LaravelEloquent

Introducción a Eloquent, el ORM de Laravel que implementa el patrón Active Record para el trabajo con datos que llegan de bases de datos relacionales.

En este artículo y los siguientes vamos a abordar el nivel de acceso a bases de datos de más alto nivel, que sería Eloquent. Eloquent es lo que se conoce como "ORM" (ObjectRelationalMapping o Mapeo de Objeto Relacional). Básicamente es un sistema que nos permite llevar la capa de persistencia en bases de datos por medio de objetos y que nos ahorra el trabajo de comunicar directamente con la base de datos.

Qué es un ORM, qué es Active Record y cómo es Eloquent en Laravel

Con Eloquent, y en general con cualquier ORM tenemos la posibilidad de trabajar con los datos que hay en las bases de datos por medio de objetos. Los datos de las tablas se mapean a objetos, evitando todo el trabajo de escribir las consultas para el acceso a la información. El acceso a sus relaciones también se realiza por medio de propiedades de objetos, lo que facilita mucho las dinámicas de acceso a la información, siempre que haya una relación de dependencia declarada entre las distintas entidades de nuestro modelo de datos.

La forma de trabajo de Eloquent implementa el patrón "Active Record", un patrón de arquitectura de software que permite almacenar en bases de datos relacionales el contenido de los objetos que se tiene en memoria. Esto se hace por medio de métodos como save(), update() o delete(), provocando internamente la escritura en la base de datos, pero sin que nosotros tengamos que componer las propias sentencias.

En Active Record una tabla está directamente relacionada con una clase. En ese caso se dice que la clase es una envoltura de la tabla. La clase en si es lo que conocemos en el framework como "modelo". Cuando nosotros creamos un nuevo objeto de ese modelo y decidimos salvarlo, se produce la creación de un registro de la tabla. Cuando el objeto se modifica y se salvan los datos, se produce el correspondiente update en ese registro. Cuando ese objeto se borra, se produce el delete sobre ese registro de la tabla.

ORM sería entonces la herramienta de persistencia y Active Record el patrón de arquitectura que se sigue para su construcción. Eloquent es el nombre con el que se conoce en Laravel esta parte del framework, que una vez nos acostumbramos a usar, nos agiliza la mayoría de las operaciones habituales del acceso a bases de datos.

Nota: Para el trabajo con el ORM Eloquent necesitamos la misma configuración que ya explicamos en el artículo de Bases de datos con Laravel.

Crear un modelo Eloquent

Cualquier modelo creado en Laravel es un "EloquentModel", así que nos permitirá realizar las acciones de acceso y modificación de los datos típicas de las bases de datos relacionales.

En realidad muchas cosas sobre los modelos ya las comentamos anteriormente en el artículo de Introducción a los modelos en Laravel. Procuraremos no repetirnos demasiado, así que te recomendamos la lectura de ese artículo previamente.

Los modelos en Laravel se ubican sueltos, dentro de la carpeta "app", aunque si tenemos muchas tablas (y por tanto muchos modelos) sería adecuado realizar algún tipo de orden en esta carpeta. Lo adecuado entonces es que funcione con el modelo de carga de clases que nos ofrece el autoload de Composer, que básicamente usa los namespaces para definir la localización de subdirectorio donde se encuentra cada uno de los archivos de las clases.

El modo más adecuado para construir nuestros modelos es usando el asistente de Artisan, con la instrucción make:model seguida del nombre de nuestro modelo.

```
phpartisanmake:modelProduct
```

Nota: Recuerda que por convención en Laravel se usan los nombres de las tablas en plural y minúsculas, mientras que en las clases de los modelos debemos colocar mayúscula en la primera letra (camelcase en realidad) y acabado en singular. Por ejemplo, para una tabla llamada "Groups", el nombre del modelo será "Group". Luego veremos cómo modificar esa convención en el caso que sea necesario.

Nuestro modelo "Product" como se ha dicho, corresponde con una clase, que se colocará en la carpeta "app" y tendrá el siguiente código.

```
namespaceApp;

useIlluminate\Database\Eloquent\Model;

classProductextendsModel
{
 //
}
```

A pesar que está prácticamente vacío de código es importante señalar que los modelos de por sí ya hacen bastante cosas, gracias a que heredan el comportamiento a partir de la clase Illuminate\Database\Eloquent\Model.

Modificar el nombre de la tabla

Por convención en Laravel, si no se especifica nada, el modelo Product se correspondería, o trabajaría, con la tabla products. ¿Pero qué se hace si ya nos vienen dados los nombres de las tablas y no se ajustan a esta convención?

Si queremos modificar el nombre de la tabla con la que trabaja un modelo podemos indicarlo a partir de la definición de la propiedad $table, asignando el valor que tiene la tabla asociada con este modelo.

```
protected$table='My_categories';
```

Esa declaración de propiedad se coloca generalmente al principio del código de la clase.

Modificar la clave primaria

Las claves primarias en Laravel también tienen su convención. Básicamente toda tabla tiene una columna llamada "id" que será "autoincrement" y que hará las veces de clave primaria. De nuevo, si esto no se ajusta a nuestra realidad solo tenemos que definir una propiedad llamada $primaryKey, asignando el nombre del campo que actúe como "Primary Key".

```
protected$primaryKey='id_categoria';
```

```
Timestamps "created_at" y "updated_at"
```

Cuando hablamos de Migraciones en Laravel, se mencionaron muy rápidamente los campos "timestamps". Estos campos, como su nombre indica, sirven para llevar marcas de tiempo.

Nota: Al crear una tabla podemos indicarle que nos cree los correspondientes campos timestamp "created_at" y "updated_at", simplemente indicándolo con el método timestamps(). Recuerda que una creación de una tabla en el archivo de migraciones se definía más o menos así:

```
Schema::create('carpetas',function(Blueprint $table){
$table->increments('id');
$table->string('nombre');
$table->text('descripcion');
$table->timestamps();
});
```

Como resulta obvio, estos timestamps nos llevan la fecha de creación y actualización de un registro. Al nivel de un modelo y por convención son necesarios para que todo funcione correctamente, porque Eloquent los actualiza automáticamente. Si no existen esos campos, al realizar las operaciones de escritura sobre la tabla, nos arrojará un error diciendo que no ha encontrado tales columnas.

En el caso puntual, para los modelos donde no necesitemos que estos campos lleven la cuenta del tiempo, porque es algo que no vemos necesario en todas las tablas, podemos evitar este comportamiento automático simplemente definiendo la propiedad $timestamps al valor false.

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

Otra cosa que podemos modificar es el formato de la fecha almacenada en estos campos. Es algo que generalmente no necesitarás, pero se puede marcar a través de la propiedad $dateFormat. Consulta la documentación en este caso.

Modelo que sobreescribe las convenciones

Veamos el código de un modelo que sobrescribe las mencionadas convenciones:

```
<?php

namespaceApp;

useIlluminate\Database\Eloquent\Model;

classCategoryRoleextendsModel
{
protected$table='rol';
$primaryKey='id_rol';
public$timestamps=false;
}
```

Conclusión

Ya conoces las generalidades de los modelos en Laravel, usando Eloquent ORM. Has visto que no es necesario escribir mucho código y lo que no has visto todavía es cómo ya nos ofrecen la mayoría de las funcionalidades listas para usar. Esto es gracias a que en Eloquent se realizan las cosas más por convención que por configuración.

En el siguiente artículo estudiaremos cómo trabajar con un modelo desde un controlador para ejecución de las principales operaciones de acceso a los datos.