APLICACIÓN CON BD

Estructura de la carpeta “database” en Laravel

En la carpeta database de Laravel tenemos las siguientes carpetas:

- **factories**: Para definir la estructura de los datos que grabaremos para nuestro desarrollo y nuestras pruebas
- **migrations**: Para definir la estructura de las tablas de nuestra base de datos
- **seeders**: Con el archivo DatabaseSeeder.php, que nos permitirá crear los datos en la base de datos

Para nuestra aplicación crearemos en phpMyAdmin una nueva base de datos que llamaremos: “aplicacion2BD”. Modifico el .env para configurar el acceso a la misma.

.env

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=aplicacion2BD
DB_USERNAME=root
DB_PASSWORD=
```

Definición de la aplicación

Se va a realizar una aplicación en Laravel, para gestionar un foro de preguntas.

- Las preguntas pertenecen a una categoría
- Pueden hacerse comentarios a una pregunta
- Existen etiquetas para las preguntas. Una pregunta podrá tener varias etiquetas.

Modelo de datos

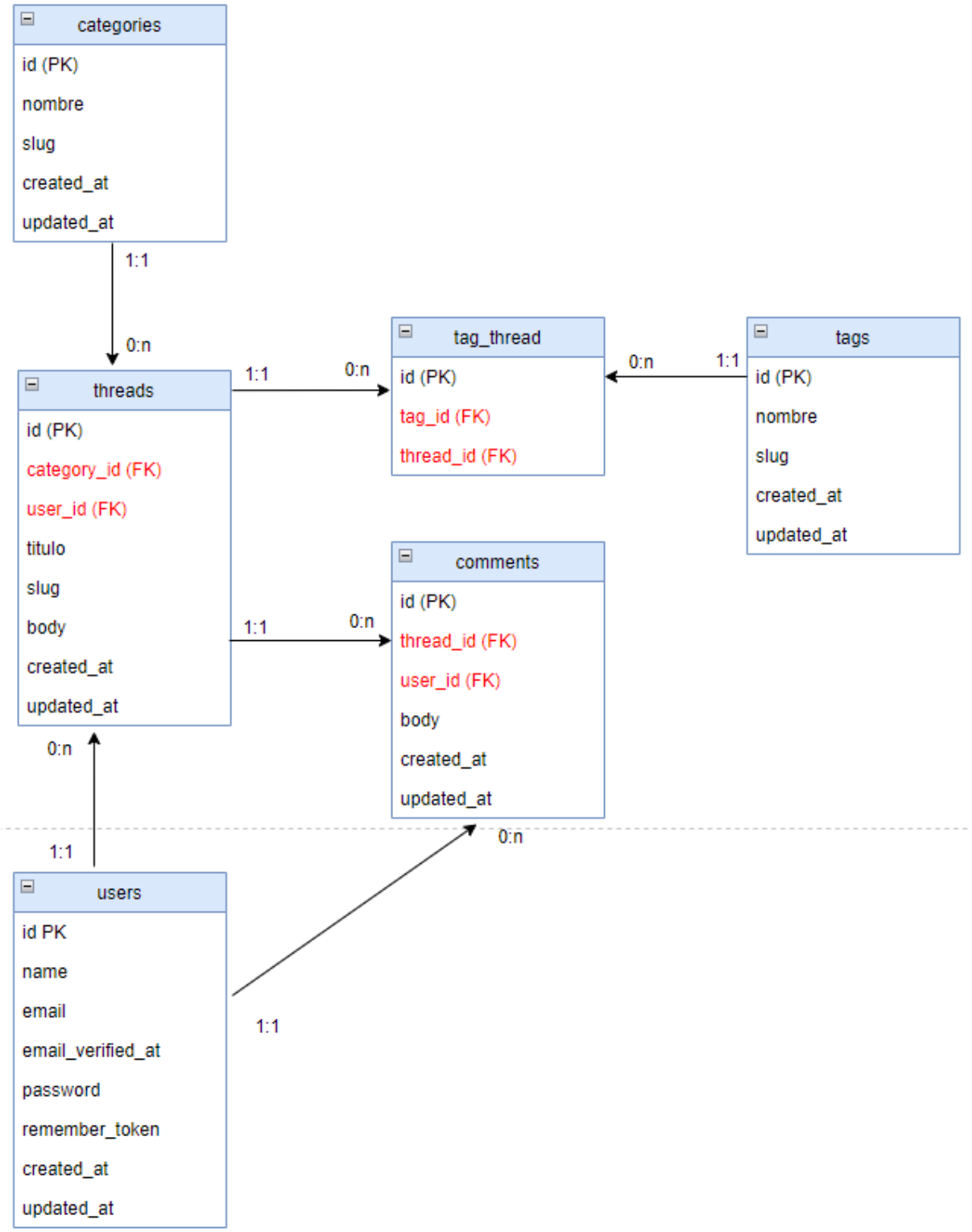
De las especificaciones anteriores, extraemos las **entidades**: categoría (category), pregunta (thread), comentario (comment) y etiqueta (tag).

Identificamos las **relaciones**:

- Las preguntas pertenecen a una categoría (1 a n)
- Los comentarios pertenecen a una pregunta (1 a n)
- Una pregunta puede tener varias etiquetas. Las etiquetas pueden utilzarse en varias preguntas (n a n)

Normalizamos este modelo para obtener un modelo físico de datos.

Modelo físico de la Base de Datos



En este modelo hemos incluido la tabla de usuarios, que viene por defecto en todas las aplicaciones de Laravel. Así podremos asociar los comentarios y las preguntas con los usuarios de nuestra apliclación.

Estructura inicial de la base de datos

Al crear el modelo de una tabla con artisan (make:model), vamos a utilizar las siguientes opciones:

- -m Generará la migración de la tabla. Ya lo hemos utilizado anteriormente.
- -f Generará la factoría de datos de la tabla

Utilizamos artisan para crear las tablas siguientes:

- categories: Tabla con categorías que permiten clasificar las preguntas
- threads: Es la tabla con los hilos de las preguntas.
- comments: Tabla de comentarios. Las preguntas tienen comentarios.
- tags: Etiquetas. Se asignan a las preguntas.

Generación de los modelos con artisan:

Creamos el modelo de las tablas, generando de forma automática la factoría y la migración
Lanzamos los siguientes comandos artisan, para crear:

- Crear el modelo (make:model)
- Crear la migración: Opción -m
- Crear la factoría: Opción -f

$ php artisan make:model Category -fm

```
alumno@alumno-VirtualBox:~/proyectos/aplicacionBD$ php artisan make:model Category -fm

  INFO  Model [app/Models/Category.php] created successfully.

  INFO  Factory [database/factories/CategoryFactory.php] created successfully.

  INFO  Migration [database/migrations/2023_01_14_163743_create_categories_table.php] created successfully.
```

$ php artisan make:model Thread -fm

```
alumno@alumno-VirtualBox:~/proyectos/aplicacionBD$ php artisan make:model Thread -fm

  INFO  Model [app/Models/Thread.php] created successfully.

  INFO  Factory [database/factories/ThreadFactory.php] created successfully.

  INFO  Migration [database/migrations/2023_01_14_164745_create_threads_table.php] created successfully.
```

$ php artisan make:model Comment -fm

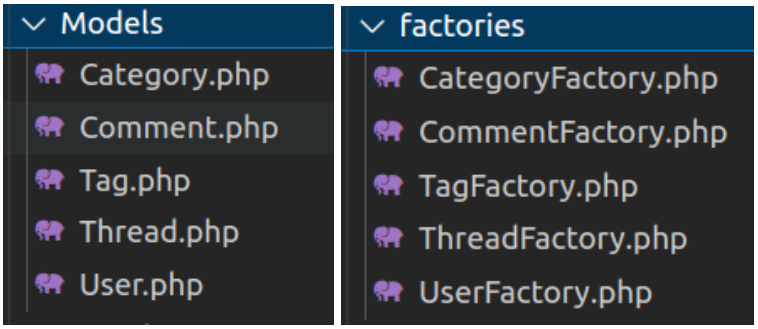
```
alumno@alumno-VirtualBox:~/proyectos/aplicacionBD$ php artisan make:model Comment -fm

  INFO  Model [app/Models/Comment.php] created successfully.

  INFO  Factory [database/factories/CommentFactory.php] created successfully.

  INFO  Migration [database/migrations/2023_01_14_165014_create_comments_table.php] created successfully.
```

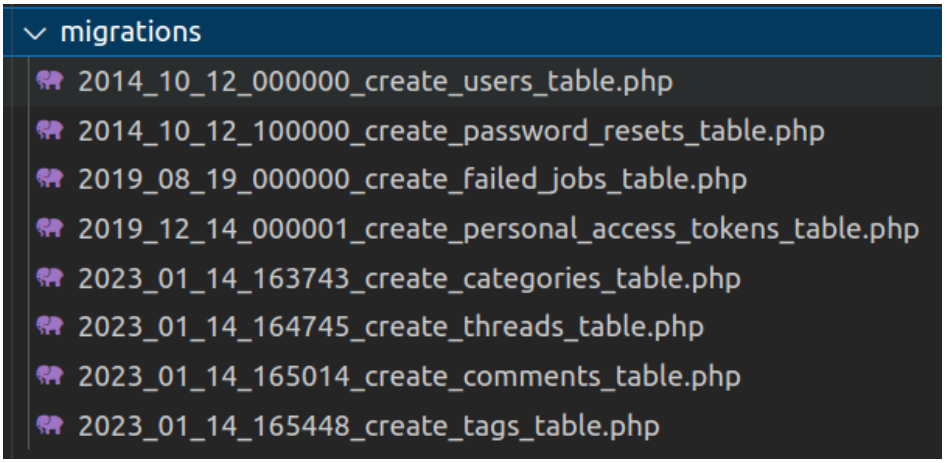
$ php artisan make:model Tag -fm

```
alumno@alumno-VirtualBox:~/proyectos/aplicacionBD$ php artisan make:model Tag -fm

  INFO  Model [app/Models/Tag.php] created successfully.

  INFO  Factory [database/factories/TagFactory.php] created successfully.

  INFO  Migration [database/migrations/2023_01_14_165448_create_tags_table.php] created successfully.
```

Estructura creada en la aplicación:

Una vez ejecutados los comandos, se han generado los modelos, las factorías y las migraciones de las tablas indicadas en nuestra aplicación Laravel:





El orden en que se crean las migraciones, es el orden en que se van a crear las tablas de la base de datos cuando ejecutemos las migraciones.

El orden es importante, sobre todo si tenemos relaciones jerárquicas, como es el caso de las categorías, que tienen preguntas, que a su vez tienen comentarios.

Configuración de la base de datos:

Migraciones

Modificaremos las migraciones generadas para reflejar la estructura física de nuestras tablas, de forma que podamos generarlas automáticamente con el comando migrate de artisan.

Migración de la tabla de categorías:

2023_01_14_163743_create_categories_table.php

```
        Schema::create('categories', function (Blueprint $table) {
            $table->id();

            $table->string('nombre');
            $table->string('slug')->unique();

            $table->timestamps();
        });
```

La migración tenía por defecto un id (clave primaria autogenerada) y los timestamps (fechas de creación y de modificación del registro).

Hemos añadido a la tabla de categorías dos campos:

1. **nombre**: El nombre de la categoría
2. **slug**:
    - El campo slug**,** en una tabla, se utiliza para tener una identificación única más humanizada que el id. Por ejemplo, la categoría 3 puede ser “coche”.
    - Lo utilizaremos cuando vayamos a acceder al registro por un identificativo textual.
    - La ventaja de utilizar un campo slug, es que si creamos una ruta con una variable que sea la categoría, queda mejor visualizar /categoria/coche, que no poner /categoria/3.
    - Para poder utilizarlo de esa forma, tendrá que ser único, por eso añadimos unique.

Migración de la tabla de preguntas (threads):

2023_01_14_164745_create_threads_table.php

```
        Schema::create('threads', function (Blueprint $table) {
            $table->id();

            $table->string('titulo');
            $table->string('slug')->unique();
            $table->text('body');

            $table->timestamps();
        });
```

Además del id y del timestamps, en la tabla de preguntas añadiremos los campos:

- **titulo**: Un título de la pregunta
- **slug**: Un identificador único en forma de texto
- **body**: El contenido o cuerpo de la pregunta. En este caso es un texto largo (text)

Migración de la tabla de comentarios:

2023_01_14_165014_create_comments_table.php

```
        Schema::create('comments', function (Blueprint $table) {
            $table->id();

            $table->text('body');

            $table->timestamps();
        });
```

En la tabla de comentarios sólo añado el campo para comentarios: "**body**".

Migración de la tabla de etiquetas:

2023_01_14_165448_create_tags_table.php

```
        Schema::create('tags', function (Blueprint $table) {
            $table->id();

            $table->string('nombre');
            $table->string('slug')->unique();

            $table->timestamps();
        });
```

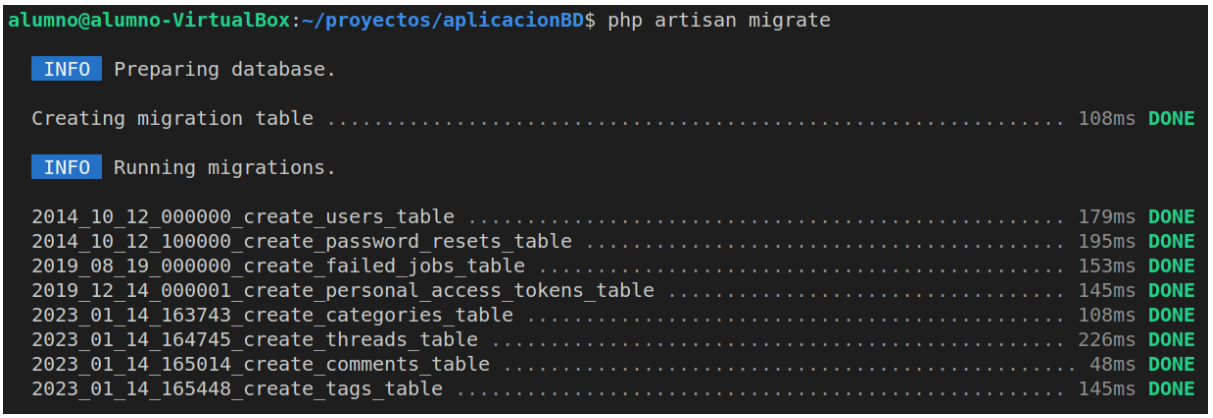
En la tabla de etiquetas, añadimos **nombre** y **slug**.

Creación de las tablas en la base de datos

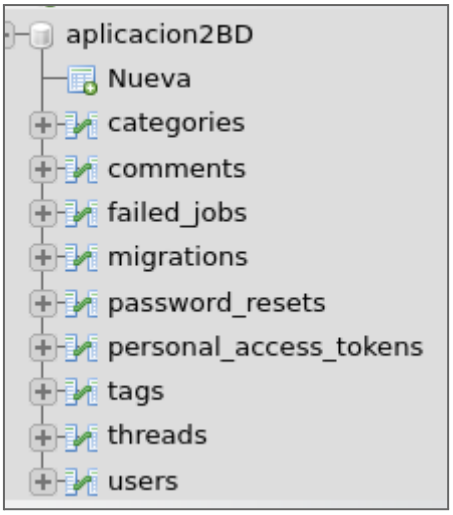
Para crear las tablas en la base de datos, podemos hacerlo mediante artisan:

```
$ php artisan migrate
```

El comando ejecutará las migraciones en el orden indicado.



Si vamos a phpMyAdmin veremos las tablas creadas:



Creación de las relaciones 1 a n

Lo primero implementaremos las claves foráneas en nuestras tablas, para asegurarnos de mantener la integridad referencial.

Paso1: Relación categoría de una pregunta

Para empezar, incluiremos el código de categoría en la tabla de preguntas, para indicar a qué categoría pertenece la pregunta:

2023_01_14_164745_create_threads_table.php

```
        Schema::create('threads', function (Blueprint $table) {
            $table->id();

            $table->unsignedBigInteger('category_id');
            $table->foreign('category_id')->references('id')->on('categories')
                ->onUpdate('cascade')
                ->onDelete('cascade');

            $table->string('titulo');
            $table->string('slug')->unique();
            $table->text('body');

            $table->timestamps();
        });
```

Hemos modificado la migración de la tabla threads (preguntas), para:

1. Añadir el campo category_id.
2. Indicar que es una clave foránea que referencia al campo id de la tabla categories
3. Indicar la opción de que las actualizaciones sean en cascada. Si se modifica el id de una categoría, ese id será modificado automáticamente en todas las preguntas de esa categoría, para hacer que coincida con el nuevo id.

4. Indicar la opción de que el borrado sea en cascada. Si se elimina una categoría, todas las preguntas de esa categoría serán eliminadas automáticamente.

De esta forma, aseguramos la integridad referencial. Si se elimina una categoría de la base de datos, las preguntas con dicha categoría serán eliminadas automáticamente.

Paso 2: Relación usuario de una pregunta

Aquí añadiremos el campo user_id, y la integridad referencial, en la tabla de preguntas.

2023_01_14_164745_create_threads_table.php

```
            $table->unsignedBigInteger('user_id');
            $table->foreign('user_id')->references('id')->on('users')
                ->onUpdate('cascade')
                ->onDelete('cascade');
```

Las líneas anteriores se añadirán en la migración, debajo de la relación con categorías, y funcionarán igual con respecto a la relación con la tabla de usuarios.

Paso 3: Relación pregunta de un comentario

A continuación añadiremos a la tabla comentarios el campo thread_id, para indicar a qué pregunta pertenece un comentario.

2023_01_14_165014_create_comments_table.php

```
        Schema::create('comments', function (Blueprint $table) {
            $table->id();

            $table->unsignedBigInteger('thread_id');
            $table->foreign('thread_id')->references('id')->on('threads')
                ->onUpdate('cascade')
                ->onDelete('cascade');

            $table->text('body');

            $table->timestamps();
        });
```

Con estos cambios (en morado), podemos ver cómo se añade el campo thread_id (código de la pregunta) y cómo se implementa la integridad referencial.

Paso 4: Ejecución de las migraciones en BD

Para actualizar nuestras tablas, de forma que incluyan las relaciones añadidas, ejecutaremos:

```
$ php artisan migrate:refresh
```

Este comando eliminará todas las tablas creadas, y las creará de nuevo con los cambios añadidos:

```
alumno@alumno-VirtualBox:~/proyectos/aplicacionBD$ php artisan migrate:refresh

  INFO  Rolling back migrations.

  2023_01_14_165448_create_tags_table ..................................... 59ms DONE
  2023_01_14_165014_create_comments_table ................................. 34ms DONE
  2023_01_14_164745_create_threads_table .................................. 57ms DONE
  2023_01_14_163743_create_categories_table ............................... 23ms DONE
  2019_12_14_000001_create_personal_access_tokens_table ................... 26ms DONE
  2019_08_19_000000_create_failed_jobs_table .............................. 25ms DONE
  2014_10_12_100000_create_password_resets_table .......................... 34ms DONE
  2014_10_12_000000_create_users_table .................................... 31ms DONE

  INFO  Running migrations.

  2014_10_12_000000_create_users_table .................................... 90ms DONE
  2014_10_12_100000_create_password_resets_table ..................... 138ms DONE
  2019_08_19_000000_create_failed_jobs_table ......................... 103ms DONE
  2019_12_14_000001_create_personal_access_tokens_table .............. 124ms DONE
  2023_01_14_163743_create_categories_table ........................... 72ms DONE
  2023_01_14_164745_create_threads_table ........................... 654ms DONE
  2023_01_14_165014_create_comments_table .......................... 176ms DONE
  2023_01_14_165448_create_tags_table ..............................  69ms DONE
```

Si vamos a phpMyAdmin, y comprobamos la estructura de la tabla threads, podremos ver que se han añadido tanto los campos como la integridad referencial:

Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra
id	bigint(20)		UNSIGNED	No	*Ninguna*		AUTO_INCREMENT
category_id	bigint(20)		UNSIGNED	No	*Ninguna*		
user_id	bigint(20)		UNSIGNED	No	*Ninguna*		
titulo	varchar(255)	utf8mb4_unicode_ci		No	*Ninguna*		
slug	varchar(255)	utf8mb4_unicode_ci		No	*Ninguna*		
body	text	utf8mb4_unicode_ci		No	*Ninguna*		
created_at	timestamp			Sí	*NULL*		
updated_at	timestamp			Sí	*NULL*		

Índices

Acción	Nombre de la clave	Tipo	Único	Empaquetado	Columna	Cardinalidad	Cotejamiento	Nulo
Editar Renombrar Eliminar	**PRIMARY**	BTREE	Sí	No	id	0	A	No
Editar Renombrar Eliminar	**threads_slug_unique**	BTREE	Sí	No	slug	0	A	No
Editar Renombrar Eliminar	**threads_category_id_foreign**	BTREE	No	No	category_id	0	A	No
Editar Renombrar Eliminar	**threads_user_id_foreign**	BTREE	No	No	user_id	0	A	No

Igualmente lo hace con la tabla de comentarios.

Creación de las relaciones n a n (muchos a muchos)

Para implementar una relación de este tipo, es necesaria una tabla adicional, de relación, también llamada pivot (pivote). En nuestro modelo, sólo tenemos una relación de ese tipo: Entre etiquetas (tags) y preguntas (threads).

Paso 1: Creación de la tabla de relación (pivot) entre preguntas y tags

- La convención para los nombres de las tablas de relación en Laravel, es el orden alfabético. Como tag va alfabéticamente antes que thread, el nombre de la tabla será tag_thread, con los nombres en *singular*.
- En cuanto al nombre de la migración, la convención de nombres es la siguiente: **create_nombreDeLaTabla_table**
- Laravel mantiene el nombre de la tabla en singular.

Esta tabla va a ser manipulada automáticamente por Laravel, por lo que no necesitaré crear un modelo, ni tampoco una factoría.

A continuación, creo la migración con artisan:

```
$ php artisan make:migration create_tag_thread_table
```

```
alumno@alumno-VirtualBox:~/proyectos/aplicacionBD$ php artisan make:migration create_tag_thread_table

  INFO  Migration [database/migrations/2023_01_15_084108_create_tag_thread_table.php] created successfully.
```

Paso 2: Configuración de la migración

En esta tabla no me interesa mantener los timestamps (fechas de creación y actualización de los registros). Por tanto, eliminaré timestamps.

2023_01_15_084108_create_tag_thread_table.php

```
Schema::create('tag_thread', function (Blueprint $table) {
    $table->id();

    $table->unsignedBigInteger('tag_id');
    $table->foreign('tag_id')->references('id')->on('tags')
        ->onUpdate('cascade')
        ->onDelete('cascade');

                    $table->unsignedBigInteger('thread_id');
    $table->foreign('thread_id')->references('id')->on('threads')
        ->onUpdate('cascade')
        ->onDelete('cascade');

    //$table->timestamps();
});
```

Añadimos a la tabla los campos de relación, tag_id y thread_id, que son la referencia a las tablas tags y threads. También los configuramos como foreign key para añadir la integridad referencial, con cascada automática de actualización y borrado.

Paso 3: Refrescamos las migraciones

Para que coja los campos, refrescamos las migraciones.

```
$ php artisan migrate:refresh
```

```
alumno@alumno-VirtualBox:~/proyectos/aplicacionBD$ php artisan migrate:refresh

  INFO  Rolling back migrations.

  2023_01_14_165448_create_tags_table ..................................... 32ms DONE
  2023_01_14_165014_create_comments_table ................................. 27ms DONE
  2023_01_14_164745_create_threads_table .................................. 21ms DONE
  2023_01_14_163743_create_categories_table ............................... 34ms DONE
  2019_12_14_000001_create_personal_access_tokens_table ................... 37ms DONE
  2019_08_19_000000_create_failed_jobs_table .............................. 22ms DONE
  2014_10_12_100000_create_password_resets_table .......................... 28ms DONE
  2014_10_12_000000_create_users_table .................................... 24ms DONE

  INFO  Running migrations.

  2014_10_12_000000_create_users_table .................................... 73ms DONE
  2014_10_12_100000_create_password_resets_table ..................... 166ms DONE
  2019_08_19_000000_create_failed_jobs_table ......................... 181ms DONE
  2019_12_14_000001_create_personal_access_tokens_table .............. 149ms DONE
  2023_01_14_163743_create_categories_table ........................... 78ms DONE
  2023_01_14_164745_create_threads_table ............................. 370ms DONE
  2023_01_14_165014_create_comments_table ............................ 179ms DONE
  2023_01_14_165448_create_tags_table ................................ 118ms DONE
  2023_01_15_084108_create_tag_thread_table .......................... 344ms DONE
```

Como podemos observar, se ha creado una tabla adicional tag_thread.

Si vamos a phpMyAdmin, podemos ver dicha tabla, y la integridad referencial

Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra
id	bigint(20)		UNSIGNED	No	*Ninguna*		AUTO_INCREMENT
tag_id	bigint(20)		UNSIGNED	No	*Ninguna*		
thread_id	bigint(20)		UNSIGNED	No	*Ninguna*		

Nombre de la clave	Tipo	Único	Empaquetado	Columna	Cardinalidad	Cotejamiento	Nulo
PRIMARY	BTREE	Sí	No	id	0	A	No
tag_thread_tag_id_foreign	BTREE	No	No	tag_id	0	A	No
tag_thread_thread_id_foreign	BTREE	No	No	thread_id	0	A	No