Relaciones en los modelos Eloquent

Entender cómo un ORM en general gestiona las relaciones entre tablas y cómo Eloquent en particular nos permite acceder a información relacionada como si estuviera en el propio modelo.

En modelos Eloquent, el ORM de Laravel que venimos abordando en diversos artículos, hemos comprobado que prácticamente no hemos colocado código alguno para que toda la "magia" de acceso a los datos funcione. Es algo que realmente llama la atención, porque nos permite mucha agilidad en el desarrollo.

Pero en el área de relaciones entre tablas vamos a tener que ayudar de algún modo al framework para que realice su increíble trabajo. De todos modos, como verás en este artículo y los siguientes, se tratará solamente de realizar unas pocas declaraciones y con ello podremos conseguir nuestro objetivo de acceder a datos relacionados, sin tener que realizar las consultas de acceso a la información.

El objetivo es conseguir que nuestros modelos, aparte de los datos que contienen las propias tablas de la base de datos con las que están asociadas, permitan acceder a los datos de todas las tablas que se hayan relacionado en la definición de la base de datos. Su acceso se realizará por medio de propiedades, de modo que en la práctica podrá parecer que esos datos pertenecen al propio modelo, aunque realmente sean datos que están en otras entidades.

Relaciones soportadas por Eloquent

Como sabes, existen diversos tipos de relaciones en bases de datos, las básicas que todo el mundo debe conocer son:

- Relaciones de uno a uno: Por ejemplo, un usuario tiene un perfil de usuario. El usuario tiene un perfil y el perfil solo pertenece a un usuario.
- De uno a muchos (1 a n): Por ejemplo, un usuario tiene posts. El usuario puede cargar un número indeterminado de post, pero un post sólo pertenece a un usuario.
- De muchos a muchos (n a m): Por ejemplo, un artículo tiene varias etiquetas y una etiqueta puede tener varios artículos. Estas relaciones generan una tabla adicional generalmente que Eloquent maneja para ti.

A estas relaciones debemos agregarle otras que Eloquent soporta, no tan comunes pero que también se encuentran en la vida real.

- Relaciones de pertenencia a través de otras entidades: Estas relaciones no son más que las relaciones que conoces de toda la vida, pero saltando a través de otras tablas. En un esquema relacional la tabla "a" se relaciona con "b" y a su vez "b" se relaciona con "c". En un caso como este, con Eloquent podemos conseguir definir una relación directa desde "a" hacia "c", pasando a través de "b". Ese paso "a través" se realiza de manera

transparente para el desarrollador. Es decir, es como si la tabla "a" estuviera directamente vinculada a "c".

- Relaciones polimórficas: esto es algo muy interesante también y se basa en un esquema de relaciones que quizás hayas tenido que resolver en algunas ocasiones "a mano" y que Laravel te hace de manera automática. Básicamente consiste en una relación donde aquella entidad con la que estás relacionando pueda ser variable. Por ejemplo, un usuario puede dar el "me gusta" a post de otro usuario, a un comentario de otro usuario, a un vídeo, a un artículo en venta… en general, a un número de entidades indeterminado y variable. De ahí el polimorfismo.

Ten en cuenta que las relaciones en la base de datos se definen a la hora de hacer migraciones. Tenemos un artículo entero dedicado al tratamiento de índices en las migraciones en el que hablamos de los índices de claves foráneas. Nuestro primer paso entonces sería establecer correctamente esas relaciones en el modelo de la base de datos, puesto que Eloquent lo necesita para su correcto funcionamiento.

A partir de ahí, podemos declarar relaciones en los modelos. Esto lo conseguimos por medio de funciones en las que básicamente informamos que tal entidad pertenece a tal otra, o que a tal entidad se le asocian muchos ejemplares de tal otra. Dependiendo del tipo de relación esta declaración de función se realizará de una manera o de otra.

Ejemplo básico sobre las ventajas de un ORM

Antes de ponernos a ver los tipos de relaciones y cómo se declaran en Eloquent creo que puede estar interesante incidir de nuevo en cómo un ORM ayuda en el acceso a los datos que están relacionados.

Piensa en este sencillo ejemplo. Tenemos un comercio electrónico en el que manejamos productos. Los productos pueden tener muchos comentarios.

El ORM permite acceder a los productos y sus datos mediante objetos.

```
$producto1=Producto::find(1);
// Ahora el producto me permite acceder a sus datos
echo$producto1->nombre; //muestra el nombre
echo$producto1->descripcion; //muestra la descripción
```

Pero además, si definimos la relación en el modelo Eloquent, también podremos acceder a los datos de las tablas relacionadas. Acceder a los comentarios de un artículo es tan sencillo como acceder a un método. Esta sería una colección con todos sus comentarios:

```
$producto1->comentarios()
```

Nota: Además es posible que al hacer el primer acceso a los productos ya estés indicando que quieres recibir sus comentarios como una de las propiedades de ese producto. Esto se consigue con una funcionalidad que se llama "EagerLoading" que también explicaremos más adelante porque resulta de mucha utilidad.

Podré usar esa colección para recorrer los comentarios, mostrarlos en la página o hacer lo que sea necesario en cada caso. En ningún caso tendré que escribir una consulta o realizar algún tipo de operación para traerme esos datos. Simplemente declaro la relación en el modelo y esos datos estarán a mi disposición siempre que lo necesite.

Es interesante mencionar que el acceso a las tablas relacionadas se hace mediante un esquema de trabajo que se conoce como "Lazy Load" o "Carga Perezosa". Eso quiere decir que, cuando se generó el modelo con los datos del producto no se cargaron directamente los comentarios. Solo en el momento en el que se intente acceder a la propiedad donde se encuentran talas comentarios es cuando Eloquent hace el trabajo de acceder a la tabla relacionada y recuperar la información.

Obviamente, este esquema de lazy load es beneficioso porque no siempre que accedes a los datos de un producto necesitas realmente que te entreguen sus comentarios. Por ejemplo, al mostrar un producto en el carrito de la compra, no necesitas saber qué comentarios dieron los usuarios a ese producto. Pero a veces sí que es interesante que esos datos relacionados se entreguen directamente y estén disponibles y precargados. Esto lo veremos más adelante, pero en Laravel podrás indicar que una consulta ya te entregue determinada información de sus tablas relacionadas. En resumen, lazyloading es el comportamiento predeterminado, pero al recuperar un modelo podemos decirle explícitamente que cargue también de inicio información de tablas que están relacionadas, como se comentó en la nota anterior.

En los siguientes artículos vamos a ir abordando cada uno de los tipos de relaciones para ver cómo se declaran en un modelo.