UT4: Utilización de Técnicas de Acceso a Datos

Índice

PHP es un lenguaje poderoso que soporta más de **15 sistemas gestores de bases de datos** (SGBD) como Oracle, SQL Server, PostgreSQL, y el más común: MySQL.

Evolución histórica

- **Hasta PHP 5 (Extensiones Nativas):** Se usaban funciones específicas para cada base de datos (ej. mysql_connect, pg_connect).
    - *Problema:* Si cambiabas de base de datos (de MySQL a Oracle), tenías que reescribir todo el código.
- **Desde PHP 5 (PDO):** Nace **PDO (PHP Data Objects)**.
    - *Ventaja:* Es una capa de abstracción. Usas la misma sintaxis de PHP sin importar qué motor de base de datos hay detrás.

Motores de Base de Datos: MySQL y MariaDB

Son los más usados en el entorno web.

- **MySQL:** Base de datos relacional muy popular, propiedad de Oracle.
- **MariaDB:** Un "fork" (derivado) de MySQL, totalmente libre (GPL), creado por el fundador original de MySQL.

Motores de almacenamiento internos (Storage Engines):

1. **MyISAM (o Aria):** Muy rápidos para lecturas, pero **no** soportan transacciones ni integridad referencial (relaciones fuertes).
2. **InnoDB (o XtraDB):** El estándar actual.
   Soporta **transacciones** (seguridad de datos) e integridad referencial. Un poco más lento, pero más seguro.

---

2. PDO: La Conexión a la Base de Datos

Para conectar con PDO, creamos una instancia de la clase PDO. El único parámetro obligatorio que necesitamos definir es el **DSN** (Data Source Name).

- Origen de datos (DSN). Es una cadena de texto que indica qué controlador se va a utilizar y a continuación, separadas por el carácter dos puntos, los parámetros específicos necesarios por el controlador, como por ejemplo el nombre o dirección IP del servidor y el nombre de la base de datos.
- Nombre de usuario con permisos para establecer la conexión.
- Contraseña del usuario.
- Opciones de conexión, almacenadas en forma de array.

```php
$dwes = new PDO('mysql:host=localhost;dbname=dwes', 'dwes', 'abc123.');
```

Si se utiliza **el controlador para MySQL**, los parámetros específicos para utilizar en la cadena DSN (separadas unas de otras por el carácter punto y coma) a continuación del prefijo mysql: son los siguientes:

- **host**: nombre o dirección IP del servidor.
- **port**: número de puerto TCP en el que escucha el servidor.
- **dbname**: nombre de la base de datos.
- **unix_socket**: socket de MySQL en sistemas Linux.

Para indicar que utilice codificación UTF-8 para los datos que se transmitan:

```php
$opciones = array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8");
$dwes =
new PDO('mysql:host=localhost;dbname=dwes','dwes','abc123.', $opciones);
```

Ejemplo de conexión controlando las excepciones:

```php
try{
     $connection= new PDO(
        dsn: 'mysql:host=localhost;dbname=dwes;charset=utf8mb4',
        username: 'dwes',
        password:'abc123.',
     )
     echo 'Conexión establecida correctamente';
 }catch (PDOException $e){
    echo match($e->getCode()){
        1049 => 'Base de datos no encontrada',
        1045 => 'Acceso denegado',
        2002 => 'Conexión rechazada',
        default => 'Error desconocido',
    };
}
```

Opciones de Configuración

Podemos pasar un array de opciones al conectar. Una muy importante es forzar UTF-8:

```php
$opciones = array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8");
```

Estructura Básica de una conexión y diferentes métodos

El objetivo es: **Crear la conexión UNA vez y pasarla a quien la necesite.**

El Patrón Singleton (La solución clásica)

Concepto: El Singleton es un patrón de diseño que asegura que una clase solo pueda crear **una única instancia** (objeto) de sí misma.

Cómo se ve el código:

```php
<?php
namespace App\Clases;

use PDO;
use PDOException;


class ConexionBD {
    // Variable estática para guardar la conexión única
    private static $instancia = null;

    // El constructor es privado para que nadie pueda hacer "new
ConexionBD()" desde fuera
    private function __construct() {}

    public static function getConexion() {

        // Si la instancia es null, significa que es la primera vez.
Creamos la conexión.
        if (self::$instancia === null) {

            try{
            $opciones = [PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION];
            // Aquí se crea la conexión REAL
            self::$instancia = new
PDO('mysql:host=localhost;dbname=dwes_01_nba;charset=utf8', 'root', '',
$opciones);

        } catch (PDOException $e){
            die("Error de conexión a la base de datos: " . $e-
>getMessage());
}
    }else{
        // Si ya existía, simplemente la devolvemos sin conectar de nuevo
        return self::$instancia;
    }
    }
}?>
```

Cómo se usa en tus archivos:

```php
<?php
require_once __DIR__ . '/../vendor/autoload.php';
use App\Clases\ConexionBD;

$pdo = ConexionBD::getConexion();

$stmt = $pdo->query("SELECT * FROM equipos");
$equipos = $stmt->fetchAll(PDO::FETCH_ASSOC);
foreach ($equipos as $equipo) {
    echo " - Nombre: " . $equipo['nombre'] . " - Ciudad: " .
$equipo['ciudad'] . "<br>";
}
?>
```

- **Ventaja:** Muy fácil de acceder desde cualquier parte (`ConexionBD::getConexion()`).
- **Desventaja:** Crea una "dependencia global". Es difícil de testear y si cambias la base de datos, tienes que tocar esta clase estática. Hoy en día se considera un poco "anticuado" para proyectos muy grandes, pero funciona.

Inyección de Dependencias (La solución profesional/moderna)

Concepto: En lugar de que tus clases busquen la conexión (como en el Singleton), tú se las entregas (se las "inyectas") cuando las creas.

Usaremos tres archivos:

1. `DatabaseFactory.php`: La clase encargada **solo** de fabricar la conexión.
2. `UsuarioModel.php`: La clase que contiene la lógica CRUD e **inyectamos** la conexión.
3. `index.php`: El archivo principal donde juntamos todo y probamos el código.

El ejemplo se basa en el uso de la siguiente tabla de base de datos:

SQL:

```sql
CREATE TABLE usuarios (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE
);
```

1. DatabaseFactory.php

Esta clase tiene un método estático. No guarda estado, simplemente le pides una conexión y te devuelve un objeto `PDO` configurado.

```php
<?php
namespace App\Database;
use PDO;
use PDOException;

class DatabaseFactory {

    public static function create(): PDO {
        // Configuración (en un proyecto real, esto vendría de variables de entorno)
        $host = 'localhost';
        $dbName = 'test_db';
        $user = 'root';
        $pass = '';
        $charset = 'utf8mb4';

        $dsn = "mysql:host=$host;dbname=$dbName;charset=$charset";

        $options = [
            PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION, // Importante para ver errores
            PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC, // Devuelve arrays asociativos
            PDO::ATTR_EMULATE_PREPARES => false, // Seguridad contra inyección SQL
        ];

        try {
            return new PDO($dsn, $user, $pass, $options);
        } catch (PDOException $e) {
            // En producción no hagas echo del error directamente por seguridad
            die("Error de conexión a la Base de Datos: " . $e->getMessage());
        }
    }
}
?>
```

2. UsuarioModel.php (El CRUD con Inyección)

Aquí es donde aplicamos la **Inyección de Dependencias**. Fíjate en el `__construct`. No creamos la conexión dentro, la **pedimos**.

```php
<?php

namespace App\Clases;
use PDO;

class UsuarioModel {
    private $pdo;

    // INYECCIÓN DE DEPENDENCIA:
    // Pedimos un objeto de tipo PDO. No nos importa de dónde viene,
    // solo necesitamos que sea una conexión válida.
    public function __construct(PDO $pdo) {
        $this->pdo = $pdo;
    }

    // --- CREATE ---
    public function registrar($nombre, $email) {
        try {
            $sql = "INSERT INTO usuarios (nombre, email) VALUES (:nombre,
:email)";
            $stmt = $this->pdo->prepare($sql);
            $stmt->execute([
                ':nombre' => $nombre,
                ':email' => $email
            ]);
            // Devolvemos el ID del usuario recién creado
            return $this->pdo->lastInsertId();
        } catch (PDOException $e) {
            echo "Error al registrar: " . $e->getMessage();
            return false;
        }
    }

    // --- READ (Todos) ---
    public function obtenerTodos() {
        $sql = "SELECT * FROM usuarios";
        $stmt = $this->pdo->query($sql);
        return $stmt->fetchAll();
    }

    // --- READ (Uno por ID) ---
    public function obtenerPorId($id) {
        $sql = "SELECT * FROM usuarios WHERE id = :id";
        $stmt = $this->pdo->prepare($sql);
```

```
        $stmt->execute([':id' => $id]);
        return $stmt->fetch();
    }

    // --- UPDATE ---
    public function actualizar($id, $nombre, $email) {
        try {
            $sql = "UPDATE usuarios SET nombre = :nombre, email = :email
WHERE id = :id";
            $stmt = $this->pdo->prepare($sql);
            return $stmt->execute([
                ':nombre' => $nombre,
                ':email' => $email,
                ':id' => $id
            ]);
        } catch (PDOException $e) {
            echo "Error al actualizar: " . $e->getMessage();
            return false;
        }
    }

    // --- DELETE ---
    public function eliminar($id) {
        try {
            $sql = "DELETE FROM usuarios WHERE id = :id";
            $stmt = $this->pdo->prepare($sql);
            return $stmt->execute([':id' => $id]);
        } catch (PDOException $e) {
            echo "Error al eliminar: " . $e->getMessage();
            return false;
        }
    }
}
?>
```

3. index.php (Probando todo)

Aquí verás el flujo completo: Crear la conexión UNA vez y usarla.

```
<?php
// Incluimos las clases
require_once 'DatabaseFactory.php';
require_once 'UsuarioModel.php';

use App\Database\DatabaseFactory;
use App\Models\UsuarioModel;
// ------------------------------------------------------------
// PASO 1: Fabricar la conexión (Solo lo hacemos una vez)
```

```php
// ------------------------------------------------------------
$miConexion = DatabaseFactory::create();
// ------------------------------------------------------------
// PASO 2: Inyectar la conexión al Modelo
// ------------------------------------------------------------
$usuarioModel = new UsuarioModel($miConexion);
echo "<h1>Prueba de CRUD con PHP y PDO</h1>";

// --- PRUEBA 1: CREATE ---
echo "<h3>1. Insertando Usuario...</h3>";
$nuevoId = $usuarioModel->registrar("Carlos Perez", "carlos@correo.com");
if ($nuevoId) {
    echo "Usuario creado con éxito. ID: " . $nuevoId . "<br>";
}

// --- PRUEBA 2: READ (Todos) ---
echo "<h3>2. Listando Usuarios...</h3>";
$usuarios = $usuarioModel->obtenerTodos();
echo "<pre>";
print_r($usuarios);
echo "</pre>";

// --- PRUEBA 3: UPDATE ---
echo "<h3>3. Actualizando Usuario (ID: $nuevoId)...</h3>";
$resultadoUpdate = $usuarioModel->actualizar($nuevoId, "Carlos Editado",
"nuevo_email@correo.com");

if ($resultadoUpdate) {
    echo "Usuario actualizado correctamente.<br>";
    // Verificamos el cambio leyendo solo ese usuario
    $usuarioActualizado = $usuarioModel->obtenerPorId($nuevoId);
    echo "Nombre actual: " . $usuarioActualizado['nombre'] . "<br>";
}

// --- PRUEBA 4: DELETE ---
echo "<h3>4. Eliminando Usuario...</h3>";
$usuarioModel->eliminar($nuevoId);
echo "Usuario eliminado.<br>";

// Verificamos que ya no existe
$usuariosRestantes = $usuarioModel->obtenerTodos();
echo "Total usuarios en base de datos: " . count($usuariosRestantes);

// ------------------------------------------------------------
// PASO FINAL: Cerrar conexión (Opcional, PHP lo hace solo)
// ------------------------------------------------------------
$miConexion = null;
?>
```

4. Seguridad y Buenas Prácticas: Archivos .env

Nunca se deben escribir las contraseñas directamente en el código PHP (hardcoding). La práctica profesional es usar un archivo de variables de entorno (.env) y la librería phpdotenv.

https://github.com/vlucas/phpdotenv

https://packagist.org/packages/vlucas/phpdotenv

Archivo .env (ejemplo):

```
DB_DSN=mysql:host=localhost;dbname=prueba;charset=utf8mb4
DB_USERNAME=root
DB_PASSWORD=secreto
```

Ejemplo de uso de phpdotenv para Patrón Singleton para la Conexión:

Explicación del código ConnectionPDODotenv:

1. **private static $connection**: Guarda la conexión para que nadie más la toque.
2. **load()**: Carga las variables del archivo .env.
3. **getConnection()**:
    - Si ya existe conexión, la devuelve.
    - Si no existe, la crea usando $_ENV['DB_DSN'], etc.
    - Configura el modo de error a ERRMODE_EXCEPTION (vital para ver fallos).

```php
<?php
require_once __DIR__ . '/vendor/autoload.php';

$dotenv = \Dotenv\Dotenv::createImmutable(__DIR__);
$dotenv->load();

final class ConnectionPDODotenv
{
    private static ?PDO $connection = null;

    final private function __construct() {}

    final public static function getConnection(): ?PDO
    {
        try {
```

```
            if ( ! self::$connection) {
                $opciones = array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET
NAMES utf8");
                self::$connection = new PDO(
                    dsn: $_ENV['DB_DSN'],
                    username: $_ENV['DB_USERNAME'],
                    password: $_ENV['DB_PASSWORD'],
                    options:$opciones
                );
                 self::$connection-
>setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
            }
        } catch (PDOException $e) {
            echo match ($e->getCode()) {
                1049 => 'Base de datos no encontrada',
                1045 => 'Acceso denegado',
                2002 => 'Conexión rechazada',
                default => 'Error desconocido',
            };
        }

        return self::$connection;
    }
     private function __clone() {}
}

$connection = ConnectionPDODotenv::getConnection();

if ($connection instanceof PDO)
{
    echo 'Conexión establecida correctamente';
}
```

- 💻 Hoja04_BBDD_01 (Ej 1,2,3,4)

5. Ejecución de Consultas

PDO distingue entre consultas que solo hacen acciones y consultas que piden datos.

A. Consultas de Acción (INSERT, DELETE, UPDATE) -> exec()

No devuelven filas de una tabla, devuelven el **número de filas afectadas**.

```
$registros=$dwes->exec('DELETE FROM stock WHERE unidades=0');
echo "<p>Se han borrado $registros registros.</p>";
```

B. Consultas de Selección (SELECT) -> query()

Devuelven un objeto PDOStatement que contiene los datos resultantes.

Si la consulta genera un conjunto de datos (SELECT) hay que utilizar el **método query**, que devuelve un objeto de la clase **PDOStatement**.

```
$dwes = new PDO("mysql:host=localhost;dbname=dwes", "dwes", "abc123.");
$resultado = $dwes->query("SELECT producto, unidades FROM stock");
```

---

6. Obtención de Resultados: El método fetch

Una vez tienes el objeto $resultado, necesitas extraer los datos fila por fila usando fetch().

```
$resultado = $dwes->query('SELECT campo1, campo2 FROM tabla WHERE
condicion');
while ($registro = $resultado->fetch())
{
echo $registro['campo1'].": ".$registro['campo2']."<br />";
}
```

Modos de obtención (fetch styles):

1. **PDO::FETCH_ASSOC**: Devuelve un array asociativo (con nombres de columnas). *El más usado.*
    - $fila['producto']
2. **PDO::FETCH_NUM**: Devuelve array numérico.
    - $fila[0]
3. **PDO::FETCH_OBJ**: Devuelve un objeto anónimo.
    - $fila->producto

4. **PDO::FETCH_BOUND**: Vincula columnas a variables PHP. Devuelve true y asigna los valores del registro a variables, según se indique con el método `bindColumn`. Este método debe ser llamado una vez por cada columna, indicando en cada llamada el número de columna (empezando en 1) y la variable a asignar.

```php
$resultado = $dwes->query('SELECT campo1, campo2 FROM tabla WHERE
condicion');
$resultado->bindColumn(1, $campo1);
$resultado->bindColumn(2, $campo2);
while ($registro = $resultado->fetch(PDO::FETCH_BOUND))
{
    echo $campo1.": ".$campo2."<br />";
```

Ejemplo FETCH_OBJ:

```php
$resultado = $dwes->query('SELECT campo1, campo2 FROM tabla WHERE
condicion');
while ($registro = $resultado->fetch(PDO::FETCH_OBJ))
{
    echo $registro->campo1.": ".$registro->campo2."<br />";
}
```

- **PDO::FETCH_BOTH**: devuelve un array con claves numéricas y asociativas. Es el comportamiento por defecto.
- **PDO::FETCH_LAZY**: devuelve tanto el objeto como el array con clave dual anterior.

💻 Hoja04_BBDD_01(Ej 5,6,7 y 8)