**Bài 13: Các mối quan hệ (Relationships) trong Eloquent**

-Trong Bài trước mình đã giới thiệu cho mọi người về Eloquent ORM trong laravel rồi, nhưng vấn đề là trong khi lập trình các ứng dụng thực tế thì sẽ tồn tại ra các mối quan hệ giữa các bảng với nhau chính vì điều đó Laravel đã cung cấp cho chúng ta các mối quan hệ có sẵn để giảm thiểu thời gian cũng như công sức của các Lập trình Viên.

**1, Các mối quan hệ trong Eloquent.**

-Trước đây Laravel version 4.0 thì chỉ cung cấp cho chúng ta 3 mối quan hệ chính, nhưng đến phiên bản hiện tại con số đó đã được nâng lên là **6**mối quan hệ.

-Gồm:

* **One to One** : Một Một.
* **One to Many**: Một nhiều.
* **Many to Many**: Nhiều nhiều.
* **Has Many Through**: Nhiều nhiều qua lại trung gian.
* **Polymorphic Relations**: Đa hình.
* **Many To Many Polymorphic Relations** : Nhiều nhiều đa hình.

-Hơi phức tạp đúng không mọi người? Để mình đi vào ví dụ cụ thể cho mọi người hiểu hơn nhé.

**2, One To One.**

-Đây là mối quan hệ đơn giản n nhất trong tất cả các mối quan hệ mà Laravel cung cấp.

 VD: Mình có một bảng **Posts** để đăng bài và một bảng **Featured\_images**để đặt ảnh đại diện cho bài post đó . Mình sẽ tạo ra mối quan hệ **1- 1** như sau: một bài viết của mình chỉ có thể có một ảnh đại diện và ngược lại một ảnh đại diện mình sẽ chỉ dùng cho một bài viết.

-Khi đó mình sẽ phải tạo 2 model là **Posts.**

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Posts extends Model

{

protected $table='Posts';

public $timestamp=false;

public function FeaturedImages()

{

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

}

}

và**FeaturedImages:**

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class FeaturedImages extends Model

{

protected $table='FeaturedImages';

public $timestamp=false;

}

- Khi này các bạn chỉ cần triệu gọi với lệnh **Eloquent** dạng như sau:

App\Posts::find(1)->FeaturedImages;

**-Chú ý:**Câu lệnh ở trên chỉ đúng nếu như khóa ngoại(forigen key) của bạn là FeaturedImages\_id. Nhưng không sao Laravel vẫn cung cấp cho chúng ta tùy biến cột khóa ngoại với cú pháp như sau:

public function FeaturedImages()

{

return $this->hasOne('App\FeaturedImages','forigen\_key');

}

-Nhưng nếu như ở bảng FeaturedImages của bạn cũng không phải liên kết với Post qua cột id thì bạn có thể khai báo thêm cho Larvel bằng cách sau:

public function FeaturedImages()

{

return $this->hasOne('App\FeaturedImages','forigen\_key','local\_key');

}

**-Định nghĩa các mối quan hệ đảo ngược.**

-Như ở phía trên mình có nói là một FearturedImages cũng chỉ thuộc về một bài post thôi, nên chúng ta hoàn toàn có thể khai báo định nghĩa đảo ngược lại với model FearturedImages như sau:

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class FeaturedImages extends Model

{

protected $table='FeaturedImages';

public $timestamp=false;

public function posts()

{

return $this->belongsto('App\Posts');

}

}

Cách gọi và truyền các khóa ngoại cho nó cũng tương tự như với model Posts.

public function posts()

{

return $this->belongsto('App\Posts','forigen\_key');

}

public function posts()

{

return $this->belongsto('App\Posts','forigen\_key','local\_key');

}

App\FearturedImages::find(1)->posts;

**3, One to Many.**

-Mối quan hệ này cũng tương tự như One to One nhưng nó chỉ khác nhau về câu lệnh thôi à.

-VD: Cũng với bảng Posts như ở VD trên nhưng lần này tôi sẽ tạo thêm một bảng Author , Nhưng ở đây nó sẽ là mối quan hệ một nhiều vì một Author thì có thể có nhiều bài posts và một bài post chỉ có một Author thôi.

-Mình sẽ tạo thêm model Author và phương thức **hasMany()** để tạo mối quan hệ cho nó.

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Author extends Model

{

protected $table='Author';

public $timestamp=false;

public function posts()

{

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

}

}

Và Larvel cũng cho chúng ta truyền khóa ngoại, cột liên kết như mối Quan hệ **One to One.**

public function posts()

{

return $this->hasMany('App\Posts','author\_id');

}

public function posts()

{

return $this->hasMany('App\Posts','author\_id','local\_key');

}

Để gọi phương thức cũng không khác gì.

App\author::find(1)->posts;

**-Định nghĩa các mối quan hệ đảo ngược.**

+Về phần này thì hoàn toàn giống phần **One to One** nên mình không viết lại nữa.

**4, Many to Many.**

-Mối quan hệ này nó hơi phức tạp hơn hai mối quan hệ kia một chút, nhưng không có gì là khó cả.

VD: Vẫn Là bảng posts như trên nhưng ta sẽ tạo thêm 2 bảng là posts\_category và category . Mối quan hệ được diễn tả như sau: Một bài posts thì sẽ thuộc một hoặc nhiều category và ngược lại một category cũng có một hoặc nhiều bài posts Và chúng được kết nối với nhau qua bảng posts\_category.

-Chúng ta sẽ thể hiện quan hệ giữa 2 bảng này bằng phương thức **belongstoMany()**như sau:

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Posts extends Model

{

protected $table='Posts';

public $timestamp=false;

public function category()

{

return $this->belongstoMany('App\category');

}

}

-Gọi phương thức cũng tương tự như 2 quan hệ trên.

App\Posts::find()->category;

**->Chú ý:**Ở đây bảng trung gian của các bạn không phải là bảng posts\_category (**quy ước mặc định của Laravel**) mà là một bảng khác thì các bạn phải tùy biến như sau:

public function category()

{

return $this->belongstoMany('App\category','table\_medium');

}

// table\_medium la ten bang trung gian cua cac ban

**Và** nếu hai column liên kết của bạn khác với quy ước của Laravel thì bạn có thể tùy biến như sau:

public function category()

{

return $this->belongstoMany('App\category','table\_medium','posts\_id','category\_id');

}

**-Định nghĩa các mối quan hệ đảo ngược.**

+Về phần này của quan hệ Many to Many thì nó lại giống hệt với phần chính của nó.

VD: chúng ta sẽ gọi phương thức posts trong category như sau.

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class category extends Model

{

protected $table='category';

public $timestamp=false;

public function posts ()

{

return $this->belongstoMany('App\Posts');

}

}

**-Truy cập vào Table trung gian qua quan hệ Many to Many.**

Như ở trên mình có nói là để sử dụng mối quan hệ Many to Many thì chúng ta cần phải giao tiếp qua một bảng trung gian. Và để truy cập vào bảng trung gian đó thì trong Laravel có cung cấp cho chúng ta một phương thức **pivot attribute.** VD: Tôi muốn xem mối quan hệ được xây dựng trên bảng trung gian posts\_category vào khi nào thì sẽ phải truy vấn như sau:

$post = App\Posts::find(1);

foreach ($post->category as $rows) {

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

}

**4, Has Many Through.**

- Ở mối quan hệ này chúng ta có thể cập vào các mối quan hệ xa thông qua một mối quan hệ trung gian.

Ví dụ, một Country model có thể có nhiều Post model thông qua một User model trung gian. Trong ví dụ này, bạn có thể dễ dàng lấy tất cả các blog post cho 1 country.

Hãy nhìn vào các bảng cần thiết để xác định mối quan hệ này:

countries

id - integer

name - string

users

id - integer

country\_id - integer

name - string

posts

id - integer

user\_id - integer

title - string

Mặc dù post không chứa cột country\_id, mối quan hệ hasManyThrough cung cấp quyền truy cập vào post của country thông qua $country->posts. Để thực hiện các truy vấn này, Eloquent kiểm tra các country\_id trên bảng user trung gian. Sau khi tìm ra id của user phù hợp, chúng được sử dụng để truy vấn bảng posts. Bây giờ chúng ta đã xem xét các cấu trúc bảng cho các mối quan hệ, hãy định nghĩa nó trên Country model.

Default

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Country extends Model

{

/\*\*

\* Get all of the posts for the country.

\*/

public function posts()

{

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

}

}

Đối số đầu tiên truyền cho phương thức hasManyThrough là tên của model cuối cùng chúng ta muốn truy cập, trong khi đối số thứ 2 là tên của model trung gian. Nếu bạn muốn tùy chỉnh các foreign key của relationship, bạn có thể truyền vào các đối số thứ 3 và thứ 4 của phương thức hasManyThrough. Đối số thứ 3 là foreign key của model trung gian, đối số thứ 4 là foreign key của model cuối cùng và đối số thứ 5 là local key.

Default

class Country extends Model

{

public function posts()

{

return $this->hasManyThrough(

'App\Post', 'App\User',

'country\_id', 'user\_id', 'id'

);

}

}

**5, Các mối quan hệ khác.**

-Ở hai mối quan hệ còn lại rất ít khi sử dụng lên mình sẽ không nói đến ở bài này, các bạn nào quan tâm có thể xem tại [đây](https://laravel.com/docs/5.3/eloquent-relationships#many-to-many-polymorphic-relations).