# 一對一關係

    public function up()

    {

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

            $table->increments('id');

            $table->integer('user\_id')->unsigned();

            $table->string('title');

            $table->text('content');

            $table->timestamps();

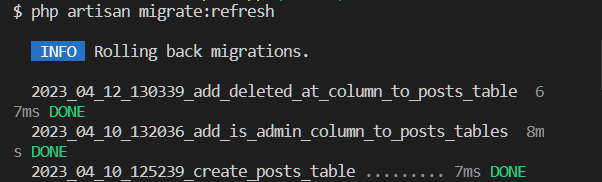
        });

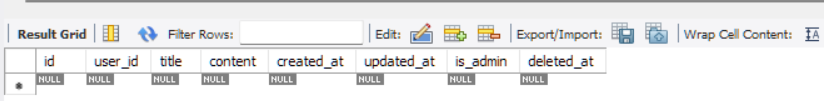
    }

回到設置posts的migration檔案，修改上述的內容 (p.s. posts沒有資料的情況下)

設置完畢後執行

$ php artisan migrate:refresh => 單純使用migrate會沒有反應，用refresh會將全部檔案drop掉再migrate





此時MySQL的posts表格會顯示如上

Inverse Relation

當資料庫和表格內容設置好後，回到User的Model設置function

    public function post() {

        return $this->hasOne('App\Models\Post');

    }

hasOne()

此處的$this指的是User，因此這行讀起來的意思是User has one post

// One to one relationship

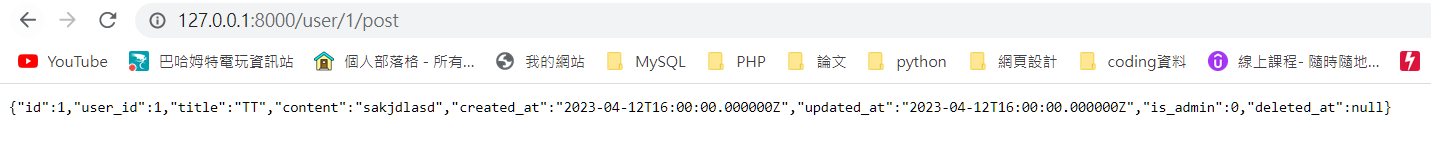
Route::get('/user/{id}/post', function($id) {

    return User::find($id)->post;

});

再到web.php中設置路徑

Eloquent 會假設對應的關聯的外鍵名稱是基於模型名稱。在這個例子裡，它會自動假設 post模型擁有 user\_id 外鍵。



    public function user() {

        return $this->belongsTo('App\Models\User');

    }

belongsTo()

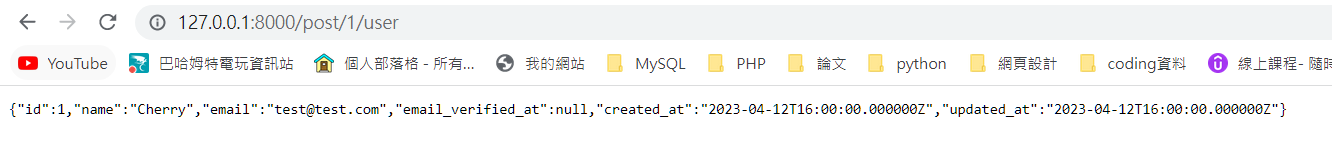
這邊的關係是 $this (這篇post) 屬於 user的

Route::get('/post/{id}/user', function($id) {

    return Post::find($id)->user;

});

再設置路徑去查看



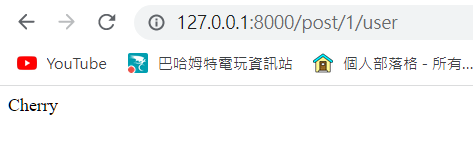
Route::get('/post/{id}/user', function($id) {

    return Post::find($id)->user->name;

});

或是只查看user的名字

return的指令會尋找posts的data中user\_id = $id的資料



# 一對多關係

// One to many relationship

Route::get('/posts', function() {

    $user = User::find(1);

    foreach($user->posts as $post) {

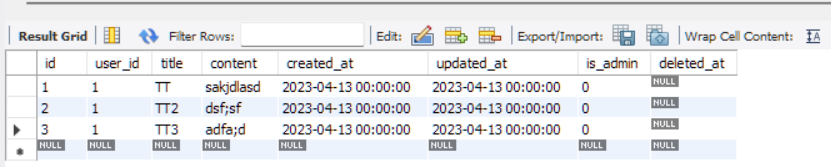
        echo $post->title."<br>";

    }

});

先定義一對多的路徑

並且到MySQL中添增user\_id=1的文章



    public function posts() {

        return $this->hasMany('\App\Models\Post');

    }

接著到User的Model中定義此功能，此user擁有很多個不同的post

# 多對多關係

$ php artisan make:model Role –m => 建立Role的Model

$ php artisan make:migration create\_users\_role\_table --create=role\_user => 建立role\_user的table的遷移檔案

    public function up()

    {

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

            $table->id();

            $table->string('name');

            $table->timestamps();

        });

    }

到create\_role\_table的遷移檔案進行設定

    public function up()

    {

        Schema::create('role\_user', function (Blueprint $table) {

            $table->id();

            $table->integer('user\_id');

            $table->integer('role\_id');

            $table->timestamps();

        });

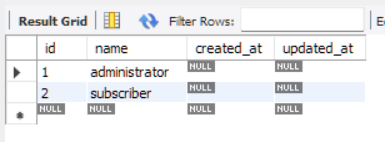
    }

再到create\_users\_role\_table的遷移檔案進行設定

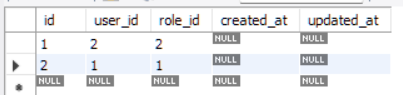
建立完畢後執行

$ php artisan migrate

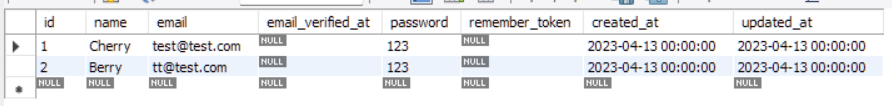
MySQL就會出現roles和role\_user的表格



建立roles的資料



建立role\_user的資料



users也多建立一筆資料

    public function roles() {

        return $this->belongsToMany('\App\Models\Role');

    }

到User的Model設置ManyToMany的關聯功能，users同時可以有多個roles

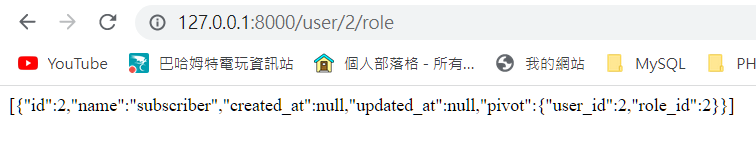
// Many to many relationship

Route::get('/user/{id}/role', function($id) {

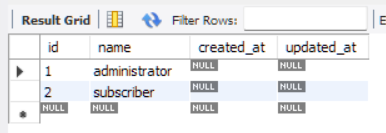
    $user = User::find($id);

    echo $user->roles;

});



pivot => 樞紐欄位



原先roles的欄位只有id, name, created\_at, updated\_at這四個欄位，卻多出了pivot欄位去連結roles和users兩個表格

// Many to many relationship

Route::get('/user/{id}/role', function($id) {

    $user = User::find($id);

    foreach($user->roles as $role) {

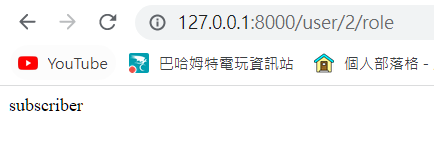
        echo $role->name;

    }

});

使用$user尋找user\_id = $id的資料，並且產生pivot table(關聯表) => 先前創立的belongsToMany()

最後回傳user的role



// Many to many relationship

Route::get('/user/{id}/role', function($id) {

    $user = User::find($id)->roles()->orderBy('id', 'desc')->get();

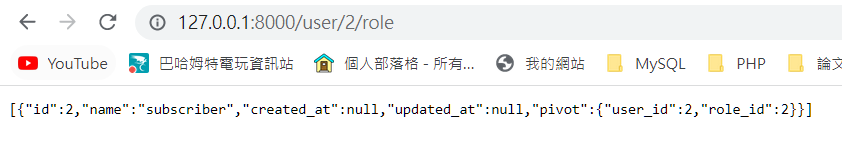
    return $user;

    // foreach($user->roles as $role) {

    //     echo $role->name;

    // }

});



# 查找關聯表(Intermediate table)

    public function roles() {

        return $this->belongsToMany('\App\Models\Role');

        // customized table name and column follow the format

        return $this->belongsToMany('\App\Models\Role', 'table\_name', 'fk\_id1', 'fk\_id2');

    }

}

底下的return是自定義的欄位名稱/關聯模型1的外鍵名稱/關聯模型2的外鍵名稱

Inverse relation

先前在User的模型中有定義user-role的資料型態(Many-to-Many)

    public function users() {

        return $this->belongsToMany('\App\Models\User');

    }

因此到Role的模型中定義role-user的資料型態(Many-to-Many)

再回到routes中

// Accessing the intermeidate table / pivot

Route::get('/user/pivot', function() {

    $user = User::find(1);

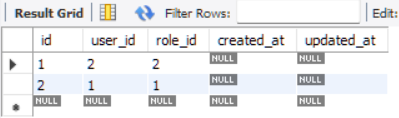
    foreach($user->roles as $role) {

        echo $role->pivot->created\_at;

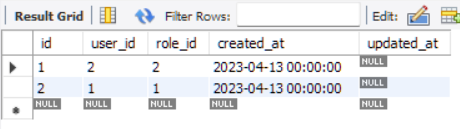
    }

});

設法取得創建pivot表格時的創建時間 => 但是會毫無反應



因為role\_user的表格沒有created\_at的時間，並且要顯示created\_at欄位，需要先在User的模型中定義



先更新資料庫中的created\_at資料

    public function roles() {

        return $this->belongsToMany('\App\Models\Role')->withPivot('created\_at');

    }

再到User模型中定義withPivor(需要顯示的欄位)



# 多重關聯練習

此處會用到三個表格 users / posts / countries，user會屬於某個country，因此需要先創建country的模型

$ php artisan make:model Country –m => 創建Country的模型

$ php artisan make:migration add\_country\_id\_column\_to\_users --table=users => 創建遷移檔案，會將country\_id遷入進users表格中

    public function up()

    {

        Schema::table('users', function (Blueprint $table) {

            //

            $table->integer('country\_id');

        });

    }

    public function down()

    {

        Schema::table('users', function (Blueprint $table) {

            //

            $table->dropColumn('country\_id');

        });

    }

設定add\_country\_id\_column\_to\_users的遷移檔案的migrate和rollback會執行的動作

也就是新增country\_id的欄位，當rollback時刪除country\_id的欄位

    public function up()

    {

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

            $table->id();

            $table->string('name');

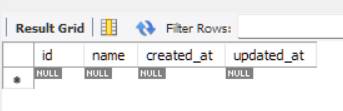
            $table->timestamps();

        });

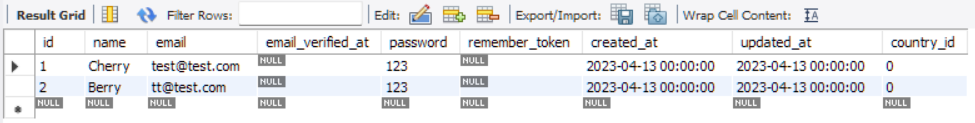
    }

到Country的模型新增需要的欄位，以上完成後，執行以下指令

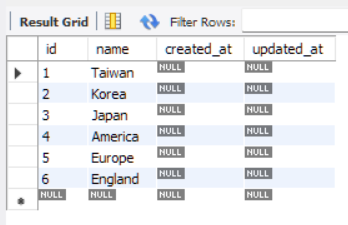
$ php artisan migrate



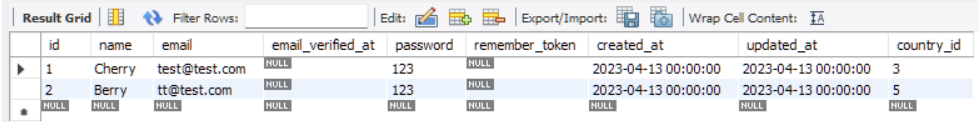
countries的表格如上所示



users則是新增了county\_id的欄位



因此先設置countries的內容



同時設置users關連哪個城市

**遠層一對多**

「遠層一對多」提供了方便簡短的方法，可以透過中介的關聯取得遠層的關聯。

例如，一個 Country 模型可能透過中介的 Users 模型關聯到多個 Posts 模型。讓我們看看定義此種關聯的資料表：



雖然 posts 本身不包含 country\_id 欄位，但 hasManyThrough 關聯透過 $country->posts 來提供我們存取一個國家的文章。要執行此查詢，Eloquent 會檢查中介表 users 的 country\_id。在找到匹配的使用者 IDs 後，就會在 posts 資料表使用它們來查詢。

    public function posts() {

        return $this->hasManyThrough('\App\Models\Post', '\App\Models\User', 'country\_id');

    }

因此到Country的模型定義需要遠層一對多關係的表格

傳遞至 hasManyThrough 方法的第一個參數為我們希望最終存取的模型名稱，而第二個參數為中介模型的名稱。第三個參數為非必要。

如果你想要自定關聯的鍵，你可以傳遞它們至 hasManyThrough 方法的第三與第四個參數。第三個參數為中介模型的外鍵名稱，而第四個參數為最終模型的外鍵名稱。

Route::get('/user/country', function() {

    $country = Country::find(5);

    foreach($country->posts as $post) {

        return $post;

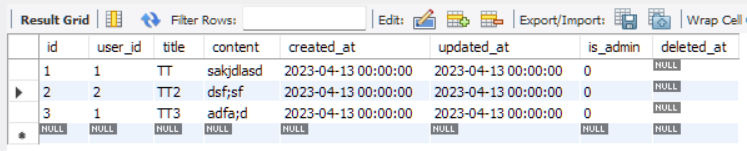
    }

});

先前的遠層一對多設置完畢後，回到routes設置路徑

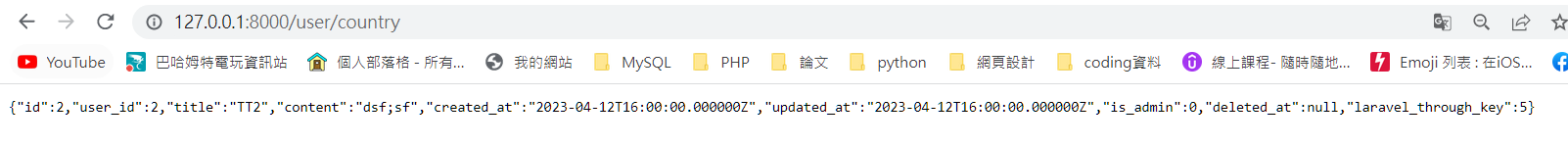
因為先前有定義有兩個user對應到不同的country => Cherry -> 3 ; Berry -> 5

因此要先找出post對應到個別哪個user

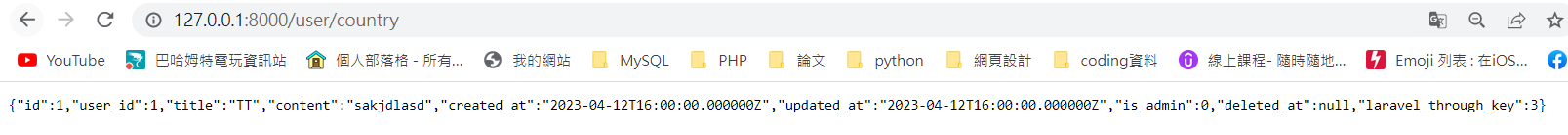


可以看到post 1和post 2分別對應到user 1和user 2

因此使用foreach去尋找資料時，理應會回傳物件到網頁中



這是搜尋country = 5時，會回傳的post



這是搜尋country = 3時，會回傳的post

# 多型關聯(Polymorphic relation)

多型關聯允許一個模型在單一的關聯從屬一個以上的其他模型。

舉個例子，假設你想儲存照片給你的工作人員及你的產品。使用多型關聯，你可以對這兩種情況使用單一的 photos 資料表。

首先，讓我們查看建立這種關聯所需的資料表結構：



有兩個要注意的重要欄位是 photos 資料表的 imageable\_id 和 imageable\_type 欄位。

* imageable\_id 欄位會包含所屬的工作人員或產品的 ID 值
* imageable\_type 欄位會包含所屬的模型類別名稱。

當存取 imageable 關聯時，imageable\_type 欄位會被 ORM 用於判斷所屬的模型是哪個「類型」。

因此先建立Photo的model和migration檔案

$ php artisan make:model Photo –m

建立好後，到photo的migration檔案進行設定

    public function up()

    {

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

            $table->id();

            $table->string('path');

            $table->integer('imageable\_id');

            $table->string('imageable\_type');

            $table->timestamps();

        });

    }

並且到post的migration檔案更改基本設定

    public function up()

    {

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

            $table->increments('id');

            // $table->integer('user\_id')->unsigned();

            $table->string('title');

            $table->text('content');

            $table->timestamps();

        });

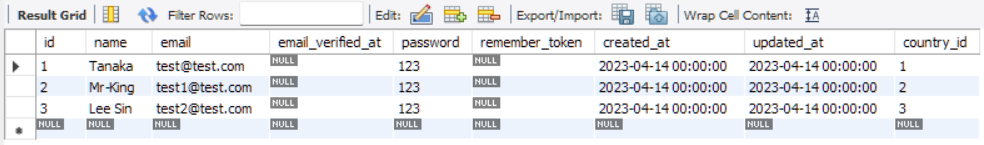
    }

因為user\_id目前用不到，因此可以將其捨棄掉

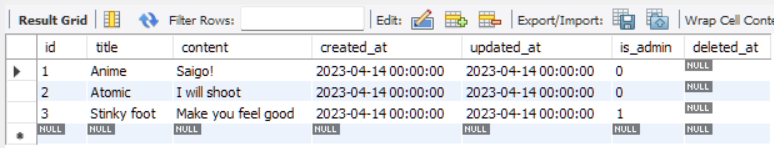
當設定完畢後，執行以下指令

$ php artisan migrate:refresh => 更新資料庫的欄位，但是先前填入的資料都被刪除

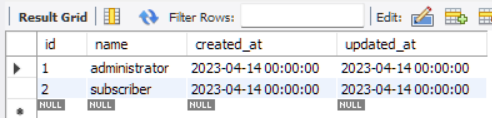
更新完畢後，重新填入需要的基本資料



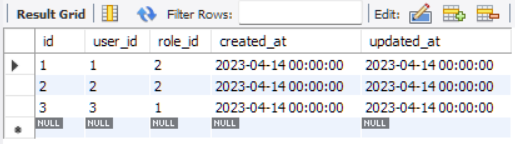
users表格



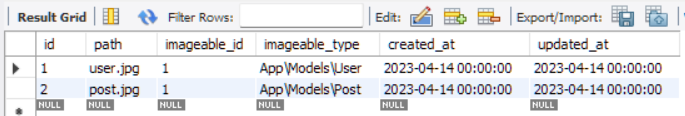
posts表格



roles表格



role\_user中介表



photos表格

    public function imageable() {

        return $this->morphTo();

    }

到Photo的model進行多型設定

透過MySQL的photos的表格，獲取使用的表格id和使用的表格型態

像先前的photos表格的第一筆資料，是取用users表格id=1的資料

|第二筆資料，則是取用posts表格id=1的資料

設定玩Photo的model後，到Post和User的模型進行設定

    public function photos() {

        return $this->morphMany('\App\Models\Photo', 'imageable');

    }

取得所有文章的照片

    public function photos() {

        return $this->morphMany('\App\Models\Photo', 'imageable');

    }

取得所有使用者的照片

以上設定完畢後，可以回到route進行設定

// Polymorphic relationship

Route::get('/user/photos', function() {

    $user = User::find(1);

    foreach($user->photos as $photo) {

        echo $photo->path."<br>";

    }

});

Route::get('/post/photos', function() {

    $post = Post::find(1);

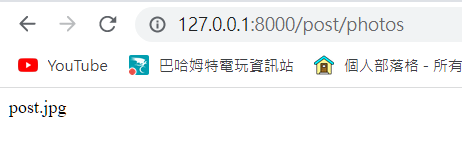
    foreach($post->photos as $photo) {

        return $photo->path;

    }

});





Inverse **遠層一對多**

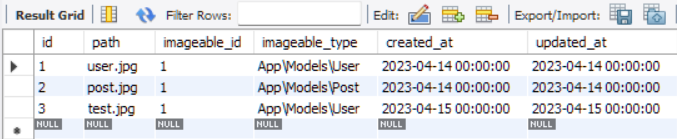
Route::get('/photo/{id}', function($id) {

    $photo = Photo::findOrFail($id);

    return $photo->imageable;

});

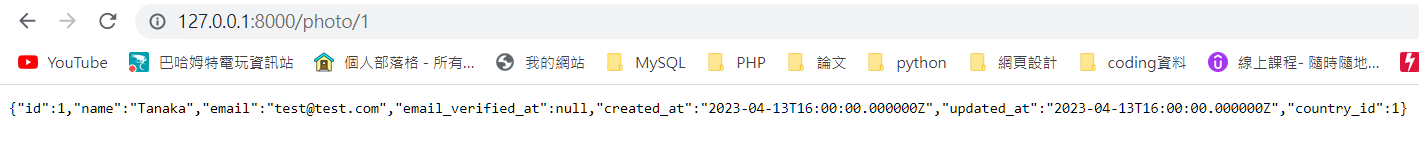
先前是用post和user去取用photo的資料，這次則是反過來用photo的內部資料去取用post和user的資料

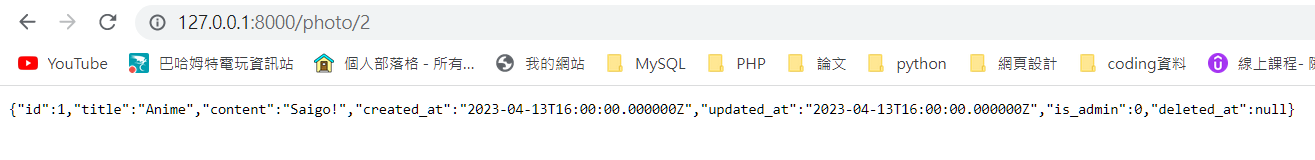


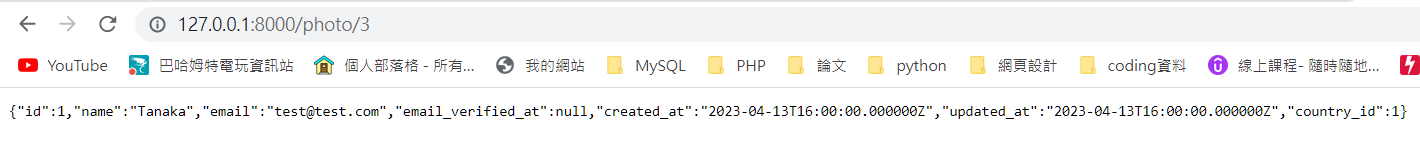
第一筆資料 取用users內部id=1的資料

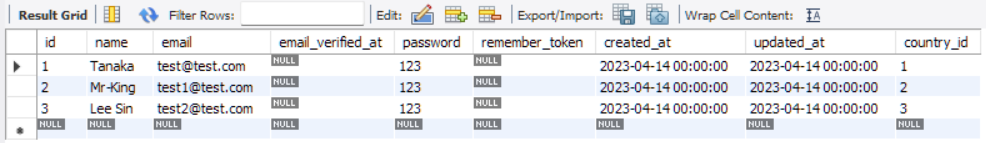
第二筆資料 取用posts內部id=1的資料

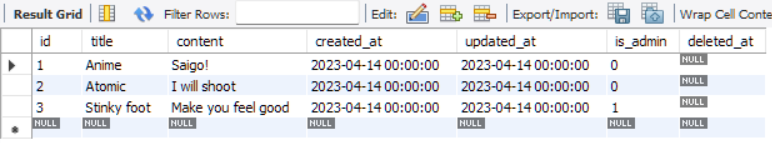
第三筆資料 取用users內部id=1的資料











# 多型關聯—多對多

除了一般的多型關聯，你也可以定義「多對多」的多型關聯。

例如，部落格的 Post 和 Video 模型可以共用多型關聯至 Tag 模型。使用多對多的多型關聯能夠讓你的部落格文章及影片共用獨立標籤的單一列表。

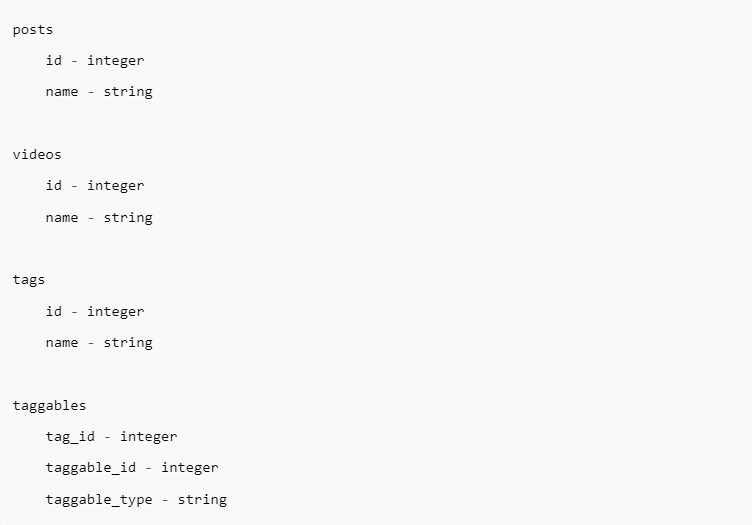
首先，讓我們查看資料表結構：

$ php artisan make:model Video –m

$ php artisan make:model Tag –m

$ php artisan make:model Taggable –m

建立三個Video, Tag, Taggable模型和遷移檔案



    public function up()

    {

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

            $table->id();

            $table->string('name');

            $table->timestamps();

        });

    }

Video模型新增 “影片名稱”的欄位

    public function up()

    {

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

            $table->id();

            $table->string('name');

            $table->timestamps();

        });

    }

Tag模型新增 “標籤名稱”的欄位

    public function up()

    {

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

$table->id();

            $table->integer('tag\_id');

            $table->integer('taggable\_id');

            $table->string('taggable\_type');

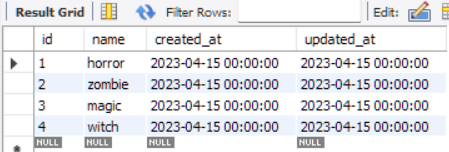
        });

    }

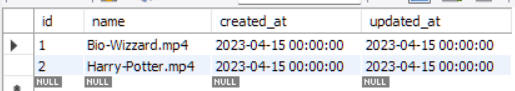
Taggable模型新增 “標籤id”, “多型id”, “多型類型”

執行migrate動作，與資料庫連動

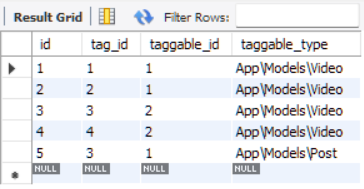
$ php artisan migrate



設置tag資料



設置video資料



設置taggables(關聯)資料

Bio-Wizzard.mp4有horror, zombie的標籤

Harry-Potter.mp4有magic, witch的標籤

post id=1有magic的標籤

    public function tags() {

        return $this->morphToMany('\App\Models\Tag', 'taggable');

    }

到Video的模型設置多型多對多的關聯

    public function tags() {

        return $this->morphToMany('\App\Models\Tag', 'taggable');

    }

到Post的模型設置多型多對多的關聯

    public function posts() {

        return $this->morphedByMany('\App\Models\Post', 'taggable');

    }

    public function videos() {

        return $this->morphedByMany('\App\Models\Video', 'taggable');

    }

到Tag的模型設置多型多對多的關聯

// Polymorphic Many to Many relationship

Route::get('/video/{id}/tag', function($id) {

    $video = Video::find($id);

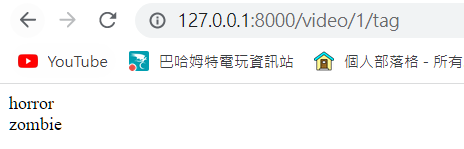
    foreach($video->tags as $tag) {

        echo $tag->name."<br>";

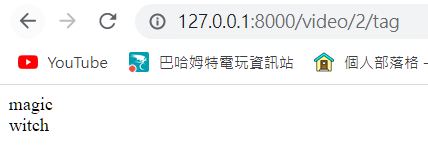
    }

});

設置完畢後，可以透過影片去查找有哪些標籤



Bio-Wizzard.mp4影片有horror, zombie的標籤



Harry-Potter.mp4影片有magic, witch的標籤

Route::get('/post/{id}/tag', function($id) {

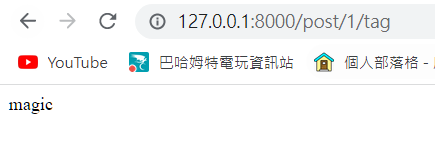
    $post = Post::find($id);

    foreach($post->tags as $tag) {

        echo $tag->name."<br>";

    }

});



Post第一篇文章有magic標籤

Inverse **多型多對多**

Route::get('/tag/{id}/video', function($id) {

    $tag = Tag::find($id);

    foreach($tag->videos as $video) {

        echo $video."<br>";

    }

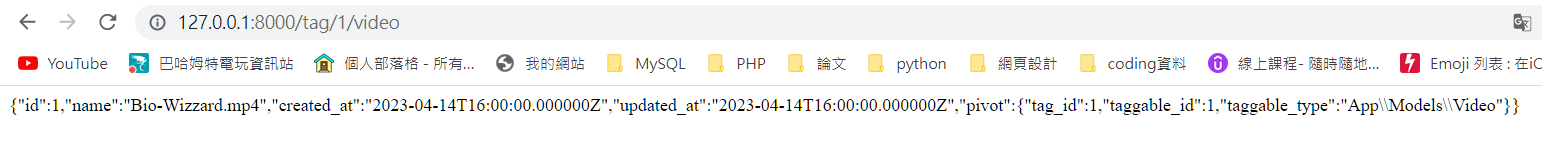
    foreach($tag->posts as $post) {

        echo $post."<br>";

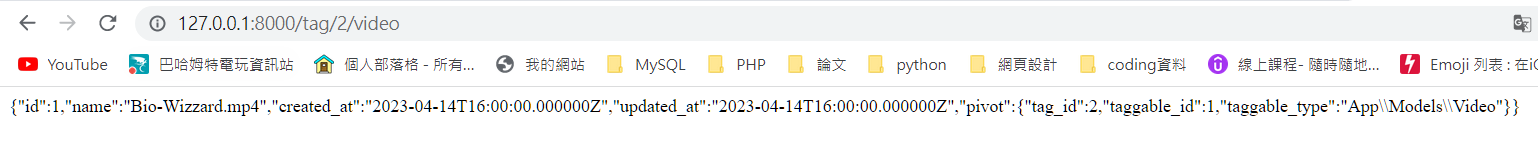
    }

});

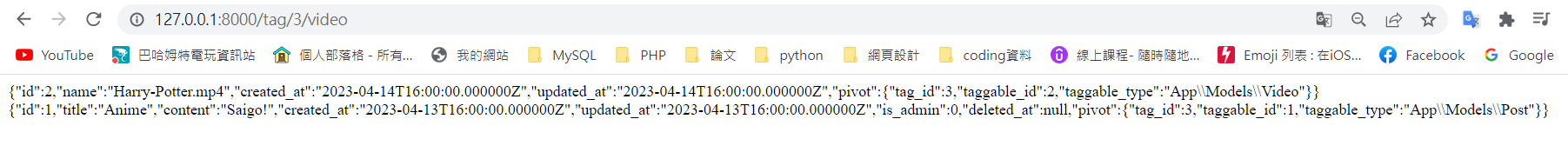
可以透過標籤去查找有那些影片/那些文章



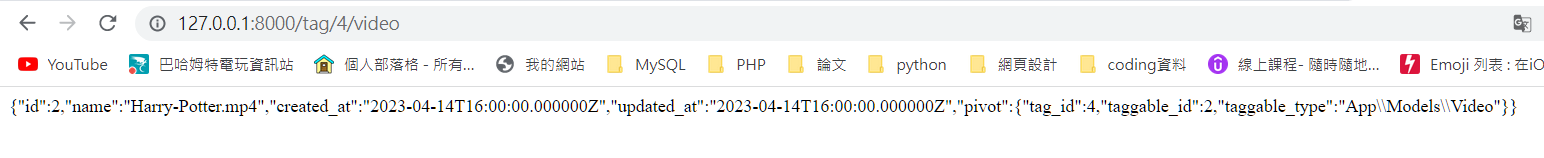
標籤1有Bio-Wizzard.mp4影片



標籤2有Bio-Wizzard.mp4影片



標籤3有Harry-Potter.mp4影片和id=1的Post文章



標籤4有Harry-Potter.mp4影片