Migraciones en Laravel

Trabajo con bases de datos y el sistema de migraciones para definir tablas y modificarlas a través del código de diferentes clases.

Las migraciones son como un control de versiones del estado de nuestras tablas. Están preparadas para ejecutarse en "ida y vuelta". Osea, podemos correr el sistema de migraciones para obtener la base de datos en su estado final, así como también podríamos volver hacia atrás, para recuperar cualquier estado de la base de datos hasta llegar al estado inicial, en el que se supone que no habría tabla alguna.

En el artículo anterior estuvimos haciendo una introducción a las migraciones en particular y al tema de las bases de datos en Laravel en general, por lo que no vamos a repetir lo mismo. Suponemos que el concepto se entiende, así como sus ventajas, y lo que queremos ahora es trasladarlo al mundo de lo concreto.

Las migraciones están dentro de la carpeta database/migrations y en la instalación de base de Laravel ya incorporan un par de archivos de migraciones, por lo que podemos observar cómo se han construido. Son un par de archivos para creación de la tabla de usuarios "users" y la tabla "password_resets", componentes del sistema de login de usuarios de Laravel.

En breve veremos cómo crear estos archivos de migraciones con nuestro propio código, pero antes vamos a aprender algunos comandos para operar con estas migraciones.

Comandos "artisan" para ejecutar las migraciones ya creadas

Ya que la instalación de Laravel viene con un par de migraciones, vamos a jugar un poco con ellas para empezar, ejecutándolas y volviendo hacia atrás.

El asistente de comandos artisan tiene una serie de instrucciones para trabajar con las migraciones. Como siempre, podemos encontrar la lista de comandos disponibles en artisan con "phpartisan".

En la sección "migrate" encontramos varios items que nos interesarán para trabajar con estas primeras migraciones que ya vienen creadas en el framework.

migrate
Este comando sirve para ejecutar todo el sistema de migraciones y poner en marcha cada una de las migraciones generadas en el proyecto. Crea el repositorio de migraciones y luego las ejecuta una por una para generar toda la estructura de la base de datos y los datos de prueba que puedan haberse cargado.

Nota: Estos comandos los ejecutamos desde la línea de comandos del servidor. Si estás usando Homestead, los ejecutas una vez conectado por ssh con la máquina virtual, desde la raíz de tu

proyecto. Como ya se ha mencionado en alguna ocasión los comandos artisan necesitan del intérprete de PHP, por lo que la sintaxis para invocarlos es la siguiente:

```
phpartisanmigrate
```

Prueba a ejecutar ese comando y verás como se generan tres tablas en tu sistema. Una de ellas llamada "migrations" sirve de control del sistema de migraciones. Luego verás otras dos tablas, que son las tablas de la aplicación para controlar a los usuarios y los reseteos de claves.

migrate:install
Sirve para crear el repositorio de migraciones. Lo que hace es crear una tabla en la base de datos que sirve para llevar el control de las migraciones realizadas y su orden. Su ejecución se realiza con el comando completo "phpartisanmigrate:install". Este comando va implícito en el comando anterior, migrate.

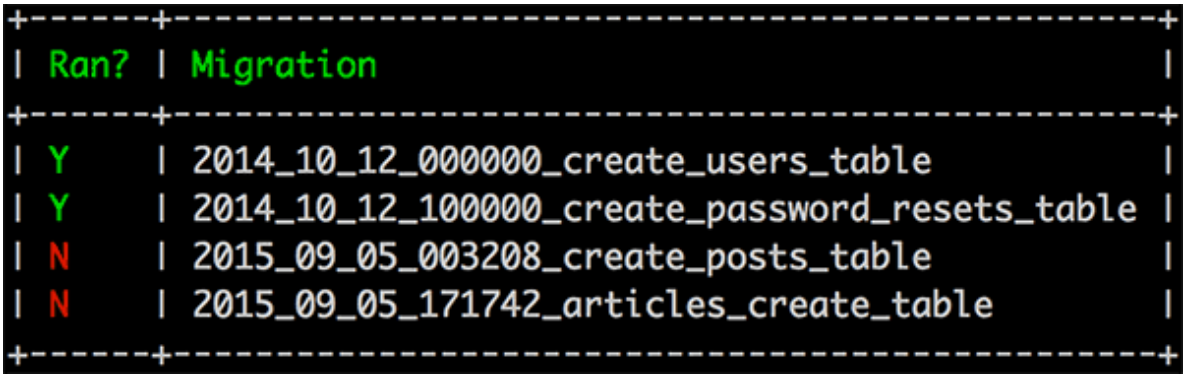
migrate:reset
Realiza una marcha atrás de todas las migraciones en el sistema, volviendo el schema de la base de datos al estado inicial.

migrate:refresh
Vuelve atrás el sistema de migraciones y las vuelve a ejecutar todas.

migrate:rollback
Vuelve hacia atrás la última migración realizada. Una migración puede incluir varias tablas que fueron migradas de una sola vez en el paso anterior.

migrate:status
Te informa de las migraciones presentes en el sistema y si están ejecutadas o no.

En la siguiente imagen puedes ver un reporte de status de migraciones, con algunas migraciones ejecutadas y otras no. No corresponde con las migraciones que tendrás ahora en tu sistema, ya que no hemos creado ninguna nueva aparte de las dos que venían originalmente en Laravel, pero dentro de poco vamos a crear esas migraciones que estás viendo en el status de la imagen:



Puedes probar estos comandos con total garantía que no vas a romper nada, ejecutando las migraciones y volviendo hacia atrás, consultando el "status" en cada paso. Eso te ayudará a entender mejor las mecánicas de las migraciones y a empezar a apreciar la potencia que hay detrás de este sistema.

Crear una nueva migración

Ahora vamos a crear nuestra primera migración, para agregar una nueva tabla en la base de datos. Como en otras ocasiones, el esqueleto de nuestra migración lo podemos crear fácilmente ayudados por un comando artisan: make:migration.

```
php artisan make:migrationcreate_posts_table
```

Habitualmente en el nombre de las migraciones indicas qué tipo de operación estás realizando al ejecutar esa migración. En este caso el nombre de la migración era "create_posts_table", osea, el "createtable" para la entidad "post".

Además, a este comando le podemos pasar un parámetro adicional para indicar a artisan el nombre de la tabla que está relacionada con la migración, que servirá para que lo incluya en el código esqueleto de la clase para esta migración.

```
php artisan make:migrationcreate_posts_table--table=posts
```

Ahora vamos a la carpeta donde se encuentras las migraciones: "database/migrations". Allí encontrarás el archivo de nuestra nueva migración. Apreciarás que todos los archivos de las migraciones tienen, en su nombre, un timestamp que ayudará a Laravel a saber en qué orden deben ejecutarse. En mi caso tiene este nombre de archivo:

```
2015_10_05_003208_create_posts_table.php
```

Obviamente, cuando generes tu migración se le asignará otro timestamp, acorde con el instante en el que se creó.

Puedes abrir el archivo ahora. El código te permitirá identificar rápidamente que todas las migraciones son hijas de la clase "Migration" (extendsMigration). Además encontrarás dos métodos ya generados que ahora te explicamos.

Métodos para procesar una migración o volver hacia atrás

El corazón de las migraciones, y aquello que tendrás que definir mediante código cuando las estás creando, son dos métodos que realizan la tarea de ejecutar la migración o volverla hacia atrás.

up()
Este método se ejecutará cuando la migración se corra, así que tiene todo el código necesario para hacer las modificaciones en el esquema de la base de datos. Por ejemplo, si estás creando una migración porque necesitas una nueva tabla, este método tendrá el código para crearla.

down()
Este método se ejecutará cuando se vuelva hacia atrás y se tengan que desechar los cambios producidos por esta migración. Tendrá el código necesario para volver hacia atrás y recuperar el estado anterior del schema de la base de datos. Si la migración se hiciera para crear una tabla, en este método pondrías el código para eliminarla.

Código para realizar cambios en la definición de la base de datos

En los métodos up() y down() colocas código PHP que se encargue de realizar los cambios pertinentes en la definición de la base de datos. Podrás hacer cosas como crear o eliminar tablas, cambiar su nombre, cambiar sus campos poniendo o quitando columnas, crear o eliminar índices, claves, etc. Sin embargo, para realizar todas estas operaciones no vas a trabajar con el lenguaje SQL del sistema gestor de la base de datos que hayas escogido, sino con un API de funciones de Laravel.

El API para hacer migraciones tiene la ventaja de ser común para cualquiera de los sistemas gestores de base de datos para los que Laravel es compatible, por lo que el código de migraciones será perfectamente válido para cualquier base de datos. Incluso si a mitad de proyecto se cambia el sistema gestor, tus migraciones seguirán sirviendo.

En este artículo vamos a hacer un ejemplo de código para crear una tabla, y su correspondiente "rollback". En futuros artículos revisaremos otros tipos de operaciones de modificación del schema, pero de momento puedes verlos en la propia documentación de Laravel "WritingMigrations".

Si queremos crear la migración para incorporar la tabla "posts", comenzaremos por definir el código para crear dicha tabla en el método up(). Las operaciones de alteración del schema de la base de datos se hacen a través de la fachada "Schema", por lo que básicamente dentro de este método realizaremos uso de esa fachada.

Para crear una tabla usamos el método estático create() de la fachada Schema, cuya invocación tendrá esta forma:

```
Schema::create('posts',function(Blueprint $table){
```

```
// código para definir esta tabla
});
```

Como se ve, el método requiere el nombre de la tabla que se va a crear y luego una función anónima (closure) que contiene el código para definir los campos de la tabla. Esta función recibe un parámetro de la clase Blueprint que tiene los métodos que necesitarás para definir la tabla.

Nota: Ese objeto de clase Blueprint nos lo inyecta Laravel en el closure, no tenemos que hacer nada para enviarlo a la función.

Como código para definir la tabla, ayudados por el objeto $table, de la clase Blueprint, realizaremos la declaración de cada uno de los campos, usando métodos como estos:

- increments() genera un campo auto-increment.
- string() genera un campo de tipo cadena (VARCHAR).
- text() genera un campo largo de cadena (TEXT).
- integer() genera un campo de tipo entero.
- boolean() genera un campo boleano.
- timestamp() genera un campo timestamp.
- timestamps(), observa que está en plural, genera las columnas created_at y updated_at para que Laravel lleve el control de estos campos automáticamente.
- …

Tienes decenas de tipos de columnas. Insistimos que los tipos de columnas no dependen del sistema gestor, sino que forman parte del API de migraciones. Por tanto, Laravel será encargado de definir el campo con el tipo nativo de la base de datos usada que mejor considere. Tienes la lista completa de tipos de columnas en la documentación de Laravel.

El código de ejemplo para la creación de la tabla de usuarios sería el siguiente:

```
publicfunctionup()
{
Schema::create('posts',function(Blueprint $table){
$table->increments('id');
$table->string('titulo',100);
$table->text('cuerpo');
$table->integer('visitas');
});
}
```

Hemos indicado una serie de campos variados. Observa que no hemos dicho por ningún lado que el campo "id" debe de ser considerado como índice o clave primaria, dado que Laravel ya lo sobreentiende, tal como comentamos en el artículo anterior.

Ahora vamos a implementar el método down(), que debe hacer el paso contrario que up(), en este caso eliminar la tabla "posts". Es muy sencillo:

```
publicfunctiondown()
```

```
{
Schema::drop('posts');
}
```

En este caso la operación de borrado de una tabla, método drop() de la facade "Schema", solamente requiere el nombre de la tabla que se debe eliminar.

Ahora podrás realizar los comandos para migrar y volver atrás en esta migración, que por si no te acuerdas son:

Ejecutar las migraciones pendientes, que como acabamos de crear una, te la pondrá en marcha. "phpartisanmigrate" Volver atrás, deshaciendo la última migración. "phpartisanmigrate:rollback"

En cada paso detente a observar la base de datos y podrás ver cómo se genera y se destruye la tabla al trabajar con el sistema de migraciones.

De momento es todo. Con lo que hemos visto podrás practicar varias cosas con el sistema de migraciones. En futuros artículos haremos nuevos ejemplos para abarcar más posibilidades.