Desarrollo Web en Entorno Servidor

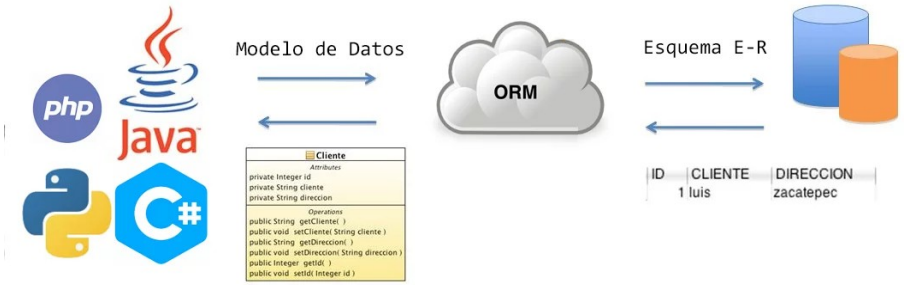
7.- PHP – ORM

IES Severo Ochoa

Índice

- ORM
- Frameworks
- Eloquent
  - La clase Model
  - Consultas
  - Operaciones
- Integrando Eloquent en la Tienda
- Relaciones

ORM



- *Object Relational Mapping*
- Automatiza el mapeo entre el modelo de datos y el esquema entidad-relación.
- Ventajas:
  - Agiliza el desarrollo y reduce la cantidad de código, permitiendo generar las clases entidad a partir de una base de datos existente, o viceversa.
  - Independiza la aplicación del tipo de base de datos (MariaDB / MySQL, Oracle, PostgreSQL…), permitiendo la obtención de datos usando objetos y métodos sin escribir consultas SQL.

No es oro todo lo que reluce...

- Penalización en el rendimiento → las consultas no están optimizadas.
- No existe un estándar homogéneo.
- Curva de aprendizaje, especialmente para modelar las relaciones.

Frameworks

- *Esqueleto y utilidades/librerías para la creación de una aplicación.*

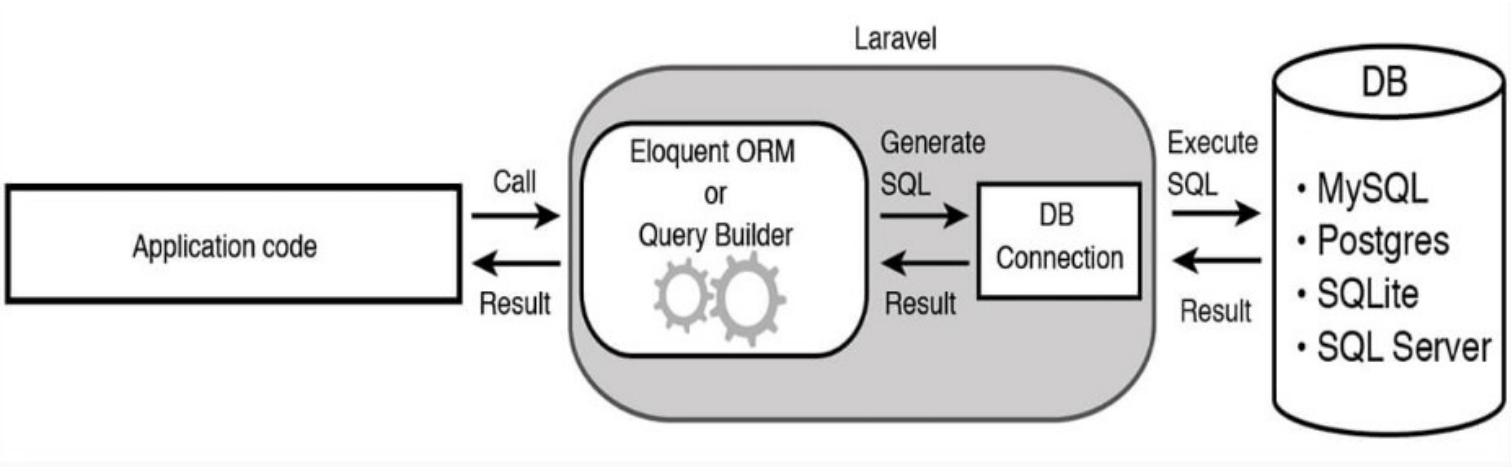
- ***Doctrine***, forma parte de *Symphony*
  - *https://www.doctrine-project.org/projects/doctrine-orm/en/2.8/index.html*
- ***Eloquent***, forma parte de *Laravel*
  - *https://laravel.com/docs/8.x/eloquent*
- *Propel*
  - http://propelorm.org/
- En Java
  - Hibernate, MyBatis

Eloquent

- `composer require illuminate/database`
- Más sencillo que Doctrine
- CoC - Convención sobre configuración
- Patrón ***Active Record***:
  - Una tabla se asocia a una clase.
  - Una instancia de un objeto está ligada a un único registro (tupla) en la tabla.
  - Después de crear y grabar un objeto, un nuevo registro se añade a la tabla. Cualquier objeto cargado obtiene su información a partir de la base de datos. Cuando un objeto se actualiza, un registro correspondiente en la tabla también es actualizado.

Eloquent II

- Además del ORM, ofrece:
  - Un adaptador que encapsula PDO → `Query Builder`
  - Un generador de tablas → `Schema`



eloquent.php

```php
<?php
include "vendor/autoload.php";

use Illuminate\Database\Capsule\Manager as Capsule;

$capsule = new Capsule;
$capsule->addConnection([
    'driver'    => 'mysql',
    'host'      => 'localhost',
    'database'  => 'dwes',
    'username'  => 'dwes',
    'password'  => 'abc123.',
    'charset'   => 'utf8',
    'collation' => 'utf8_unicode_ci',
    'prefix'    => '',
]);
// Permite el acceso mediante métodos estáticos
$capsule->setAsGlobal();
// Carga el framework
$capsule->bootEloquent();
```

Model

- Toda entidad de nuestro modelo debe heredar de `Model`.
- Al hacerlo, añadirá automáticamente lo métodos `save, update, delete, get, all` ...

```php
use Illuminate\Database\Eloquent\Model;

class Tienda extends Model {}

$tiendas = Tienda::all();
```

Model II - convenciones

- La tabla debe coincidir con el nombre del modelo pero en **plural y minúsculas**.
  - Sino: `protected $table = 'nombreTabla';`
- La clave primaria debe
  - denominarse **id**
    - Sino: `protected $primaryKey = 'nombreCampo'`
  - Ser un **entero autoincrementable**
    - Sino: `protected $keyType = 'string'`

```
class Tienda extends Model {
    protected $table = 'tienda';
    protected $primaryKey = 'cod';
    // protected $keyType = 'string';
}
```

Model III - Timestamps

- *Eloquent* espera que todas las tablas tenga dos columnas de tipo fecha nombradas `created_at` y `updated_at`, las cuales rellenará automáticamente al crear y actualizar un registro respectivamente.
- Si nuestras tablas no tienen esas columnas:

```
public $timestamps = false;
```

- Si las tenemos pero tienen otro nombre:

```
const CREATED_AT = 'fCreacion';
const UPDATED_AT = 'fActualizacion';
```

Consultas

https://laravel.com/api/8.x/Illuminate/Database/Query/Builder.html

```php
include "eloquent.php";

// Todas las tiendas, a partir del modelo
$tiendasM = Tienda::all();
// Todas las tiendas mediante QueryBuilder (más rápido)
$tiendasQB = Tienda::get();

// Tienda por PK
$tienda = Tienda::find(1);
// Tienda por PK, si no lo encuentra lanza ModelNotFoundException
$tienda = Tienda::findOrFail(1);

// 5 primeras tiendas con tlf 600100100 ordenadas por nombre
$tiendasMovil = Tienda::where("tlf","600100100")->orderBy("nombre")
                        ->take(5)->get();
// Primera tienda con tlf 600100100
$tiendaMovil = Tienda::firstWhere("tlf","600100100");

// Cantidad de tiendas
$count = Tienda::where("tlf","600100100")->count();
// Precio más alto
$max = Producto::where("familia", "CAMARA")->max('PVP');
```

Inserciones y actualizaciones

ies severo ochoa

```php
// Para insertar, creamos un Model y luego save
$tienda = new Tienda();
$tienda->nombre = "Elotienda";
$tienda->save();

// También se pueden crear mediante create
$tienda = Tienda::create([ "nombre" => "Elotienda 2"]);

// Para modificar, recuperamos un Model y luego save
$tienda1 = Tienda::find(1);
$tienda1->tlf = "966100100";
$tienda1->save();

// Modificación masiva
// En el modelo añadir
// protected $fillable = ['campoModificable1', ...];
Producto::where('familia', "MP3")
      ->update(['PVP' => 123456]);

// Cálculos - increment, decrement
Producto::where('familia', "MP3")
      ->increment('PVP', 500);
```

Inserciones y actualizaciones II

ies severo ochoa

```php
// firstOrCreate: busca, y si no lo encuentra, lo crea
$tienda3 = Tienda::firstOrCreate(["nombre" => "Elotienda3"]);
// firstOrNew: busca, y si no lo encuentra, lo instancia
$tienda4 = Tienda::firstOrNew(["nombre" => "Elotienda4"]);
$tienda4->save();

// updateOrCreate, si lo encuentra (Elotienda3) actualiza su tlf,
// si no, lo inserta
$tienda5 = Tienda::updateOrCreate(
    ['nombre' => 'Elotienda3'],
    ['tlf' => "966123123"]
);

// upsert (valoresInsertar, columnasIndentificadoras, columnasModificar)
Tienda::upsert([
    ["nombre" => "Elotienda3", "tlf" => "966123123"],
    ["nombre" => "Elotienda4", "tlf" => "966123124"],
    ["nombre" => "Elotienda5", "tlf" => "966123125"] ]
    , ["nombre"]
    , ["tlf"]
);
```

Borrados

```php
// Buscamos el modelo y la borramos
$tiendaBorrar1 = Tienda::findOrFail(33);
$tiendaBorrar1->delete();

Tienda::findOrFail(["nombre" => "Elotienda4"])->delete();

// Borramos por su PK
Tienda::destroy(4);
Tienda::destroy([5,6]);
```

- *Eloquent* permite trabajar con borrados *soft*, (utiliza una columna `deleted_at`): https://laravel.com/docs/8.x/eloquent#soft-deleting

Integrando *Eloquent* en nuestra aplicación

ies severo ochoa

- Para introducir *Eloquent* y usarlo como alternativa a *PDO*, debemos:
  - Crear una factoría para centralizar la creación/configuración de *Eloquent*
  - Crear diferentes clase que hereden de `Model` (dependerá de las relaciones 1:1, 1:N, N:M).
  - Crear una nueva implementación de nuestros repositorios

EloquentFactory

```php
use Illuminate\Database\Capsule\Manager as Capsule;
class EloquentFactory {
    public static function getManager(): Manager {
        $config = include "config/configuracion.php";

        $capsule = new Capsule;
        $capsule->addConnection([
        'driver'    => $config["db"]["driver"],
        'host'      => $config["db"]["host"],
        'database' => $config["db"]["name"],
        'username' => $config["db"]["username"],
        'password' => $config["db"]["password"],
        'charset'   => 'utf8',
        'collation' => 'utf8_unicode_ci',
        'prefix'    => '',
        ]);

        $capsule->setAsGlobal();
        $capsule->bootEloquent();

        return $capsule;
    }
}
```

TiendaModel

```php
class TiendaModel extends Model
{
    protected $table = 'tienda';
    protected $primaryKey = 'cod';
    protected $fillable = ['nombre', 'tlf'];
    public $timestamps = false;

    public static function toEntity(TiendaModel $t) : Tienda {
        $result = new Tienda($t->cod);
        $result->setNombre($t->nombre);
        $result->setTelefono($t->tlf);
        return $result;
    }
}
```

EloquentTiendaRepository I

ies severo ochoa

```
class EloquentTiendaRepository implements TiendaRepository {
    private $log;
    private $capsule;

    public function __construct(LoggerInterface $logger) {
        $this->log = $logger;
        $this->capsule = EloquentFactory::getManager();
    }

    public function getTiendaPorCodigo(int $cod): ?Tienda {
        $result = null;
        try {
            $result = TiendaModel::findOrFail($cod);
        } catch (ModelNotFoundException $e){
            $this->log->warning("No se han podido encontrar la tienda
$cod", [$e]);
            throw new TiendaException("Error al encontrar las tienda", $e-
>getCode(), $e);
        }
        return TiendaModel::toEntity($result);
    }
```

EloquentTiendaRepository II

```
public function getAll(): ?array {
    $result = null;

    try {
        $result = TiendaModel::get()->map( function (TiendaModel $model){
            return TiendaModel::toEntity($model); } );
    } catch (ModelNotFoundException $e){
        $this->log->warning("No se han podido listar las tiendas", [$e]);
        throw new TiendaException("Error al listar las tiendas", $e-
>getCode(), $e);
    }

    return $result->toArray();
}
```

- `map()` se ejecuta por cada elemento de la colección, y transforma cada ejemplar de `TiendaModel` en una instancia de `Tienda` (entity).

¿Alguna pregunta?