

Eloquent ORM

范聖佑 Shengyou Fan
新北市樹林國小 (2015/07/09)

```
1  /**
2   * Display the specified resource.
3   *
4   * @param int $id
5   * @return Response
6   */
7  public function show($id)
8  {
9      $post = Post::with('comments')->where('id', $id)->first();
10
11      return View::make('post.show')->with('post', $post);
12  }
```

單元主題

- 什麼是 ORM？
- Laravel 的 ORM 慣例以及如何透過 ORM 對資料做新增、查詢、更新、刪除等動作
- ORM 間如何設定欄位的關聯？
- 什麼是 tinker 互動指令工具？tinker 可以幫助我們什麼？
- 設定工作坊專案所需的 ORM 關聯

artisan tinker 簡介

tinker 互動指令列

- PHP 本身就有內建互動指令 (REPL) 模式，可以在這個模式底下測試 PHP 程式碼，並立即獲得回饋
 - \$ php -a
- 從 Laravel 4.1 開始整合了 Boris，提供 tinker 互動指令列功能，除了進入 PHP 的 REPL 模式外，並加載 Laravel 所有物件環境，讓開發者可以在指令模式底下測試 Laravel 程式碼 (Boris 相依於 pcntl 外掛，Windows 平台沒有實作，因此 tinker 在 Windows 底下功能受限)
- Laravel 5.0 則是將 Boris 更換為 psysh，除了讓 tinker 功能更多外，也讓 Windows 開發者有完全的功能

```
$ [php] artisan tinker
```

進入 tinker 互動指令列

artisan tinkler

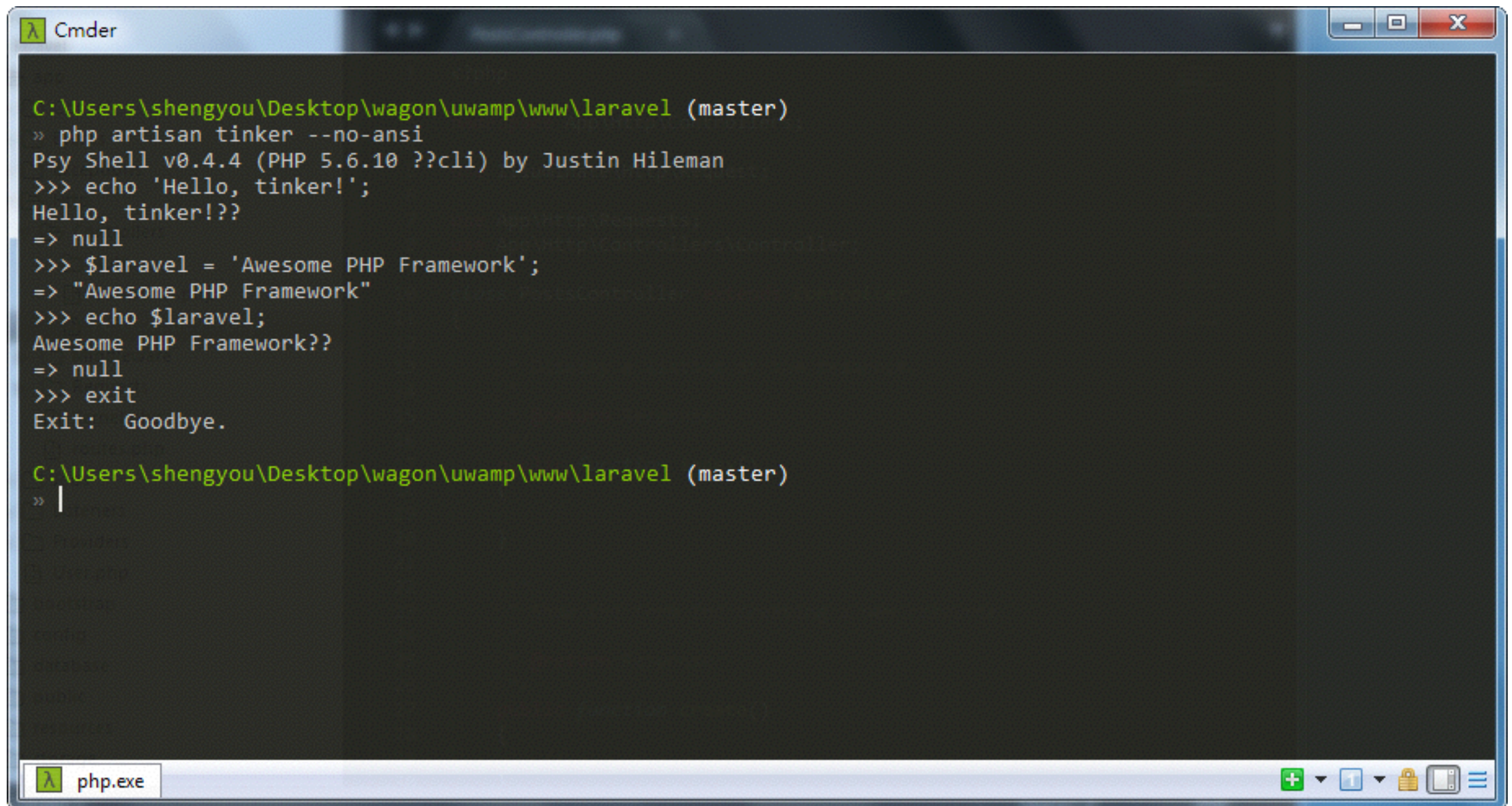
- 進入 PHP 互動 (REPL) 模式，並載入 Laravel 環境
 - 可以直接在指令模式下測試 PHP 程式碼，或是實驗如何操作 Laravel 物件
 - 可用 `exit` 或用 `Ctrl-C` 結束 `tinkler` 模式

- 範例：

```
$ php artisan tinkler
Psy Shell v0.4.4 (PHP 5.6.10 - cli) by Justin Hileman
>>> echo 'Hello, tinkler!'
Hello, tinkler!↵
=> null
>>> exit
Exit: Goodbye.
```

tinker 畫面

在 artisan tinker 底下測試 Laravel 程式



```
C:\Users\shengyou\Desktop\wagon\uwamp\www\laravel (master)
>> php artisan tinker --no-ansi
Psy Shell v0.4.4 (PHP 5.6.10 ??cli) by Justin Hileman
>>> echo 'Hello, tinker!';
Hello, tinker!??
=> null
>>> $laravel = 'Awesome PHP Framework';
=> "Awesome PHP Framework"
>>> echo $laravel;
Awesome PHP Framework??
=> null
>>> exit
Exit: Goodbye.

C:\Users\shengyou\Desktop\wagon\uwamp\www\laravel (master)
>> |
```

dd() 測試工具

- Laravel 內建的除錯函式
 - 在 Laravel 4.2 版以前，`dd()` 其實就是把變數丟進 `var_dump()` 後 `exit()`
 - 在 Laravel 5.0 後，`dd()` 升級成呼叫 Symfony 的 `VarDumper` 元件，畫面更好用了！
 - 可以在任何地方使用，測試程式碼流程、邏輯、了解物件內容
- 範例：

```
$post = 'Hello, world!';  
dd($post);
```


使用 Eloquent ORM

什麼是 ORM ？

- Object-Relational Mapping
- 在寫資料庫查詢時，往往需要組合 SQL 查詢式，除了要自行處理字串跳脫等安全性防護機制外，也要自行轉換從資料庫取出的資料格式。當查詢式變長、關聯日趨複雜時，在撰寫對應的功能時會更為辛苦
- 而 ORM 則是依照慣例，把資料庫裡一筆一筆的資料變成類似物件的概念來操作。這些物件透過繼承增加了更多的方法，讓資料庫的操作 (CRUD) 更加直覺、方便

Laravel 的 ORM 慣例

- Laravel 的 Model/ORM 依照以下慣例：
 - 一種資源 (resource) 對應到資料庫裡的一個資料表 (table)
 - 一個 Model 對應到資料表裡的一列 (row)
 - 每一個 Model 裡預設都要有一個 auto-increment 的 id 做為 primary key，並預設有 created_at 及 updated_at 紀錄 Model 產生及更新的時間戳記
 - 資料表的名稱用英文複數、蛇底式命名；而 Model 的名稱就用英文單數、大駝峰式命名

查詢資料

- 取出全部資料
 - `Model::all()`
- 用 primary key 取出單或數筆資料
 - `Model::find($id)`、`Model::find([/* ids */])`
- 增加查詢條件式
 - `Model::where('欄位', '條件', '值')`
 - `Model::orderBy('欄位', '排序方式')`
- 串聯多個條件
 - `Model::where(/*略*/)->orderBy(/*略*/)->get()`

Collection 類別

- Laravel 實作了 **Collection** 類別，擴充了原有陣列 (array) 的功能，讓從資料表取出來的資料更好操作
- 用 Eloquent 查詢取得的回傳物件就是 **Collection**，由於其實作了數個 PHP 預先定義的介面，包括：
ArrayAccess、Arrayable、Countable、
IteratorAggregate、Jsonable、
JsonSerializable，所以可以直接當陣列使用，或是直接輸出成 JSON 格式

Collection 的運用方式

- 透過 Model 查詢資料庫後，Laravel 會回傳一個 Collection 物件，裡面放了回傳資料的 Model，其行為就像陣列一樣，可以支援用 foreach 取出資料

```
$posts = \App\Post::all(); // $posts 是 Collection
foreach($posts as $post)
{
    echo $post->title; // $post 是 Model
}
```

- 若是用於 API，也可直接轉成 JSON 輸出

```
$posts = \App\Post::all();

return $posts->toJson();
```

Helper 與 Facade

- 平常也可以將陣列轉成 `Collection` 使用，可以用 `collect()` helper function 或 `Collection Facade`

```
// 使用 helper function  
$collection = collect([1, 2, 3]);
```

```
// 使用 Facade  
$collection = Collection::make([1, 2, 3]);
```

是 Model 還是 Collection ?

- 若是查詢回傳的是一個資料集 (Recordset) 就是 Collection ; 若查詢回傳的是一筆資料 (Row) 就是 Model

```
// Collection
```

```
$posts = \App\Post::all();
```

```
$hotPosts = \App\Post::where('is_feature', 1)->get();
```

```
// Model
```

```
$post = \App\Post::find($id);
```

```
$post = \App\Post::orderBy(/*略*/)->first();
```


新增資料

- 使用 `new` 建構式
 - 直接使用 `new` 建構式產生 Model 實體，再存檔
- 範例：

```
$post = new \App\Post;  
$post->title = 'My Post Title';  
$post->save();
```

新增資料

- 使用 Facade 的 `create` 方法
 - 直接從陣列新增一筆資料，陣列的 `key` 值自動對應到資料表的欄位
- 語法：
 - `Model::create($array)`

- 範例：

```
$post = \App\Post::create([  
    'title' => 'My Title',  
    'sub_title' => 'My Sub Title',  
    'content' => 'Post Content',  
    'is_feature' => true,  
]);
```

Mass Assignment

- Laravel 的 ORM 可以直接用 Mass Assignment 的方式直接新增資料，透過陣列 key 值與資料表欄位名稱的對應可以迅速的將資料寫入資料庫內
- 這樣的作法雖然語法簡潔、快速方便，但若使用不當可能會有安全性問題，比方說密碼欄位有可能因此被覆寫
- 為了提升安全性，在 Model 裡預設將所有的欄位都設定為不可使用 Mass Assignment 做為防禦手段。若要使用的話，則要設定 **fillable** 屬性將要使用 Mass Assignment 的欄位打開

Mass Assignment 警告

The screenshot shows a web browser window with the address bar set to `localhost:8000`. The page displays a "Whoops! There was an error" message. The error details are as follows:

- Exception:** `Illuminate\Database\Eloquent\MassAssignmentException`
- File:** `C:\Users\shengyou\Desktop\wagon\uwamp\www\laravel\vendor\laravel\framework\src\Illuminate\Database\Eloquent\Model.php`
- Line:** 404

The stack trace on the left lists the following frames:

- 35. `Illuminate\Database\Eloquent\Model::fill` (Line 404)
- 34. `Illuminate\Database\Eloquent\Model::fill` (Line 259)
- 33. `Illuminate\Database\Eloquent\Model::__construct` (Line 527)
- 32. (Line 527)

The code snippet on the right shows the relevant code in `Model.php`:

```
397.         // assignment to the model, and all
398.         others will just be ignored.
399.         if ($this->isFillable($key))
400.         {
401.             $this->setAttribute($key,
402.             $value);
403.         }
404.         elseif ($totallyGuarded)
405.         {
406.             throw new
MassAssignmentException($key);
407.         }
408.     }
```

At the bottom, it says "No comments for this stack frame."

設定 fillable、guarded

- 在 Model 內設定 **fillable** 屬性，指定哪些欄位可以透過 Mass Assignment 來寫入/更新資料
- **guarded** 則是相反的屬性，設定後可以保護特定欄位不使用 Mass Assignment
- 範例：

```
class Post extends Model {  
    protected $fillable = [  
        'title',  
        'sub_title',  
        'content',  
        'is_feature'  
    ];  
    protected $guarded = ['id', 'password'];  
}
```

刪除資料

- 使用 Model 的 `delete` 方法
 - 先取出 Model 實體後，再刪除

- 語法：

- `$model->delete()`

- 範例：

```
$post = \App\Post::find(1);  
$post->delete();
```

刪除資料

- 使用 Facade 的 `destroy` 方法
 - 用 `primary key` 刪除對應的資料，可支援多筆刪除
- 語法：
 - `Model::destroy(/* primary key */)`

- 範例：

```
// 使用 primary key 刪除一筆資料  
\App\Post::destroy(1);
```

```
// 使用 array 刪除多筆資料  
\App\Post::destroy([1, 2, 3]);
```

```
// 使用多參數傳值刪除多筆資料  
\App\Post::destroy(1, 2, 3);
```

更新資料

- 使用 Model 的 `update` 方法
 - 先將資料庫取出 Model 實體，再從陣列更新資料，陣列的 `key` 值自動對應到資料表的欄位
- 語法：
 - `$model->update($array)`
- 範例：

```
$post = \App\Post::find(1);  
$post->update([  
    'title' => 'new title',  
    'sub_title' => 'new sub title',  
    'content' => 'new content',  
    'is_feature' => false,  
]);
```

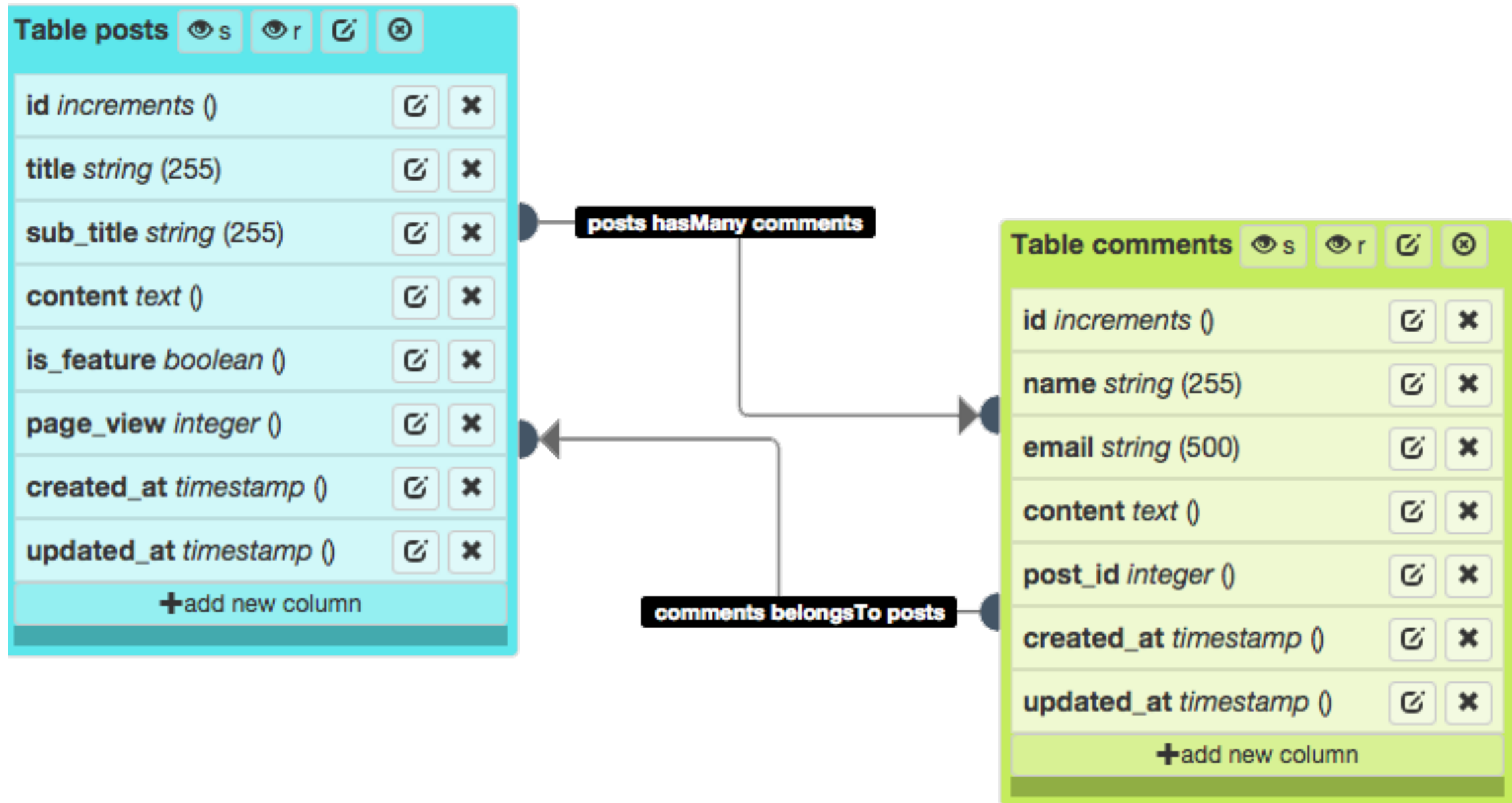

儲存資料

- 使用 Model 的 `save` 方法
 - 先取出 Model 實體後，再針對屬性更新，全部完成後再存檔
- 語法：
 - `$model->save()`
- 範例：

```
$post = \App\Post::find(1);  
$post->title = 'My New Title';  
$post->sub_title = 'My New Sub Title';  
$post->content = 'My New Content';  
$post->is_feature = false;  
$post->save();
```

資料表關聯

資料表間的關聯



Model 關聯類型

- A 擁有多個 B
 - 一對多
 - 例：一個 Post 有很多個 Comment
 - `hasMany()`
- B 歸屬於 A
 - 一對多反向
 - 例：每個 Comment 都會屬於一個 Post
 - `belongsTo()`

資料關聯設定

- 只要在 Model 內定義自己與其他 Model 間的關聯類型，Laravel 就會自動將這個關聯變成物件間的屬性，在查詢時完全不需要自行下 SQL 語法

- 語法：

```
public function {另一個 Model 的名稱(單/複數)}()  
{  
    return $this->{關聯類型}('{Model 名稱}');  
}
```

- 範例：

```
/* app/Post.php */  
public function comments()  
{  
    return $this->hasMany('App\Comment');  
}
```

透過 Model 取得關聯資料

- 由於 ORM 已經把資料表的 row 變成一個物件，row 裡的資料變成物件的屬性值，而物件間的關聯也一樣是屬性值，透過這個屬性值就可以取得關聯後的 Model，再用一樣的方式印出該 Model 的資料即可

- 範例：

```
// 先取得 Post 實體
```

```
$post = \App\Post::find(1);
```

```
echo $post->title;
```

```
// 把 Post 的 Comment 印出來
```

```
foreach($post->comments as $comment)
```

```
{
```

```
    // $post->comments 是一個 Collection
```

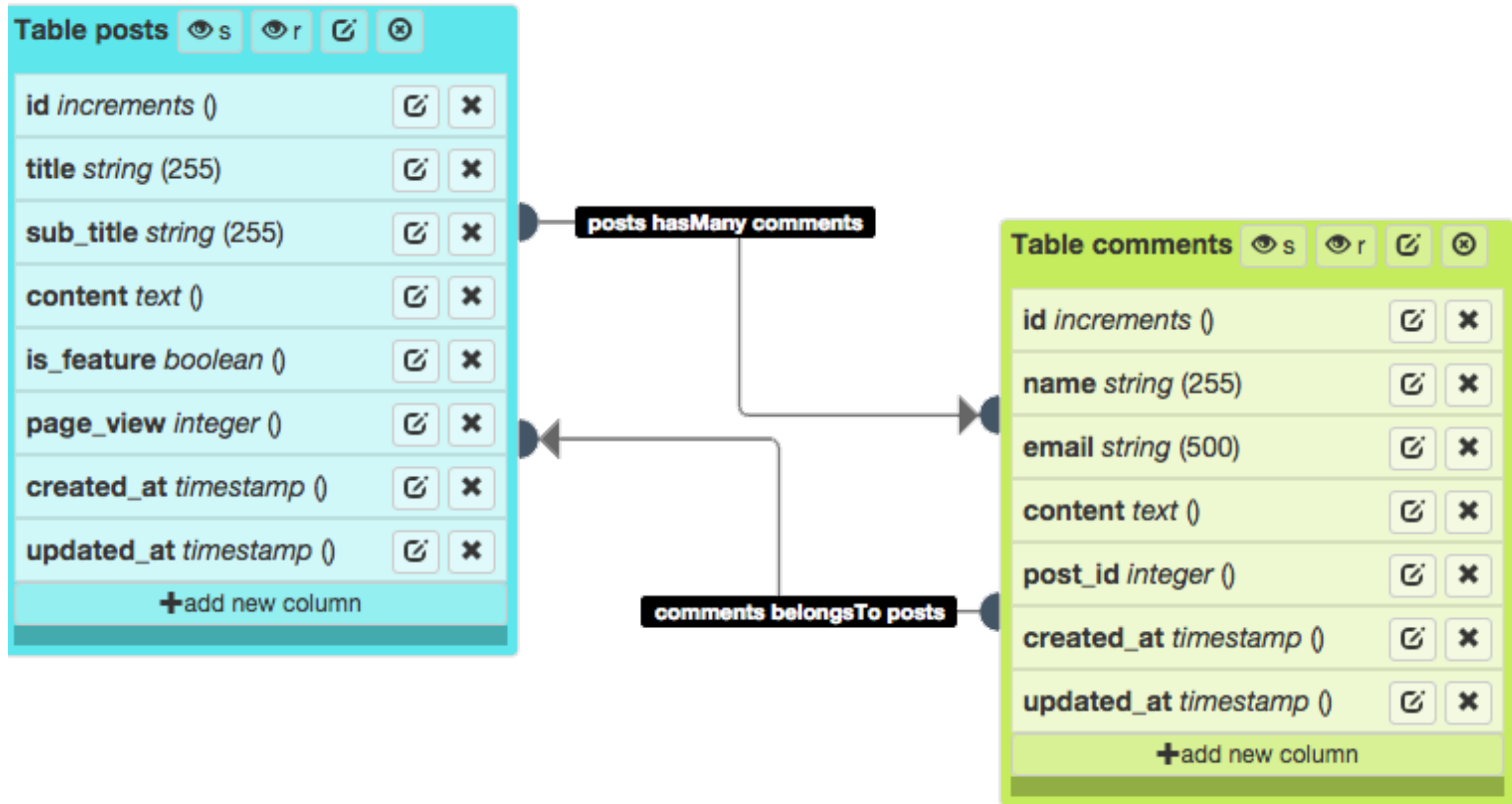
```
    // $comment 就是 Comment Model
```

```
    echo $comment->content;
```

```
}
```

設定專案 Model

專案資料庫設計



設定 Model、測試 Eloquent

- 設定三個 Model 內的 `fillable` 屬性
- 設定 Post 及 Comment 兩個 Model 的資料關聯
- 在 `php artisan tinker` 裡測試 Eloquent 的使用方式
- 在 `app/Http/routes.php` 裡測試 Eloquent 的使用方式
- 練習 `dd()` 的使用方式

存檔點

- 試著把現在已經可以運作的程式碼加入版本控制內
- 流程提醒：
 - working directory > staging area > commit

單元總結

- 在這個單元裡我們學到了些什麼？
 - tinker 的使用方式
 - Laravel 的 ORM 慣例及資料表關聯設定
 - 使用 ORM 來操作資料庫內的資料



歡迎提問討論