Introducción a los DAOs en PHP

Imaginemos que estamos desarrollando una aplicación web en PHP que debe interactuar con una base de datos. Podríamos escribir directamente consultas SQL en cada parte del código donde las necesitemos. Sin embargo, esto genera varios problemas:

1. **Repetición de código**: Tendríamos que escribir consultas similares varias veces.
2. **Dificultad para realizar cambios**: Si cambia la estructura de la base de datos, tendríamos que actualizar todas las consultas manualmente.
3. **Mayor propensión a errores**: Es fácil introducir errores en consultas escritas manualmente y repetidas.
4. **Falta de abstracción**: La lógica de negocio se mezcla con las consultas SQL, haciendo que el código sea más difícil de leer y mantener.

Para resolver estos problemas, utilizamos un patrón llamado **DAO (Data Access Object)**.

¿Qué es un DAO?

Un DAO es una clase que actúa como intermediario entre nuestra aplicación y la base de datos. Su principal función es encapsular todo el acceso a la base de datos y proporcionar métodos específicos para las operaciones que queremos realizar, como insertar, actualizar, eliminar o consultar datos.

Con un DAO:

- **No escribimos SQL directamente en nuestra lógica de negocio**.
- Podemos **reutilizar métodos** para interactuar con la base de datos.
- Mantenemos el código **modular y organizado**.
- Aislamos la lógica de negocio de la lógica de acceso a datos.

Creación de DAOs en PHP: Ejemplo Práctico

Clase Base `DB`

La base de cualquier DAO es una conexión centralizada a la base de datos. Aquí entra en juego nuestra clase `DB`. Esta clase permite conectarse a la base de datos utilizando PDO, que es una extensión de PHP para trabajar con bases de datos de forma segura y eficiente.

Características clave de `DB`:

1. Gestiona la conexión a la base de datos con los parámetros de configuración (`host`, `user`, `pass`, `base`).
2. Ofrece dos métodos para ejecutar consultas:
    - **`ConsultaSimple`**: Para operaciones como `INSERT`, `UPDATE` y `DELETE`.
    - **`ConsultaDatos`**: Para recuperar datos, como en un `SELECT`.
3. La conexión se cierra automáticamente tras cada operación.

Relación de Herencia entre `DB` y `DaoSituaciones`

La herencia es clave en este diseño. La clase `DaoSituaciones` extiende la clase `DB`, lo que significa que hereda todas sus propiedades y métodos. Esto evita duplicar código y nos permite reutilizar la lógica de conexión y consulta en múltiples DAOs.

Constructor del DAO:

```
public function __construct($base)
{
    parent::__construct($base); // Llama al constructor de la clase DB
}
```

Al utilizar `parent::__construct($base)`, aseguramos que el constructor de la clase base `DB` se ejecuta al crear un objeto `DaoSituaciones`. Este enfoque centraliza la lógica de conexión en `DB`, haciendo que `DaoSituaciones` pueda enfocarse en las operaciones específicas de la tabla `Situaciones`.

Métodos del DAO `DaoSituaciones`

A continuación, explicamos todos los métodos implementados en el DAO, haciendo énfasis en cómo encapsulan las operaciones sobre la tabla `Situaciones` utilizando los métodos heredados de la clase `DB`.

Método `listar`

Recupera todas las filas de la tabla `Situaciones` y las convierte en objetos `Situacion`.

```
public function listar()
{
    $consulta = "select * from Situaciones";
    $this->situaciones = [];
    $this->ConsultaDatos($consulta);

    foreach ($this->filas as $fila) {
        $situ = new Situacion();
        $situ->__set("Id", $fila["Id"]);
        $situ->__set("Nombre", $fila["Nombre"]);
        $this->situaciones[] = $situ;
    }
}
```

Explicación:

1. Ejecutamos una consulta `SELECT` con el método heredado `ConsultaDatos`.
2. Recorremos las filas devueltas y creamos un objeto `Situacion` por cada fila.
3. Guardamos esos objetos en el array `$situaciones`.

Método `obtener`

Obtiene una fila específica de la tabla según el `Id`.

```
public function obtener($Id)
{
    $consulta = "select * from Situaciones where Id=:Id";
    $param = [":Id" => $Id];
    $this->ConsultaDatos($consulta, $param);

    if (count($this->filas) == 1) {
        $fila = $this->filas[0];
        $situ = new Situacion();
        $situ->__set("Id", $fila["Id"]);
        $situ->__set("Nombre", $fila["Nombre"]);
        return $situ;
    } else {
        echo "<b>El Id introducido no corresponde con ninguna situación
administrativa</b>";
        return null;
    }
}
```

Explicación:

1. Utilizamos un parámetro dinámico `:Id` para buscar una fila específica.
2. Si la consulta devuelve exactamente una fila, creamos un objeto `Situacion` con sus datos.
3. Si no encuentra filas, mostramos un mensaje de error.

Método `borrar`

Elimina una fila de la tabla según el `Id`.

```
public function borrar($Id)
{
    $consulta = "delete from Situaciones where Id=:Id";
    $param = [":Id" => $Id];
    $this->ConsultaSimple($consulta, $param);
}
```

Explicación:

1. Ejecutamos una consulta `DELETE` con un parámetro dinámico.
2. No se devuelve nada, ya que el método solo modifica la tabla.

Método `insertar`

Inserta una nueva fila en la tabla utilizando un objeto `Situacion`.

```
public function insertar($situ)
{
    $consulta = "insert into Situaciones values(:Id,:Nombre)";
    $param = [
        ":Id" => $situ->__get("Id"),
```

```
        ":Nombre" => $situ->__get("Nombre")
    ];
    $this->ConsultaSimple($consulta, $param);
}
```

Explicación:

1. La consulta usa parámetros dinámicos para insertar datos.
2. Los valores de los parámetros se obtienen de las propiedades del objeto `Situacion`.

Método `actualizar`

Modifica los datos de una fila existente según su `Id`.

```
public function actualizar($situ)
{
    $consulta = "update Situaciones set Nombre=:Nombre where Id=:Id";
    $param = [
        ":Id" => $situ->__get("Id"),
        ":Nombre" => $situ->__get("Nombre")
    ];
    $this->ConsultaSimple($consulta, $param);
}
```

Explicación:

1. La consulta `UPDATE` actualiza el campo `Nombre` de una fila específica.
2. Los valores se obtienen del objeto `Situacion`.

Ventajas de Usar Herencia para Crear DAOs

1. **Centralización de la Lógica de Conexión**:
    - `DB` gestiona toda la lógica de conexión y consultas.
    - Los DAOs heredan esta funcionalidad, evitando la duplicación de código.
2. **Especialización**:
    - Cada DAO se centra en una tabla específica, como `DaoSituaciones`, que encapsula las operaciones sobre la tabla `Situaciones`.
3. **Reutilización y Escalabilidad**:
    - Podemos crear nuevos DAOs para otras tablas (como `DaoUsuarios` o `DaoProductos`) simplemente heredando de `DB`.

Relación con Frameworks como Laravel

Frameworks como Laravel implementan este patrón mediante **modelos**. Por ejemplo:

- En Laravel, `Situacion::all()` realiza un `SELECT * FROM Situaciones`, similar a nuestro método `listar`.

- Esto se logra gracias a **Eloquent**, que automatiza gran parte del trabajo que hemos hecho manualmente en este ejemplo.

Conclusión

El uso de DAOs permite mantener el código organizado, seguro y escalable. Nuestra implementación demuestra cómo:

1. La herencia centraliza la lógica en la clase base `DB`.
2. Los DAOs encapsulan operaciones específicas de cada tabla.
3. Las entidades (`Situacion`) representan las filas como objetos.

Este enfoque es robusto, fácil de mantener y escalable, ideal tanto para proyectos pequeños como para aplicaciones más complejas.