Relaciones de 1 a 1 en Laravel Eloquent

Cómo implementar relaciones de 1 a 1 en modelos de Eloquent, desde la creación de las migraciones, modificación de los modelos y su uso.

En este artículo nos vamos a introducir en una parte más práctica, abordando cómo debemos configurar relaciones en una aplicación Laravel, usando el ORM Eloquent.

En este artículo veremos el flujo completo para tratar una relación de 1 a 1 en Laravel, comenzando por la creación de una migración y posteriormente la definición de las relaciones dentro de los modelos, así como el uso de esos modelos. Aunque Laravel te ayuda mucho y Eloquent necesita pocas configuraciones, no es un tema trivial, así que trataremos de explicarlo detenidamente.

Relaciones de uno a uno

Las relaciones de uno a uno no son las más habituales, y de hecho muchas veces se evitan, pero tenemos ejemplos típicos que se repiten en diferentes aplicaciones: por ejemplo, un usuario tiene un perfil de usuario.

La relación de 1 a 1 se define en ese caso porque un usuario solo puede tener un perfil de usuario y porque un perfil de usuario solo puede pertenecer a un usuario.

Nota: Este tipo de relaciones a veces podrían suponer juntar ambas tablas en una. En el caso del usuario y su perfil podríamos tener ambos conjuntos de datos en la tabla users, pero podría haber diversos motivos por los cuales separar esta información en dos tablas. Por ejemplo motivos de almacenamiento, ya que no todos los usuarios pueden tener perfil definido y quizás no queramos meter en la tabla de usuario muchos campos que puede que no se rellenen la mayoría de las veces. Incluso así, juntar ambas tablas en una puede suponer mejora de rendimiento, cuando la información siempre se va a consultar al mismo tiempo.

En términos de programación no encontraremos muchas diferencias en lo que supone el trabajo de acceso a ambas tablas, gracias a Eloquent. Es decir, si definimos correctamente los modelos, poco nos importará que estas informaciones se encuentren en tablas separadas.

Migración a la hora de definir una relación de 1 a 1

En la definición de las tablas, programada en las correspondientes migraciones, debemos comenzar a definir las relaciones, ya que Laravel toma esa información para poder configurar los modelos.

En el ejemplo que nos ocupa tenemos dos tablas: users y profiles. La migración de la tabla users nos la dan hecha en Laravel, puesto que el propio framework ya implementa unos

mecanismos para poder registrar y loguear usuarios. Por tanto, solamente necesitaremos definir la migración de la tabla Profile.

En la migración colocaremos los campos de la tabla perfil, para almacenar todos los datos que necesitemos y la definición de la clave foránea.

```
Schema::create('profiles',function(Blueprint $table){
$table->integer('user_id')->unsigned();
$table->primary('user_id');
$table->text('bio');
$table->string('company',150);
$table->string('technologies',200);
$table->timestamps();
$table->foreign('user_id')->references('id')->on('users')-
>onDelete('cascade');
});
```

Es interesante apreciar que nuestra tabla tiene el campo "user_id". En la última línea de la definición de esta tabla, verás definida la clave foránea como referencia al campo "id" de la tabla "users". Además, para evitar que se puedan borrar usuarios y se queden "colgados" sus perfiles sin referencia, hemos definido un "on delete cascade". Eso provocará que los perfiles se borren automáticamente cuando se borre un usuario, algo muy práctico para ahorrarnos realizar esa operación doble de borrado y sobre todo importante para que debido a un olvido estemos borrando usuarios sin borrar sus perfiles.

Nota: He definido también como clave primaria user_id. Esto no sería absolutamente necesario, pero realmente desde el punto de vista de una clave primaria, user_id cumple con los requisitos, pues no se deben producir dos perfiles para un mismo usuario. Esto ayudará a respetar la integridad de los datos, evitando que se puedan insertar dos perfiles para el mismo usuario por error. Pero para ese mismo cometido podríamos también haber definido un índice único. Esto es más un criterio de diseño de base de datos que de Laravel, pues el framework no necesitaría que se definiese ese campo como clave primaria para que la relación de 1 a 1 se pueda configurar.

Declaración de las relaciones para la configuración de los modelos

En Laravel, la declaración de relaciones se realiza mediante una función, que se suele colocar en ambos modelos pertenecientes a ambas tablas relacionadas. No obstante, no es una necesidad realizar estas funciones, puesto que solo harán falta si realmente necesitas acceder desde un modelo a su información relacionada. Es decir, si entre mis operaciones de acceso a la información de un usuario necesito acceder al perfil, entonces configuraré la relación en el modelo del usuario. Pero si nunca voy a necesitar, dado un perfil, acceder a los datos del usuario al que pertenece, no necesitaría configurar esa relación en el modelo del perfil.

Todas las funciones responden a un patrón similar de declaración, aunque usaremos diversos métodos ayudantes dependiendo del tipo de relación. Y esto vale para todos los tipos de relaciones, de 1 a muchos, de muchos a muchos, etc. De momento nos centramos

en las relaciones de 1 a 1, pero mucho de lo que aprenderás ahora se puede aplicar de manera muy similar a otros tipos de relaciones. Lo verás más claro con un poco de código.

Definir la relación desde el usuario hacia el perfil

Nuestro modelo User debe informar que tiene un perfil asociado. Esta relación la querrás configurar siempre, porque realmente es muy necesario que dado un usuario puedas acceder a su perfil.

Es tan sencillo como agregar este método a la clase User, definida en app/user.php

```
publicfunctionprofile(){
return$this->hasOne('App\Profile');
}
```

Puedes ver el patrón que usaremos en esta y otros tipos de relaciones. Usamos un método con el nombre de aquel modelo que estamos relacionando. (function profile() para relacionar con el modelo Profile). Como código de la función colocamos un return sobre lo que te devuelve el método hasOne(), indicando a este método el modelo con el que queremos relacionar con su namespace.

Nota: En esta ocasión el tipo de relación está definida por el método hasOne(), pero en otros casos usaremos otros métodos como hasMany() o belongsTo().

Obviamente, para que esto funcione, aparte del modelo User, que te lo dan hecho, tendrás que crear el modelo Profile, que sería nuevo. En el artículo de crear modelos Eloquent está detallado el proceso.

También es importante mencionar que para que esta declaración de relación salga correctamente, deben darse varios patrones de nombrado de modelos, tablas, claves, etc. Eloquent asume que la clave foránea de la tabla que relacionamos debe corresponder con la clave primaria "id" de la tabla de la que partimos (en este caso clave foránea en la tabla profiles "user_id" debe corresponder con la el campo "id" de la tabla users, o aquel campo definido como $primaryKey). Si no es este el caso, podemos definir la relación indicando parámetros adicionales:

```
return$this-
>hasOne('App\Profile','clave_foranea','clave_local_a_relacionar');
```

Definir la relación inversa, del perfil hacia el usuario

Si lo consideras necesario, también podrías definir en el modelo Profile la relación hacia el usuario al que pertenece.

Nota: Esto lo hemos mencionado antes, pero insistimos en que la decisión de definir o no esta relación depende de tus necesidades. En concreto en este caso esa configuración no siempre será

necesaria, puesto que generalmente necesitarás acceder a un perfil de un usuario dado y no a un usuario de un perfil dado.

Esta relación de pertenencia (un perfil pertenece a un usuario) se define mediante el método belongsTo().

```
publicfunctionuser()
{
return$this->belongsTo('App\User');
}
```

El esquema de trabajo es muy similar. Usamos un método user, porque es el usuario el que se querrá relacionar al perfil. Luego usamos belongsTo() indicando el modelo con el que estamos relacionando. Del mismo modo, se deben dar una serie de patrones asumidos por Laravel en nombres de identificadores, claves, etc. Si no es así podemos invocar al método belongsTo() con parámetros adicionales. Esto lo hemos explicado para la relación anterior (usuario a perfil), pero te sugerimos leer la documentación para mayores informaciones.

Usar modelos con tablas relacionadas

Una vez definidos los modelos correctamente, para dar soporte a las relaciones deseadas, podremos acceder a los datos de las tablas relacionadas como si fueran datos del propio modelo. Es decir, una vez realizados los pasos descritos anteriormente, ya solo nos queda disfrutar de las facilidades que nos aporta el ORM.

De todos modos, como conclusión a este artículo vamos a presentar varios códigos que realizan operaciones típicas que querremos realizar con este tipo de relaciones.

Para empezar, si vas a hacer uso de ambos modelos, lo primero que tendrás que hacer es declarar los correspondientes "use" de sus espacios de nombres.

```
useApp\Profile;
useApp\User;
```

Nota: Recuerda que el namespace de los modelos en Laravel está en App, ya que los modelos se encuentran generalmente sueltos en la carpeta "app", pero nosotros podríamos decidir otro tipo de organización si lo consideramos así.

Crear un perfil y asociarlo a un usuario:
Esta operativa la realizas creando un perfil mediante el modelo y luego asignando ese perfil a un modelo de usuario. No te olvides de salvar el usuario luego.

```
$perfil=newProfile;
$perfil->bio='Esta es la biografía de un usuario';
$perfil->company='EscuelaIT';
$perfil->technologies='PHP, Javascript, Apache, HTML';
$user=User::find(1);
$user->profile()->save($perfil);
```

Borrar un usuario y su perfil:
Realmente, si hemos definido el "delete cascade" no necesitamos más que borrar el usuario y el perfil se borrará automáticamente.

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

Acceder al perfil de un usuario y comprobar su existencia:
El acceso a las tablas relacionadas, una vez definida la correspondiente relación en el modelo (hasOne / belongsTo), se hace como si fueran propiedades del propio modelo.

```
$usuario=User::find(2);
$perfil=$usuario->profile;
if($perfil){
echo'tengo perfil '.json_encode($perfil);
}else{
return'no tiene profile';
}
```

En esta operación cabe señalar dos detalles:

1. El dato del perfil es "lazy load", a no ser que especifiquemos lo contrario. Veremos más adelante cómo.
2. Una vez hemos accedido a la propiedad profile del usuario ($usuario->profile) en el objeto $usuario ya se encontrarán los datos del perfil. Por ello, las siguientes veces que se acceda a esa propiedad Laravel no necesitará irse de nuevo a la base de datos para buscar el perfil.

Desde un perfil, acceder al usuario al que pertenece:
Si hemos definido también la relación desde el perfil al usuario (belongsTo) podremos acceder desde un perfil al usuario al que le pertenece.

```
$perfil=Profile::find(1);
$user=$perfil->user;
```