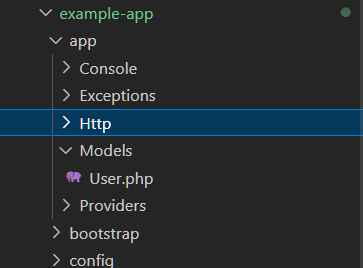
# Laravel eloquent ORM

Eloquent ORM 提供了簡潔的實作和資料庫互動，每個table都有一個對應的 "Model" 可以跟table互動，在做完前面DB的設定以後[Entity / DB簡介]，就可以使用。

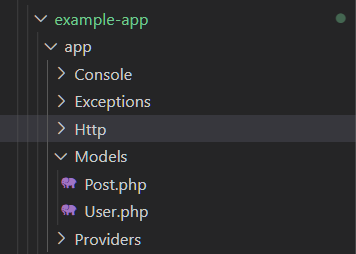
# **Create** Model檔案

$ php artisan make:model Post –m => 若有加後方的-m，則會建立遷移檔案

$ php artisan make:model Post => 若無則建立Model檔案



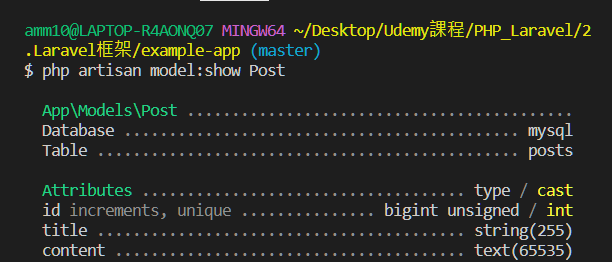
model檔案會建立在app/models/的資料夾中



在terminal中輸入上述的指令就會建立新的model檔案

#注意 : 資料庫中的表格應為複數的單字；Laravel中的Model檔案名稱應為單數的單字

例如: MySQL => posts； Laravel Model => post



可以使用php artisan model:show model\_name來檢視先前建立的model內容

# **Eloquent ORM 表名**

先前建立model時，沒有告訴eloquent哪個資料庫對應建立的Post模型。

按照eloquent的慣例，除非明確指定表格的名稱，否則使用”snake case”蛇型命名法，class的複數名稱將用作於表明。

因此Eloquent將會假設Post模型將記錄存儲在Posts表格中；而AirTrafficController模型會將記錄存儲在air\_traffic\_controllers表中。

#補充 : 蛇型命名法

簡單來說是用 \_ 來分隔單字的命名規則，像是 fibonacci\_sequence

#自行定義 – model name

若模型的名稱對應到資料庫中的表格不符合上述的規定，則可以到model中自定義屬性來手動指定model的表名。

class Post extends Model

{

    use HasFactory;

    protected $table = "new\_blogger";

}

若建立的model名稱為post，則相對的資料庫要存在posts的表格；但是此處重新定義了post模型對應的表格名稱為new\_blogger。

#eloquent – 指定PK

class Post extends Model

{

    use HasFactory;

    protected $table = "new\_blogger";

    protected $primaryKey = 'id';

}

甚至也可以透過eloquent指定primary key。

# **Model -> Database 取用資料**

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class Post extends Model

{

    use HasFactory;

}

App/Models/Post.php的內部程式碼如上

use App\Models\Post;

建立好models後，到routes/web.php中引入Post.php檔案，使得web.php可以使用Post擴展models的所有功能。

/\*

|--------------------------------------------------------------------------

| ELOQUENT ORM(OBJECT RELATION MODEL)

|--------------------------------------------------------------------------

\*/

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

  $posts = Post::all();

  foreach($posts as $post) {

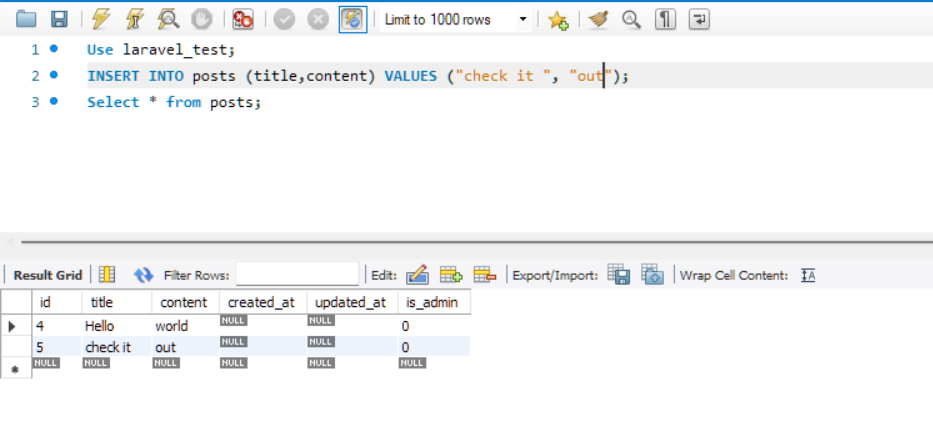
    return $post->title;

  }

});

$posts儲存Post(database => posts)的所有資料

透過foreach將$posts中的每筆資料一一取出，但是因為使用return，因此只會跑一次結果





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

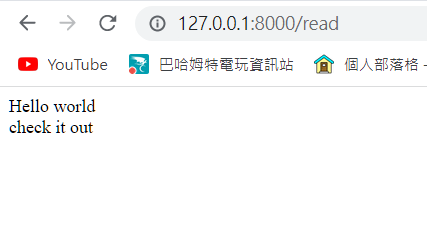
  $posts = Post::all();

  foreach($posts as $post) {

    echo $post->title." ".$post->content."<br>";

  }

});



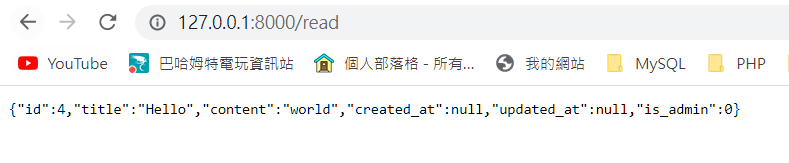
使用echo就可以將posts表格中的所有資料一一列印出來

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

  $posts = Post::find(4);

  return $posts;

});



使用find，可以尋找單筆資料，find(索引)

# **Reading/Finding with constraints**

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

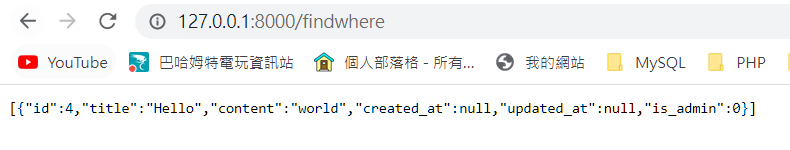
  $posts = Post::where('id', 4)->orderBy('id', 'desc')->take(1)->get();

  return $posts;

});

使用條件去取用資料

取用posts表格中id=4的資料，orderBy(排序)id並且由大至小(DESC)，take(取幾筆資料)，get()回傳結果。



# **More way to retrieve data**

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

//   $posts = Post::findOrFail(1);

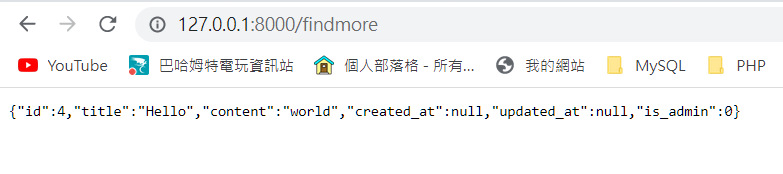
//   return $posts;

    $posts = Post::where('id', '<', 50)->firstOrFail();

    return $posts;

});

findOrFail()和firstOrFail()是在條件中，若有找尋到符合條件的資料則回傳該筆資料，若沒有找尋到則丟出一個error資訊。



# **Inserting/Saving Data**

  DB::insert('insert into posts(title, content) value(?, ?)', ['PHP with laravel', 'Laravel is the best thing that has happened to PHP']);

先前使用的是直接與資料庫進行互動，因此都是用SQL的語法

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

    $posts = new Post;

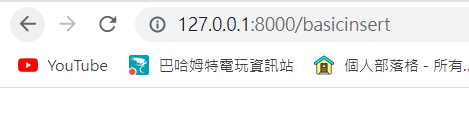
    $posts->title = 'New ORM title insert';

    $posts->content = 'Elequent is too hard to me...';

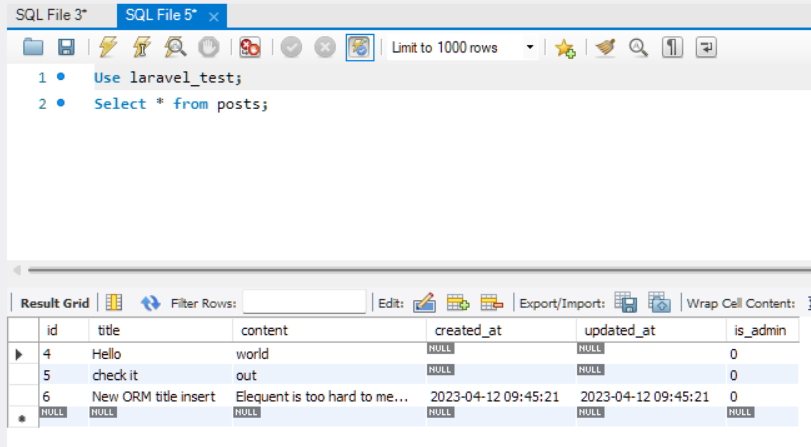
    $posts->save();

});

透過呼叫model來與MySQL的資料庫進行互動



到該網頁後會沒有任何反應 => 因為直接儲存資料進posts的表格中了



可以發現id=6有新增一筆資料，且連創建和更新時間都有

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

    $posts = Post::find(6);

    $posts->title = 'New ORM title insert 2';

    $posts->content = 'Elequent is too hard to me... 2';

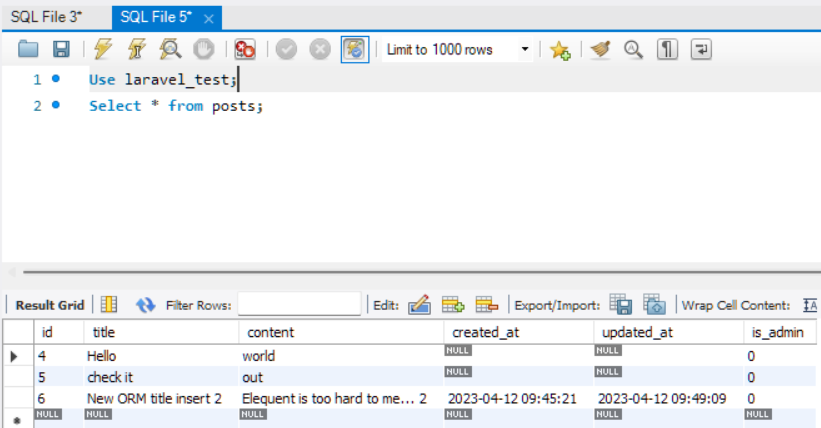
    $posts->save();

});

前一個語法是insert進表格中，此處為update表格的內容

因為不需要新建立insert的內容，只需從存在的物件中進行修改，因此不用new Post，而是從post中找尋id=6的資料

修改完畢後save()，並且前往localhost:8000/basicupdate會出現空白頁面



而MySQL WorkBench的內容也有隨之修改

# **Creating data and configuring mass assignment**

批量分配是發送一組數據的過程，這些數據將立即保存到指定的模型。一般來說，你不需要一個一個地在你的模型上保存數據，而是在一個單一的過程中。

批量分配固然好，但背後存在一定的安全問題。如果有人將一個值傳遞給模型並且在沒有保護的情況下，他們肯定可以修改包括 ID 在內的所有字段怎麼辦？這並不好。

Fillable 允許您指定模型中哪些字段是可批量分配的，您可以通過將特殊變量添加$fillable到模型來實現。

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

    Post::create(['title'=>'the create method', 'content'=>'I want to learn more thing about PHP Laravel']);

});



無法進行批量分配，因為沒有允許title和content字段能進行分配

class Post extends Model

{

    use HasFactory;

    protected $fillable = [

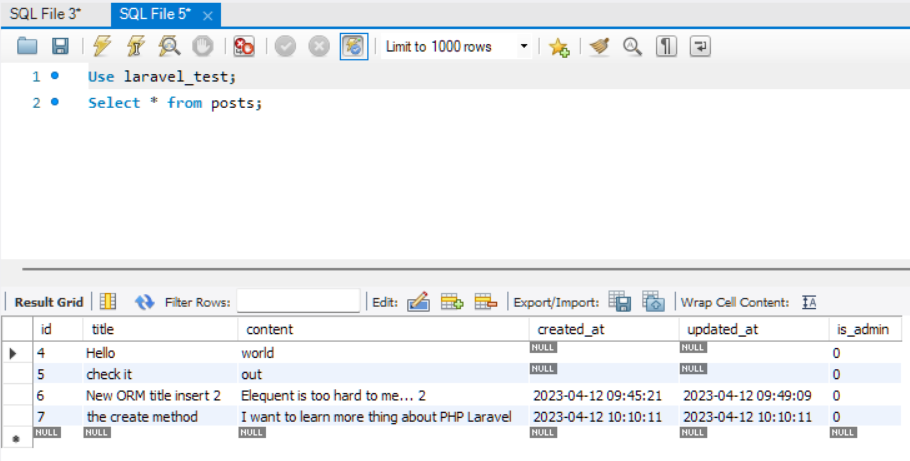
        'title',

        'content'

    ];

}

因此到Post.php的model中設置fillable特殊變量，能夠允許哪些欄位可以進行更改，此處新增的為title和content

透過fillable新增title和content欄位後，再次前往localhost:8000/create頁面時，就會完成新增一筆資料的動作

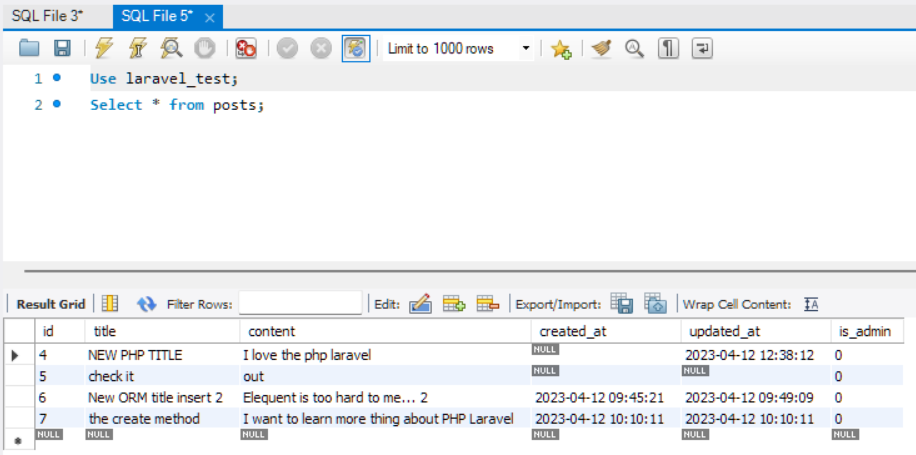
# **Updating with Eloquent**

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

    Post::where('id', 4)->where('is\_admin', 0)->update(['title'=>'NEW PHP TITLE', 'content'=>'I love the php laravel']);

});

首先尋找post中id=4且is\_admin為0(false)的資料，並且更改該筆資料的title和content



id=4的資料卻時更改，且原先沒有create和update的時間，此時update的時間卻更新

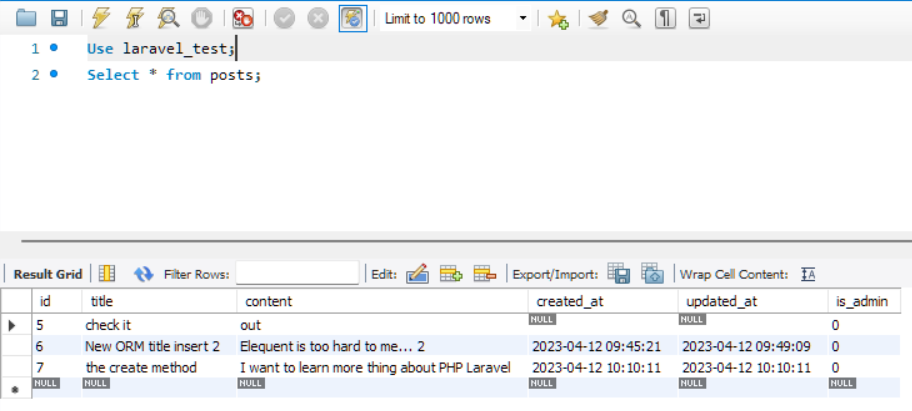
# **Deleting with Eloquent**

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

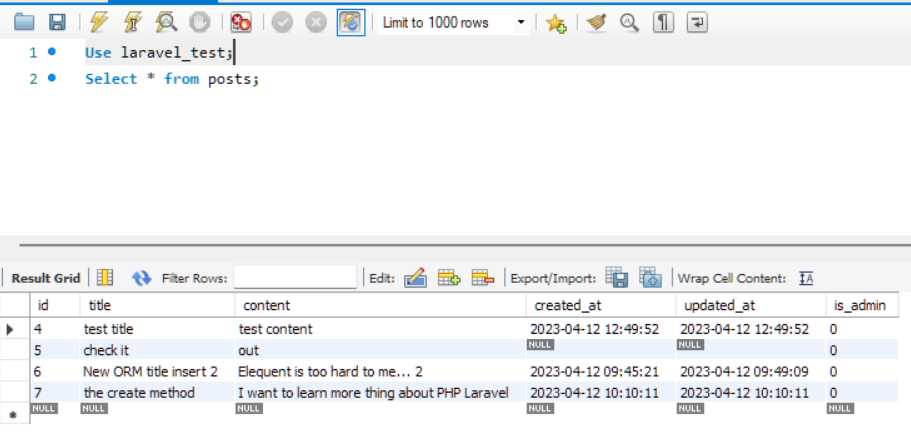
    Post::where('id', 4)->delete();

});

(自己設定的)直接找出id=4的資料，並且刪除



首先將id=4的資料先重新輸入進去



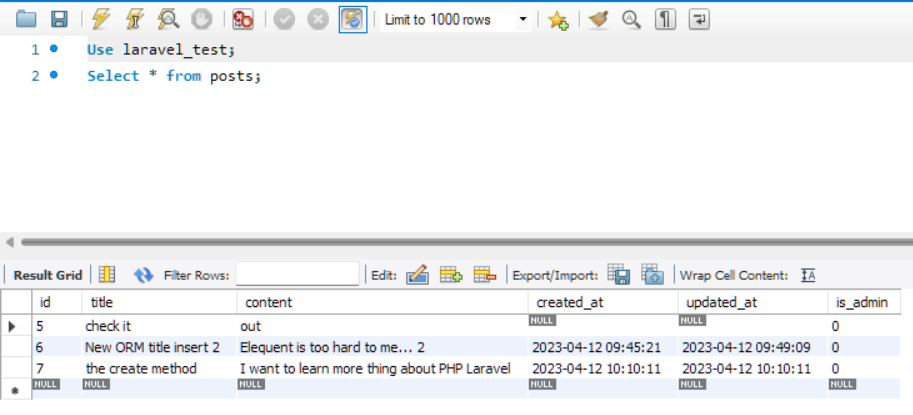
Route::get('/delete', function() {

    $posts = Post::find(4);

    $posts->delete();

});

先找出posts id=4的資料，再執行刪除的動作。

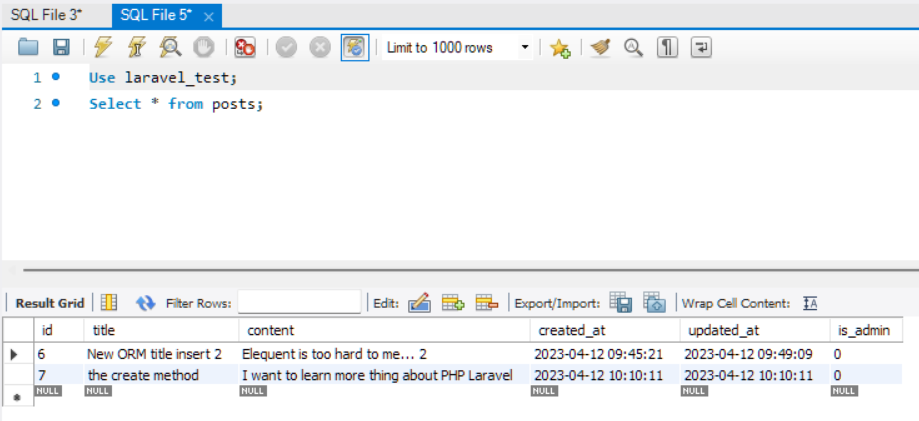


Route::get('/delete1', function() {

    Post::destroy(5);

});

其他刪除的方式為destroy()，直接刪除id=5的資料



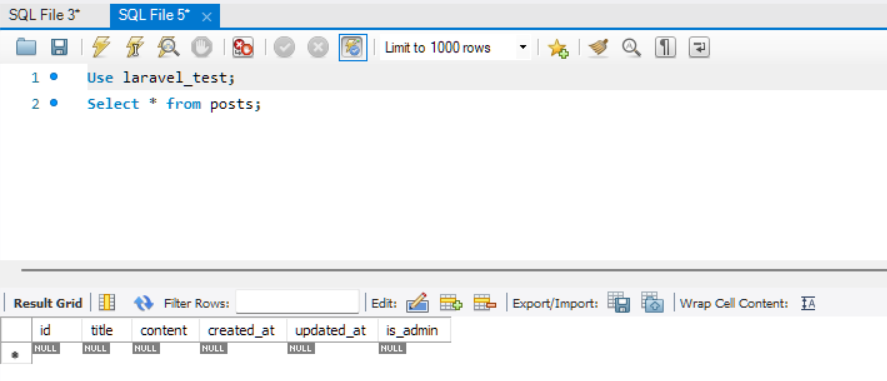
Route::get('/delete1', function() {

    Post::destroy([6,7]);

    Post::where('is\_admin', 0)->delete();

});

上述兩種方式，可以將剩下的資料都刪除掉。



# **Soft Deleting/Trashing**

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\SoftDeletes;

class Post extends Model

{

    use HasFactory;

    use SoftDeletes;

    protected $dates = ['deleted\_at'];

    protected $fillable = [

        'title',

        'content'

    ];

}

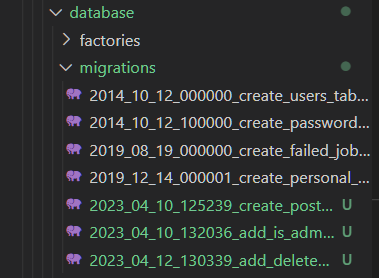
先到models中新增要使用到的新功能 softdeletes

並且要定義protected $dates => $dates是在Model中的其中一項功能

儲存deleted\_at變數，該變數是用來記錄softDeletes的日期

新增完後要到database的migration中新增遷移檔案

$ php artisan make:migration add\_deleted\_at\_column\_to\_posts\_table --table=posts



    public function up()

    {

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

            //

            $table->softDeletes();

        });

    }

    /\*\*

     \* Reverse the migrations.

     \*

     \* @return void

     \*/

    public function down()

    {

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

            //

            $table->dropColumn('deleted\_at');

        });

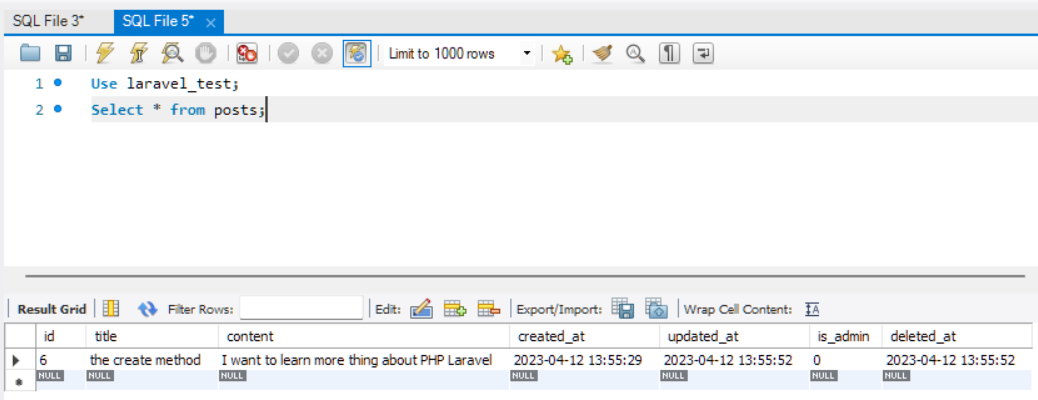
    }

再回到routes中設定需要softdelete的該筆資料

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

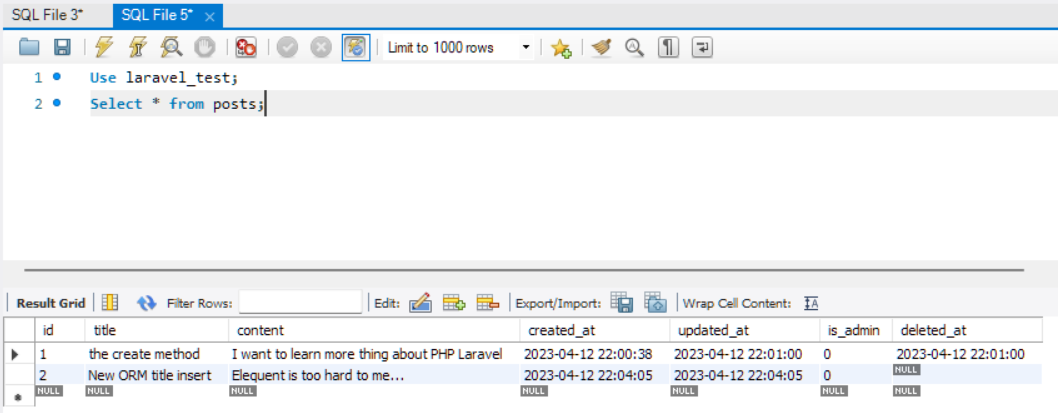
    Post::find(6)->delete();

});



當執行delete()時，該筆資料沒有被刪除掉，但是deleted\_at儲存刪除的時間記錄

# **Retrieving deleted/trashed records**



首先執行basicinsert的內容，新增一筆資料

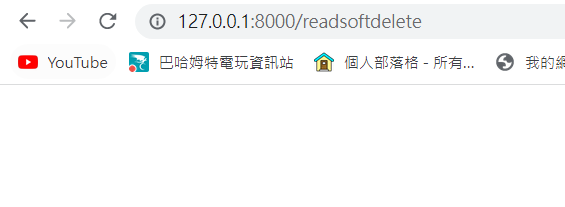
Route::get('/readsoftdelete', function() {

    $posts = Post::find(1);

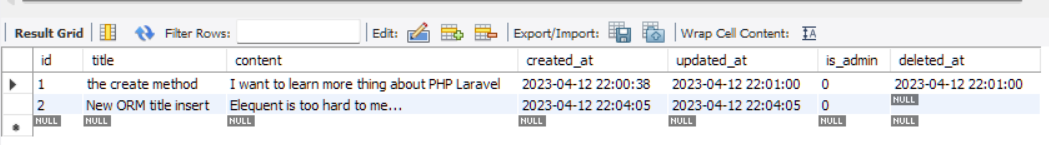
    return $posts;

});

新增readsoftdelete的頁面，該頁面會讀取先前刪除id=1的資料



但是因為該筆資料先前刪除了，因此部會顯示在頁面中



但是資料庫還有該筆資料存在

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

    // $posts = Post::find(1);

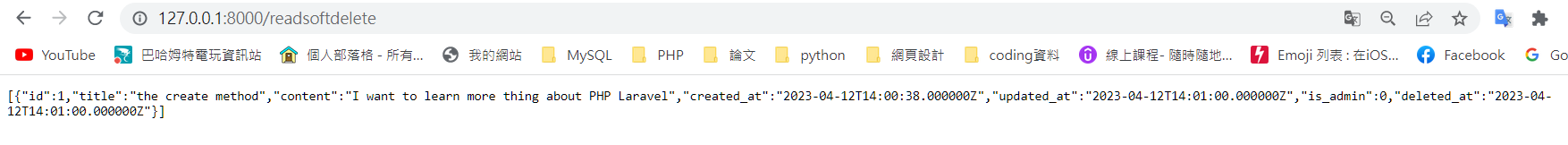
    // return $posts;

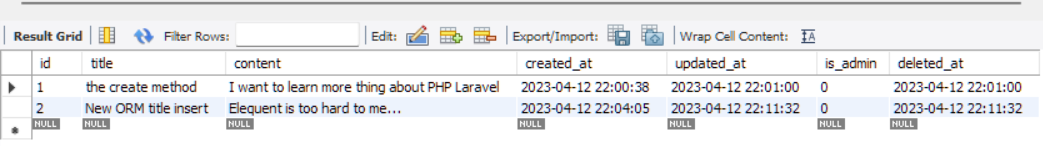
    $posts = Post::withTrashed()->where('id', 1)->get();

    return $posts;

});

使用withTrashed()去查詢已刪除，但尚未完全刪除的資料





同樣將第二筆資料丟進垃圾桶中

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

    // $posts = Post::find(1);

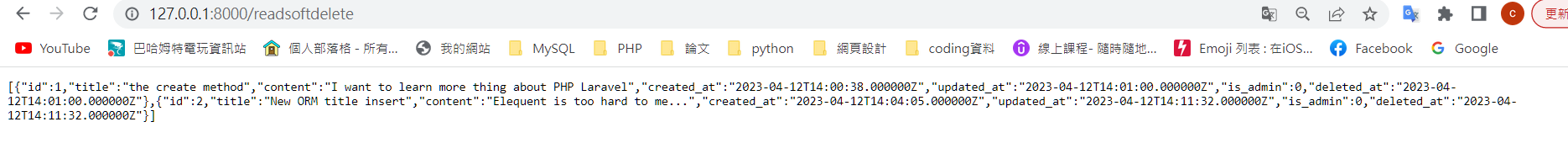
    // return $posts;

    $posts = Post::withTrashed()->where('is\_admin', 0)->get();

    return $posts;

});

用is\_admin為0的方式去搜尋同樣在垃圾桶中的資料並回傳



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

    // $posts = Post::find(1);

    // return $posts;

    // $posts = Post::withTrashed()->where('is\_admin', 0)->get();

    // return $posts;

    $posts = Post::onlyTrashed()->get();

    return $posts;

});

onlyTrashed()同樣是搜尋在垃圾桶中的資料，若沒有給與條件則回傳在垃圾桶中的資料。

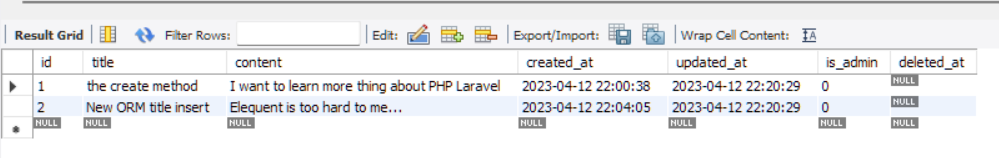
# **Restoring deleted/trashed records**

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

    Post::withTrashed()->where('is\_admin', 0)->restore();

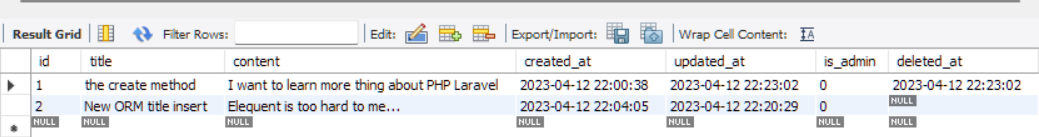
});

單純用withTrashed()搜尋被刪除的資料，並且使用restore()將其恢復



成功後deleted\_at的刪除時間都被重置為Null => 刪除資料前

# **Deleting a record permanently**

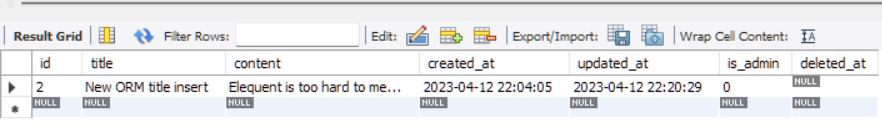


先將其中一筆資料再次設定為softdelete的資料

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

    Post::onlyTrashed()->where('is\_admin', 0)->forceDelete();

});



使用forcedelete()後，會將符合條件且在垃圾桶中的資料給強制刪除(永久刪除)