RA4: Acceso a Datos y Persinstencia

Reto 1: El Puntero de Archivo

Pregunta: Si el archivo CSV tiene 1 cabecera + 5 filas de productos, ¿cuántas veces hay que ejecutar fgetcsv?

6 veces. fgetcsv() lee UNA línea por cada llamada. El bucle while ejecuta fgetcsv() hasta que devuelve FALSE (fin de archivo). Por tanto: 1 llamada para cabecera + 5 llamadas para productos = 6 veces.

Reto 2: Modos de Apertura

Pregunta: ¿Qué ocurre si abro el CSV con fopen(csvFile, 'w')?

B - El archivo se borra (queda vacío) al abrirlo en modo escritura.

El modo 'w' (write) vacía el archivo al abrirlo. Para LEER un CSV debemos usar 'r' (read).

Reto 3: El misterio del ID

Pregunta: ¿Qué comando usar para que el primer producto insertado tenga ID 1?

```
$pdo→exec("TRUNCATE TABLE productos");
```

TRUNCATE vacía la tabla Y reinicia el contador AUTO_INCREMENT a 1. DELETE solo borra filas pero el contador sigue desde donde estaba.

Reto 4: Seguridad y Marcadores

Código completado:

```
$stmt = $pdo→prepare("INSERT INTO productos (nombre, precio, stock) V
ALUES (?, ?, ?)");
```

```
$stmt→execute([$nombre, $precio, $stock]);
```

Los marcadores ? evitan inyección SQL. En execute() pasamos los valores en el mismo orden que los marcadores.

Reto 5: El Botón del Pánico

Pregunta: Si hay un fallo en la fila 500 y NO se ejecuta commit, ¿cuántos productos habrá en la BD?

0 productos

beginTransaction() desactiva el autoguardado. Los datos se guardan en RAM temporal. Si NO hacemos commit(), al apagarse el servidor se ejecuta automáticamente un rollBack() y TODO se deshace. Por eso las transacciones garantizan atomicidad (todo o nada).

Reto 6: Diccionario de Errores

Error: SQLSTATE[HY000] [2002] Connection refused

1. En el panel de XAMPP, verificar que el módulo MySQL esté en estado "Start" (verde)
2. Verificar el puerto: si XAMPP usa 3307 en lugar de 3306, el host debe ser:

```
$host = "localhost:3307";
```

Connection refused significa que PHP intenta conectar pero nadie responde. O MySQL está apagado, o estamos llamando al puerto equivocado.

Reto 7: El Mapeo de Datos (Arrays)

Situación: El CSV cambió de formato. Ahora es: PRECIO, STOCK, NOMBRE

Código actual (INCORRECTO):

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

Código corregido:

```
$stmt→execute([$datos[2], $datos[0], $datos[1]]);
// nombre (posición 2), precio (posición 0), stock (posición 1)
```

El prepare espera (nombre, precio, stock) pero el CSV ahora tiene (precio, stock, nombre). Debemos reordenar los índices del array.

Reto 8: Filtrado de Datos (IF)

Objetivo: NO importar productos con stock = 0

Código:

```
while (($datos = fgetcsv($gestor)) !== FALSE) {
    // Filtrar productos agotados
    if ($datos[2] == 0) {
        continue; // Saltar esta iteración
    }

    $stmt→execute([$datos[0], $datos[1], $datos[2]]);
}
```

continue salta a la siguiente iteración del bucle sin ejecutar el código posterior (el INSERT).

Reto 9: Encuentra el Error de Rendimiento

Código LENTO (INCORRECTO):

```
while ($datos = fgetcsv($gestor)) {
    $stmt = $pdo→prepare("INSERT..."); // DENTRO DEL BUCLE
    $stmt→execute(...);
}
```

El error es poner prepare() DENTRO del while. Esto obliga a MySQL a compilar la consulta 1.000 veces.

Código RÁPIDO (CORRECTO):

```
$stmt = $pdo->prepare("INSERT..."); // FUERA DEL BUCLE
while ($datos = fgetcsv($gestor)) {
    $stmt->execute(...);
}
```

Explicación: prepare() compila la SQL una sola vez. Luego execute() reutiliza esa plantilla compilada.

Reto 10: Contador de Éxitos

Código:

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

while ($datos = fgetcsv($gestor)) {
    $stmt->execute(...);
    $contador++; // 2. Incrementar en cada inserción exitosa
}

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

La variable $contador se incrementa con ++ cada vez que se ejecuta una inserción.

Reto 11: Nivel Pro - Cambiar a UPDATE

Código UPDATE:

```
$sql = "UPDATE productos SET precio = ? WHERE nombre = ?";
$stmt = $pdo->prepare($sql);

while ($datos = fgetcsv($gestor)) {
    $stmt->execute([$datos[1], $datos[0]]); // precio, nombre
}
```

UPDATE modifica filas existentes. Buscamos por nombre (WHERE) y actualizamos el precio (SET). El orden de los parámetros en execute() debe

coincidir con los marcadores.

Reto 12: El Puerto Maldito

Problema: XAMPP usa puerto 3307

Solución:

```
$host = "localhost:3307";
```

Si MySQL no está en el puerto por defecto (3306), debemos especificar el puerto con dos puntos :puerto

Reto 13: El Chivato de Errores

Línea que falta:

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

Esta línea activa el modo de excepciones. Sin ella, si hay un error SQL, PHP no muestra nada (pantalla en blanco). Con esto, lanza una Exception visible.

Reto 14: Evitar el Crash

Solución:

```
$pdo->exec("CREATE TABLE IF NOT EXISTS productos ...");
```

IF NOT EXISTS hace la operación idempotente. Si la tabla ya existe, MySQL no hace nada (no da error). Puedes ejecutar el script mil veces sin problemas.

Reto 15: Reinicio de Fábrica

Comando completo:

```
$pdo->exec("TRUNCATE TABLE productos");
```

TRUNCATE vacía la tabla Y resetea el AUTO_INCREMENT a 1. Es más rápido que DELETE porque no borra fila a fila, sino que reconstruye la tabla.

Reto 16: Mapeo de Columnas (El Cruce)

Datos CSV: [0]=Precio, [1]=Nombre, [2]=Stock
BD espera: nombre, precio, stock

Código:

```
$stmt→execute([
    $datos[1], // Nombre (posición 1 del CSV)
    $datos[0], // Precio (posición 0 del CSV)
    $datos[2]  // Stock (posición 2 del CSV)
]);
```

Reordenamos el array para que coincida con el orden de las columnas en el INSERT.

Reto 17: Filtrado Lógico (El IF)

Código:

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

    if ($precio <= 0) {
        continue; // Saltar productos con precio 0 o negativo
    }

    $stmt→execute(...);
}
```

El operador <= verifica "menor o igual". continue omite el código restante del bucle y pasa a la siguiente iteración.

Reto 18: Atomicidad

Código completo:

```
try {
    $pdo→beginTransaction();
```

```
    // ... insertamos 1000 productos ...

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

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

commit() confirma los cambios. rollBack() deshace todo hasta el beginTransaction(). Esto garantiza que la BD nunca queda a medias.

Reto 19: Implementación del Singleton

Código completo:

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

    // 1. Constructor PRIVADO (nadie puede hacer new Database())
    private function __construct() {}

    public static function conectar() {
        // 2. Lógica Singleton
        if (self::$instance === null) {
            $opciones = [
                PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
            ];
            self::$instance = new PDO("mysql:host=localhost;dbname=tienda",
"root", "", $opciones);
        }

        // 3. Retorno
        return self::$instance;
    }
}
```

El constructor privado impide crear instancias desde fuera. El método estático conectar() crea la conexión solo la primera vez, luego siempre devuelve la misma.

Reto 20: Seguridad Frontend (XSS)

Código vulnerable:

```
echo "<h1>Producto: " . $producto->nombre . "</h1>"; // PELIGRO!
```

Solución segura:

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

htmlspecialchars() convierte los caracteres especiales HTML en entidades seguras. Si un atacante guardó `<script>alert('Hacked')</script>`, se mostrará como texto plano, no se ejecutará.

Reto 21: Trampa de Binding (Teoría)

Pregunta: ¿Qué valor se guarda en la BD?

B - "Eliminado"

bindParam() vincula la VARIABLE por referencia, no el valor. Cuando cambiamos `$estado = "Eliminado"` DESPUÉS del bind pero ANTES del execute, la consulta usa el valor actual ("Eliminado").

Por esto es más seguro usar execute([$valor]) o bindValue() que bindParam().

Reto 22: Integridad Referencial Manual

Orden correcto:

```
try {
    $pdo->beginTransaction();

    // PASO 1: Desvincular libros
    $stmt1 = $pdo->prepare("UPDATE libros SET autor_id = NULL WHERE au
tor_id = ?");
```

```
    $stmt1->execute([$id]);

    // PASO 2: Borrar al autor
    $stmt2 = $pdo->prepare("DELETE FROM autores WHERE id = ?");
    $stmt2->execute([$id]);

    // PASO 3: Guardar cambios
    $pdo->commit();

} catch (Exception $e) {
    // PASO 4: Deshacer si falla
    $pdo->rollBack();
}
```

Primero ponemos autor_id a NULL en los libros, luego borramos el autor. Si falla algo, rollBack() deshace ambos pasos.

Reto 23: FETCH_CLASS

Código:

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

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

// COMPLETA:
$misProductos = $stmt->fetchAll(PDO::FETCH_CLASS, 'Producto');

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

PDO::FETCH_CLASS inyecta los datos de las columnas en las propiedades de la clase. Así obtenemos objetos reales en lugar de arrays.

Reto 24: DDL Avanzado (Infraestructura)

Código SQL:

```
CREATE TABLE pedidos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    usuario_id INT,
    -- Definición de FK
    CONSTRAINT fk_user
    FOREIGN KEY (usuario_id) REFERENCES usuarios(id)
) ENGINE=InnoDB CHARACTER SET utf8mb4;
```

- ENGINE=InnoDB es obligatorio para transacciones y claves foráneas
- utf8mb4 soporta emojis (4 bytes)
- FOREIGN KEY vincula usuario_id con la tabla usuarios

BLOQUE: INFRAESTRUCTURA Y HERRAMIENTAS

Misión 1: La Base de Datos Moderna

Tareas:

1. Crear base de datos `socialmedia`
2. Configuración: `utf8mb4_unicode_ci` (para emojis)
3. Motor: `InnoDB` (para transacciones)

phpMyAdmin:

- Crear nueva BD → Cotejamiento: utf8mb4_unicode_ci
- Operaciones → Motor de almacenamiento: InnoDB

utf8mb4 permite 4 bytes (emojis). InnoDB soporta ACID y claves foráneas. MyISAM es obsoleto.

Misión 2: Seguridad y Permisos

Tarea: Crear usuario `appuser` con permisos limitados

Permisos a marcar:

- Datos: SELECT, INSERT, UPDATE, DELETE
- Estructura: DROP, TRUNCATE
- Administración: GRANT, SUPER

Principio de Mínimo Privilegio. Si un hacker entra, solo puede leer/modificar datos, no puede borrar tablas ni crear usuarios.

Misión 3: Ingeniería Inversa con DBeaver

Pasos:

1. Crear tabla `usuarios` (id, email, password) visualmente
2. Clic derecho → Generar SQL → DDL
3. Copiar el código generado

Código ejemplo:

```
CREATE TABLE usuarios (
    id INT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(100) NOT NULL,
    password VARCHAR(255) NOT NULL
) ENGINE=InnoDB;
```

Las herramientas profesionales generan SQL desde diagramas. Esto permite compartir código entre equipos.

Misión 4: Diagnóstico de Conexión (Error 1045)

Código:

```
$pass = "admin123";
```

Error: Access denied for user 'root'@'localhost'

1. ¿Por qué falla? Porque en XAMPP, root NO tiene contraseña por defecto. El código intenta conectar con "admin123" pero la BD espera "" (vacío).
2. ¿Qué hacer? Ir a phpMyAdmin → Privilegios → root → Cambiar contraseña a "admin123". O cambiar el código a `$pass = "";`

BLOQUE: ARQUITECTURA DE SOFTWARE

Misión 1: El Camaléon Multi-Motor

Código:

```
$motor = "sqlite"; // Variable de control

switch ($motor) {
    case "mysql":
        $dsn = "mysql:host=$host;dbname=$db;charset=utf8mb4";
        break;
    case "sqlite":
        $dsn = "sqlite:/ruta/absoluta/basedatos.sqlite"; // Completo
        break;
}
```

PDO es agnóstico. Cambiando solo el DSN, nuestra aplicación funciona con diferentes motores.

Misión 2: El Guardián de la Conexión (Singleton)

Código:

```
public static function conectar() {
    // Si la propiedad estática $instance es NULL
    if (self::$instance === null) {
        self::$instance = new PDO(...);
    }

    // Si ya existía, devolvemos la existente
    return self::$instance;
}
```

Solo se crea UNA conexión por todo el ciclo de vida. Evita "Too many connections".

Misión 3: Separación de Poderes (Repository)

Código:

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

        return $stmt→fetchObject('Producto'); // Devuelve objeto
    }
}
```

El patrón Repository centraliza todas las consultas SQL. El controlador solo pide datos, no escribe SQL.

Misión 4: Mapeo de Objetos (ORM Nativo)

Código:

```
$usuarios = $stmt→fetchAll(PDO::FETCH_CLASS, 'Usuario');
```

PDO::FETCH_CLASS convierte cada fila en un objeto de la clase Usuario. Así trabajamos con objetos inteligentes en lugar de arrays.

BLOQUE: SEGURIDAD Y DATOS AVANZADOS

Misión 1: El Caos Temporal (Fechas)

Problema: La BD guarda "2025-12-31" (ISO), pero el usuario español espera "31-12-2025"

Código:

```
$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('d-m-Y'); // Salida: 31-12-2025
```

NUNCA guardes fechas en formato español en la BD. Usa ISO (YYYY-MM-DD) y convértelas al mostrar.

Misión 2: El Botón del Pánico (Transacciones)

Código:

```
try {
    $pdo->beginTransaction(); // Congelamos guardado

    $stmt1->execute(...); // Restar Stock
    $stmt2->execute(...); // Crear Pedido

    $pdo->commit(); // SI TODO VA BIEN, confirmar

} catch (Exception $e) {
    $pdo->rollBack(); // SI ALGO FALLA, deshacer
    die("Error en la compra");
}
```

commit() = guardar partida. rollBack() = cargar partida anterior. Si falla el paso 2, el paso 1 también se deshace.

Misión 3: Defensa contra XSS (Frontend)

Problema: Un hacker guardó `<script>robarCookies()</script>` en un comentario

Código vulnerable:

```
echo $mensaje; // PELIGRO! Se ejecuta el JS
```

Código seguro:

```
echo htmlspecialchars($mensaje); // Convierte < en &lt;
```

htmlspecialchars() convierte caracteres especiales en entidades HTML. El código se muestra como texto, no se ejecuta.

Misión 4: El Arquitecto de Software (Directorios)

Estructura profesional:

1. **Database.php** (Clase de conexión) → `src/` o `config/`
2. **estilos.css** (Diseño web) → `public/css/`
3. **db_passwords.php** (Claves secretas) → `config/` (NUNCA en public)

La carpeta `public/` es la única visible al navegador. Los archivos sensibles van fuera.

BLOQUE: DISEÑO Y DESPLIEGUE

Misión 1: La Estrategia del Borrado

Tabla: Autores → Libros

- Configuración: `ON DELETE CASCADE`
- Efecto: Si borro al autor, sus libros se borran automáticamente

Tabla: Libros → Préstamos

- Configuración: `ON DELETE RESTRICT`
- Efecto: NO se puede borrar un libro si tiene préstamos activos (seguridad)

Código SQL:

```
-- Relación 1: CASCADE
ALTER TABLE libros
ADD CONSTRAINT fk_autor
FOREIGN KEY (autor_id) REFERENCES autores(id)
ON DELETE CASCADE;

-- Relación 2: RESTRICT
ALTER TABLE prestamos
ADD CONSTRAINT fk_libro
FOREIGN KEY (libro_id) REFERENCES libros(id)
ON DELETE RESTRICT;
```

Misión 2: El Arquitecto de Datos (Designer)

Tarea: Generar diagrama E-R automático

phpMyAdmin:

- Diseñador → Arrastra las tablas → Exportar a PDF/PNG

DBeaver:

- Clic derecho en BD → Ver diagrama ER

Si las líneas de unión no aparecen, faltan las claves foráneas (FOREIGN KEY).

Misión 3: Exportación Profesional (Deploy)

Problema: Si exportas sin DROP TABLE, al importar en producción dará error "Table already exists"

Solución:
phpMyAdmin → Exportar → Personalizado → Marcar:

- "Agregar sentencia DROP TABLE"

Resultado en el SQL:

```
DROP TABLE IF EXISTS productos;
CREATE TABLE productos (...); -- Ahora funciona aunque exista
```

DROP TABLE IF EXISTS borra la tabla vieja antes de crear la nueva. Esto hace el script idempotente.

Misión 4: El Simulacro de Incendio (Restauración)

Pasos:

1. DROP DATABASE bibliotecapro; (Destruir)
2. CREATE DATABASE bibliotecapro; (Vacía)
3. Importar → Seleccionar archivo .sql
4. Verificar: Mensaje verde "Importación ejecutada exitosamente"

Un backup no sirve de nada si nunca lo has probado. Este ejercicio simula un desastre real y su recuperación.