Modelos de datos mediante ORM

El mapeado objeto-relacional (más conocido por su nombre en inglés, *Object-Relational mapping*, o por sus siglas ORM) es una técnica de programación para convertir datos entre un lenguaje de programación orientado a objetos y una base de datos relacional como motor de persistencia. Esto posibilita el uso de las características propias de la orientación a objetos, podremos acceder directamente a los campos de un objeto para leer los datos de una base de datos o para insertarlos o modificarlos.

Laravel incluye su propio sistema de ORM llamado *Eloquent*, el cual nos proporciona una manera elegante y fácil de interactuar con la base de datos. Para cada tabla de la base datos tendremos que definir su correspondiente modelo, el cual se utilizará para interactuar desde código con la tabla.

Definición de un modelo

Por defecto los modelos se guardarán como clases PHP dentro de la carpeta `app`, sin embargo Laravel nos da libertad para colocarlos en otra carpeta si queremos, como por ejemplo la carpeta `app/Models`. Pero en este caso tendremos que asegurarnos de indicar correctamente el espacio de nombres.

Para definir un modelo que use *Eloquent* únicamente tenemos que crear una clase que herede de la clase `Model`:

```
<?php
namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    //...
}
```

Sin embargo es mucho más fácil y rápido crear los modelos usando el comando `make:model` de Artisan:

```
php artisan make:model User
```

Este comando creará el fichero `User.php` dentro de la carpeta `app` con el código básico de un modelo que hemos visto en el ejemplo anterior.

Convenios en *Eloquent*

Nombre

En general el nombre de los modelos se pone en singular con la primera letra en mayúscula, mientras que el nombre de las tablas suele estar en plural. Gracias a esto, al definir un modelo no es necesario indicar el nombre de la tabla asociada, sino que *Eloquent* automáticamente buscará la tabla transformando el nombre del modelo a minúsculas y buscando su plural (en inglés). En el ejemplo anterior que hemos creado el modelo `User` buscará la tabla de la base de datos llamada `users` y en caso de no encontrarla daría un error.

Si la tabla tuviese otro nombre lo podemos indicar usando la propiedad protegida `$table` del modelo:

```
<?php
namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    protected $table = 'my_users';
}
```

Clave primaria

Laravel también asume que cada tabla tiene declarada una clave primaria con el nombre `id`. En el caso de que no sea así y queramos cambiarlo tendremos que sobrescribir el valor de la propiedad protegida `$primaryKey` del modelo, por ejemplo: `protected $primaryKey = 'my_id';`.

> Es importante definir correctamente este valor ya que se utiliza en determinados métodos de *Eloquent*, como por ejemplo para buscar registros o para crear las relaciones entre modelos.

Timestamps

Otra propiedad que en ocasiones tendremos que establecer son los *timestamps* automáticos. Por defecto *Eloquent* asume que todas las tablas contienen los campos `updated_at` y `created_at` (los cuales los podemos añadir muy fácilmente con *Schema* añadiendo `$table->timestamps()` en la migración). Estos campos se actualizarán automáticamente cuando se cree un nuevo registro o se modifique. En el caso de que no queramos utilizarlos (y que no estén añadidos a la tabla) tendremos que indicarlo en el modelo o de otra forma nos daría un error. Para indicar que no los actualice automáticamente tendremos que modificar el valor de la propiedad pública `$timestamps` a *false*, por ejemplo: `public $timestamps = false;` .

A continuación se muestra un ejemplo de un modelo de *Eloquent* en el que se añaden todas las especificaciones que hemos visto:

```
<?php
namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    protected $table = 'my_users';
    protected $primaryKey = 'my_id'
    public $timestamps = false;
}
```

Uso de un modelo de datos

Una vez creado el modelo ya podemos empezar a utilizarlo para recuperar datos de la base de datos, para insertar nuevos datos o para actualizarlos. El sitio correcto donde realizar estas acciones es en el controlador, el cual se los tendrá que pasar a la vista ya preparados para su visualización.

Es importante que para su utilización indiquemos al inicio de la clase el espacio de nombres del modelo o modelos a utilizar. Por ejemplo, si vamos a usar los modelos `User` y `Orders` tendríamos que añadir:

```
use App\User;
use App\Orders;
```

Consultar datos

Para obtener todas las filas de la tabla asociada a un modelo usaremos el método `all()` :

```
$users = User::all();

foreach( $users as $user ) {
    echo $user->name;
}
```

Este método nos devolverá un array de resultados, donde cada item del array será una instancia del modelo `User`. Gracias a esto al obtener un elemento del array podemos acceder a los campos o columnas de la tabla como si fueran propiedades del objeto (`$user->name`).

> Nota: Todos los métodos que se describen en la sección de "Constructor de consultas" y en la documentación de Laravel sobre "Query Builder" también se pueden utilizar en los modelos Eloquent. Por lo tanto podremos utilizar *where, orWhere, first, get, orderBy, groupBy, having, skip, take,* etc. para elaborar las consultas.

Eloquent también incorpora el método `find($id)` para buscar un elemento a partir del identificador único del modelo, por ejemplo:

```
$user = User::find(1);
echo $user->name;
```

Si queremos que se lance una excepción cuando no se encuentre un modelo podemos utilizar los métodos `findOrFail` o `firstOrFail`. Esto nos permite capturar las excepciones y mostrar un error 404 cuando sucedan.

```
$model = User::findOrFail(1);

$model = User::where('votes', '>', 100)->firstOrFail();
```

A continuación se incluyen otros ejemplos de consultas usando Eloquent con algunos de los métodos que ya habíamos visto en la sección "Constructor de consultas":

```
// Obtener 10 usuarios con más de 100 votos
$users = User::where('votes', '>', 100)->take(10)->get();

// Obtener el primer usuario con más de 100 votos
$user = User::where('votes', '>', 100)->first();
```

También podemos utilizar los métodos agregados para calcular el total de registros obtenidos, o el máximo, mínimo, media o suma de una determinada columna. Por ejemplo:

```
$count = User::where('votes', '>', 100)->count();
$price = Orders::max('price');
$price = Orders::min('price');
$price = Orders::avg('price');
$total = User::sum('votes');
```

Insertar datos

Para añadir una entrada en la tabla de la base de datos asociada con un modelo simplemente tenemos que crear una nueva instancia de dicho modelo, asignar los valores que queramos y por último guardarlos con el método `save()` :

```
$user = new User;
$user->name = 'Juan';
$user->save();
```

Para obtener el identificador asignado en la base de datos después de guardar (cuando se trate de tablas con índice auto-incremental), lo podremos recuperar simplemente accediendo al campo `id` del objeto que habíamos creado, por ejemplo:

```
$insertedId = $user->id;
```

Actualizar datos

Para actualizar una instancia de un modelo es muy sencillo, solo tendremos que recuperar en primer lugar la instancia que queremos actualizar, a continuación modificarla y por último guardar los datos:

```
$user = User::find(1);
$user->email = 'juan@gmail.com';
$user->save();
```

Borrar datos

Para borrar una instancia de un modelo en la base de datos simplemente tenemos que usar su método `delete()` :

```
$user = User::find(1);
$user->delete();
```

Si por ejemplo queremos borrar un conjunto de resultados también podemos usar el método `delete()` de la forma:

```
$affectedRows = User::where('votes', '>', 100)->delete();
```

Más información

Para más información sobre como crear relaciones entre modelos, *eager loading*, etc. podéis consultar directamente la documentación de Laravel en:

http://laravel.com/docs/5.1/eloquent