETE`).
- **Ingenieria Inversa:** Usar herramientas profesionales como DBeaver para generar diagramas visuales y codigo SQL automaticamente.
- **Gestion de Usuarios:** Aplicar el principio de 'Minimo Privilegio' para que nuestra web sea un bunker.
- **Despliegue e Idempotencia:** Aprender a exportar e importar bases de datos sin que los errores de 'tabla ya existente' detengan el proceso.

Misión 1: La Estrategia del Borrado (Cascada vs Null)

Concepto Teórico: Reglas de Integridad (FK)

Cuando relacionamos tablas, debemos decidir qué ocurre con los hijos si borramos al padre.

- **CASCADE:** Si borras al Usuario, se borran sus Pedidos. (Útil para limpiezas profundas).
- **SET NULL:** Si borras al Usuario, los Pedidos se quedan, pero el campo `usuario_id` se pone a NULL. (Útil para mantener histórico contable).
- **RESTRICT:** No te deja borrar al Usuario si tiene Pedidos. (Seguridad por defecto).

Tu Tarea (Diseño Relacional)

En tu base de datos `biblioteca_pro`, configura estas dos relaciones exactas:

1. **Autores → Libros:** Configura `ON DELETE CASCADE`. (Si borro al autor, sus libros desaparecen).
2. **Libros → Préstamos:** Configura `ON DELETE RESTRICT`. (Prohibido borrar un libro si está prestado a alguien).

Evidencia: *Captura de la 'Vista de Relaciones' en phpMyAdmin o DBeaver mostrando las reglas.*

Misión 2: El Arquitecto de Datos (Designer)

Concepto Teórico: Diagramas E-R Automáticos

No dibujes los diagramas a mano en Paint. Las herramientas profesionales (Workbench, DBeaver, phpMyAdmin Designer) generan el mapa visual real leyendo la estructura que has creado. Si las líneas de unión no aparecen, es que te faltan las Claves Foráneas (FK).

Tu Tarea

1. Asegúrate de que todas tus tablas (autores, libros, prestamos) estén relacionadas correctamente.
2. Usa la herramienta **"Diseñador"** (phpMyAdmin) o **"Diagrama ER"** (DBeaver) para generar el mapa.
3. Organiza las cajas para que las líneas no se crucen demasiado.

Evidencia: *Exporta el diagrama a imagen/PDF y pégalo aquí.*

Misión 3: Exportación Profesional (Deploy)

Concepto Teórico: Idempotencia en Despliegues

Al llevar tu BD local a la nube, no puedes subirla 'a ver qué pasa'. Necesitas un script `.sql` robusto. La opción **'Add DROP TABLE'** es vital: le dice al servidor 'Si esta tabla ya existe vieja, bórrala y crea la nueva limpia'. Sin esto, la importación fallará por errores de 'Table already exists'.

Tu Tarea (Exportar)

Genera un backup completo (`.sql`) de tu base de datos, pero debes marcar la opción **'Agregar sentencia DROP TABLE / VIEW / ...'** en el menú de exportación 'Personalizado".

Evidencia: *Abre el archivo .sql generado con el Bloc de Notas y pega las primeras 10 líneas donde se vea el comando DROP.*

Misión 4: El Simulacro de Incendio

Concepto Teórico: Restauración

Una copia de seguridad no sirve de nada si nunca has probado a restaurarla. Muchos backups fallan al restaurarse por problemas de orden (intentar crear el hijo antes que el padre) o por tamaños de archivo (Max Upload Size).

Tu Tarea (Destruir y Reconstruir)

1. **BORRA** manualmente tu base de datos completa (`DROP DATABASE biblioteca_pro`). ¡Sin miedo!
2. Crea una base de datos vacía con el mismo nombre.
3. Usa la pestaña **Importar** para cargar tu archivo `.sql` de la Misión 3.

Evidencia: *Captura de pantalla del mensaje verde de phpMyAdmin: 'Importación ejecutada exitosamente'.*