Cómo usar modelos de Eloquent en Laravel 5

Cómo podemos usar modelos para recuperar información de las tablas de la base de datos, con el ORM Eloquent en Laravel 5.

Ahora que sabemos crear y configurar nuestros modelos podemos comenzar a usarlos para recuperar información de la base de datos. Esta es una operación sencilla, ya que realmente nuestros modelos realizarán todo el trabajo pesado de conectarse a la base de datos y ejecutar las correspondientes consultas para obtener o modificar información.

Parte de la operativa de acceso a los datos ya se conoce, puesto que le dimos un repaso inicial en el capítulo de QueryBuilder. En el caso de los modelos, cuando recuperas información tienes casi los mismos métodos que conoces de QueryBuilder, con la diferencia que no necesitas especificar la tabla, puesto que ya la sabe el modelo. Algunas cosas cambian pero pocas, como verás a continuación.

Obviamente, para que esto funcione tendrás que tener creado no solo el modelo, sino también la tabla correspondiente asociada en el sistema gestor de base de datos, mediante la correspondiente migración.

Namespace del modelo

Recuerda que los modelos están creados dentro de la carpeta "app" de la aplicación, sueltos ahí. Sus clases forman parte del Namespace App y para usarlos primero debemos indicar que vamos a usar tal espacio de nombres.

```
useApp\User;
```

Eso nos indicará que estamos usando un modelo llamado User, que está en la carpeta "app", como todos los modelos. Gracias al Autoload se cargará ella sola. A partir de entonces ese modelo lo podremos usar directamente con el nombre de la clase.

Nota: Si decidimos no declarar el "use" del namespace, podemos usar el modelo igualmente, siempre que indiquemos el nombre de la clase predecida de su espacio de nombres. "App\User".

Usar un modelo

Un modelo se podrá usar desde cualquiera de las clases de Laravel, como middlewares o seeders, pero lo más habitual es usarlo desde los controladores. Una vez declarado su ruta mediante el espacio de nombres podremos ejecutar diversos métodos generalmente estáticos para recuperar información.

Recuperar todos los registros:
Podremos recuperar todos los registros de una tabla con el método all().

```
Articulo::all();
```

Lo que obtenemos será una colección, que podremos usar para lo que sea necesario. Por ejemplo lo podríamos enviar a una vista para que presentarlo en pantalla. Eso lo puedes ver en el siguiente controlador.

```
<?php
namespaceApp\Http\Controllers;

useIlluminate\Http\Request;
useApp\Http\Requests;
useApp\Articulo;

classArticulosControllerextendsController
{
        publicfunctionindex(){
                $articulos=Articulo::all();
                returnview('articulo',[
                        'articulos'=>$articulos
                ]);
        }
}
```

Nota: El tema de las colecciones es muy interesante. Las podemos usar como si fueran arrays simples, pero realmente son objetos cuyas funcionalidades van mucho más allá, permitiendo una gama importante de operaciones. Hablaremos más adelantes sobre colecciones con detalle.

Recuperar un registro:
Si lo que quieres hacer es buscar un registro en concreto de este modelo existe un método muy sencillo que permite recuperar un único elemento dado un id de clave primaria.

```
Articulo::find(18);
```

Esto nos devolverá los datos del artículo con identificador 18. En concreto lo que devuelve esta función es un modelo, un objeto de la clase App\Articulo. Como es un modelo dispondremos de los métodos que nos facilitan los modelos de Eloquent, pero también lo podremos usar como si fuera un simple array con los valores del artículo en cuestión que acabamos de recuperar. Por ejemplo podríamos pasar esos datos a una vista, con el siguiente código de función en el controlador de antes "ArticulosController".

```
publicfunctionmuestraId($id){
        $articulo=Articulo::find($id);
        returnview('articulo.muestra',[
                'articulo'=>$articulo
        ]);
}
```

Recuperar artículos mediante condiciones complejas:
Como hemos dicho, los modelos permiten una gama de métodos que ya se conocen de QueryBuilder, así que podemos usarlos para acceder a juegos de datos más específicos.

Con este método estaremos recibiendo una colección con todos los artículos escritos dentro del último año.

```
publicfunctionindex(){
        $articulos=Articulo::where('fecha','>',time()-(365*24*3600))-
>get();
        returnview('articulo.index',[
                'articulos'=>$articulos
        ]);
}
```

Tal como aprendimos, mediante QueryBuilder podríamos encadenar diversos métodos para realizar búsquedas más complejas. Ahora vamos a variar este mismo código para sacar los 10 últimos artículos (según su fecha de publicación) que comienzan por la palabra "introducción"

```
publicfunctionindex(){
        $articulos=Articulo::where('nombre_articulo','like','introduccion%
')
                ->orderBy('fecha','desc')
                ->take(10)
                ->get();
        returnview('articulo.index',[
                'articulos'=>$articulos
        ]);
}
```

Acceso a los campos de un registro mediante propiedades de un modelo

Cuando tenemos un objeto modelo, una instancia de la clase de un modelo de Eloquent, podemos acceder a sus propiedades y así estaremos accediendo a los campos del registro seleccionado en el modelo.

No deberíamos escribir salida directamente desde un controlador, pero sirva este ejemplo de muestra del acceso a propiedades de una instancia del modelo.

```
publicfunctionmuestraId($id){
        $articulo=Articulo::find($id);
        echo$articulo->nombre_articulo;
        echo '<br>';
        echo$articulo->descripcion_articulo;
}
```

Modificar un registro dado

En el artículo anterior explicamos brevemente el patrón Active Record de arquitectura de software. Dijimos que proponía realizar operaciones para la persistencia de datos con

métodos como save() o delete(). Estos métodos nos servirán en Eloquent para poder almacenar información que haya dentro de un modelo.

Ponte el caso que queremos recuperar un artículo para modificar su título. Lo primero será hacerse con un objeto de la clase del modelo App\Articulo, que contenga el artículo que se desea modificar. Luego cambiaremos las propiedades del objeto tal cual, asignando nuevos valores a las propiedades deseadas. Por último invocaremos el método save() para que esas modificaciones se almacenen en la tabla asociada.

```
publicfunctionmuestraId($id){
        $articulo=Articulo::find($id);
        $articulo->nombre_articulo='Este es el nuevo título';
        $articulo->save();
}
```

Borrar un registro dado

De una manera muy similar borramos un registro de la tabla. Accediendo a la instancia del modelo, cargando el registro que se desea borrar, y luego invocando el método delete().

```
publicfunctionborraId($id){
        $articulo=Articulo::find($id);
        $articulo->delete();
}
```

Crear un nuevo registro

Para crear un nuevo registro en la tabla del modelo basta con crear una instancia del modelo, mediante su constructor. Luego asignamos todos los valores a ese nuevo modelo a través de sus propiedades y por último invocamos el método save() para que los datos se almacenen. Solo al hacer el save() el registro se guarda en la base de datos.

```
publicfunctioncrear(){
        $articulo=newArticulo;
        $articulo->nombre_articulo="Este es el título";
        $articulo->descripcion_articulo="Esta es la descripción";
        $articulo->save();
        echo'creado artículo con id: '.$articulo->id;
}
```

Como alternativa podríamos enviar todos los datos del nuevo artículo como un array asociativo a la función constructora del modelo. Sin embargo, por motivos de seguridad esta operación tiene sus limitaciones.

El código de la función en el controlador tendría una forma como esta:

```
publicfunctioncrear(){
        $articulo=newArticulo([
                        'nombre_articulo'=>'Este es el título',
                        'descripcion_articulo'=>'Esta es la descripción'
```

```
                ]);
        $articulo->save();
        echo'creado artículo con id: '.$articulo->id_articulo;
}
```

Sin embargo encontraremos que ejecutarlo tal cual nos devuelve el error "MassAssignmentException in Model.php". La solución pasa por especificar en el modelo qué campos de la tabla artículo queremos permitir que se seteen mediante esta alternativa con el constructor.

Este sería el código de mi modelo, una vez indicada la posibilidad de setear esos campos mediante el constructor del modelo.

```
namespaceApp;

useIlluminate\Database\Eloquent\Model;

classArticuloextendsModel
{

        protected$fillable=['nombre_articulo','descripcion_articulo'];
}
```

Conclusión

Has aprendido a usar los modelos para la realización de un completo conjunto de posibilidades. Seguro que apreciarás las ventajas y facilidades de trabajar contra el ORM Eloquent.

Todavía nos quedan cosas importantes que estudiar, como el tema de las relaciones entre tablas. En breve nos pondremos con ello, de momento te recomendamos tomarte el tiempo para practicar lo aprendido.