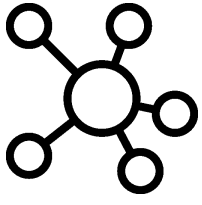
Relationships

Sidrit Trandafili

@sidis405

@strandafili

- Sviluppatore PHP 15+ anni
- Laravel da 4 anni
- Back-end
  - php
  - java
  - node
  - python
- Front-end
  - VueJS
  - Vanilla JS
  - TailwindCSS
  - Sass
- Devops + Sysops da 13 anni



Relationships

Relationships

> *Le tabelle del database sono spesso connesse tra di loro. Per esempio un post può avere più commenti, oppure un ordine può essere connesso all'utente che lo ha piazzato*

1. One to One
2. One to Many
3. Many to Many
4. HasManyThrough
5. Polimorfiche

Relationships

Setup

Prendiamo in considerazione i seguenti model

- User
- Profile
- Post
- Category
- Comment
- Tag
- Country
- Video

```
php artisan make:model User -m
```

```
php artisan make:model Profile -m
```

```
php artisan make:model Post -m
```

```
php artisan make:model Category -m
```

```
php artisan make:model Comment -m
```

```
php artisan make:model Tag -m
```

```
php artisan make:model Country -m
```

```
php artisan make:model Video -m
```

Relationships

One to One

Questa relazione è molto semplice. Per esempio un utente ha un solo profilo.

Per implementare questa relazione creiamo il metodo profile() nel modello User come segue:

```php
namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * Il profilo di questo utente.
     */
    public function profile()
    {
        return $this->hasOne(App\Profile::class);
    }
}
```

In questo modo Eloquent si aspetta di trovare il campo user_id nella tabella Profiles

Relationships

One to One

Il primo parametro della funzione è il model da associare. A questo punto possiamo reperire il modello connesso tramite Eloquent

```
$phone = User::find(1)->profile;
```

In verità il metodo hasOne riceve ben 3 parametri ma eloquent sta assumendo che le chiavi primarie delle tabelle siano 'id' e che le chiavi esterne seguano la convenzione '{nomeModello}_id'.

```
return $this->hasOne(App\Profile::class, '{chiave_id_esterna}', '{id_locale}');
```

Siamo liberi di sovrascrivere questa convenzione

```
return $this->hasOne(App\Profile::class,
    'id_utente_nella_tabella_profili',
    'id_primario_tabella_utenti'
    );
```

Relationships

One to One

Come dovremmo ragionare per la relazione inversa?

Come trovare l'istanza User di un oggetto Profile?

Creiamo il metodo user() nel modello Profile come segue:

```php
namespace App;

use Illuminate\Database\Eloquent\Model;

class Profile extends Model
{
    /**
     * L'utente di questo profilo
     */
    public function user()
    {
        return $this->belongsTo(App\User::class);
    }
}
```

In questo modo Eloquent si aspetta di trovare il campo user_id nella tabella Profiles

Relationships

One to One

Il primo parametro della funzione è il model da associare. A questo punto possiamo reperire il modello connesso tramite Eloquent

```
$user = Profile::find(1)->user;
```

In verità il metodo belongsTo riceve ben 3 parametri ma eloquent sta assumendo che le chiavi primarie delle tabelle siano 'id' e che le chiavi esterne seguano la convenzione '{nomeModello}_id'.

```
return $this->belongsTo(App\User::class, '{chiave_id_esterna}', '{id_locale}');
```

Siamo liberi di sovrascrivere questa convenzione

```
return $this->belongsTo(App\User::class,
    'id_utente_nella_tabella_profili',
    'id_primario_tabella_utenti'
    );
```

Relationships

One to Many

Questo tipo di relazione definisce i casi nei cui un modello è relazionato a più istanze di un altro modello. Per esempio: un User ha più istanze Post.

Creiamo il metodo posts() nel modello User come segue:

```
namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * I post di questo utente.
     */
    public function posts()
    {
        return $this->hasMany(App\Post::class);
    }
}
```

In questo modo Eloquent si aspetta di trovare il campo user_id nella tabella Posts

Relationships

One to Many

A questo punto possiamo reperire il modello connesso tramite Eloquent

```
$posts = User::find(1)->posts;

foreach($posts as $post){

    // $post->title

}
```

Il metodo hasOne riceve ben 3 parametri ma eloquent sta assumendo che le chiavi primarie delle tabelle siano 'id' e che le chiavi esterne seguano la convenzione '{nomeModello}_id'. Siamo liberi di sovrascrivere questa convenzione

```
return $this->hasMany(App\Post::class,
    'id_utente_nella_tabella_post',
    'id_primario_tabella_utenti'
    );
```

Relationships

One to Many

Come dovremmo ragionare per la relazione inversa?

Come trovare l'istanza User di un oggetto Post?

Creiamo il metodo user() nel modello Post come segue:

```php
namespace App;

use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    /**
     * L'utente di questo post
     */
    public function user()
    {
        return $this->belongsTo(App\User::class);
    }
}
```

In questo modo Eloquent si aspetta di trovare il campo user_id nella tabella Posts

Relationships

Many To Many

Un esempio di questa relazione è Post <-> Tag. Un post può avere più tag, e un Tag può essere associato a vari Post. Questo richiede una tabella intermedia (pivot) che contiene il post_id e il tag_id chiamata post_tag.

Creiamo il metodo tags() nel modello Post come segue:

```php
namespace App;

use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    /**
     * I tag di questo post.
     */
    public function tags()
    {
        return $this->belongsToMany(App\Tag::class);
    }
}
```

In questo modo Eloquent si aspetta di trovare i campi post_id tag_id nella tabella post_tag

Relationships

Many to Many

Il primo parametro della funzione è il model da associare. A questo punto possiamo reperire il modello connesso tramite Eloquent

```
$tags = Post::find(1)->tags;

foreach($tags as $tag){
    // $tag->name
}
```

L'inverso di una relazione

belongsToMany è sempre

belongsToMany ma con indici invertiti

belongsTo many riceve ben 4 parametri ma eloquent sta assumendo che le chiavi primarie delle tabelle siano 'id' e che le chiavi esterne seguano la convenzione '{nomeModello}_id'. Siamo liberi di sovrascrivere questa convenzione

```
return $this->belongsToMany(App\Tag::class,
    'nome_della_tablella_pivot',
    'id_post_nella_tabella_pivot',
    'id_tag_nella_tabella_pivot'
    );
```

Relationships

HasManyThrough

Abbiamo definito che un User ha più Post. Premettendo che un'istanza User ha un'istanza Country, come facciamo a reperire tutti i Post di quel Country?

Creiamo il metodo posts() nel modello Country come segue:

```php
namespace App;

use Illuminate\Database\Eloquent\Model;

class Country extends Model
{
    /**
     * L'utente di questo profilo
     */
    public function posts()
    {
        return $this->hasManyThrough('App\Post', 'App\User');
    }
}
```

In questo modo Eloquent si aspetta di trovare il campo user_id nella tabella Posts e country_id in Users. Definite tutte e due tramite hasOne o belongsTo.

Relationships

HasManyThrough

Il primo parametro della funzione è il model finale da associare. Il secondo è il model intermedio.

```
$posts = Country::find(1)->posts;

foreach($posts as $post){
    // $post->title
}
```

L'inverso di una relazione

hasManyThrough è sempre

hasManyThrough ma con indici invertiti

hasManyThrough riceve 6 parametri se se seguiamo le convenzioni Laravel, li possiamo omettere o volendo anche sovrascivere.

```
return $this->hasManyThrough(
        'App\Post',
        'App\User',
        'country_id', // Chiave esterna tabella users
        'user_id', // Chiave esterna tabella posts
        'id', // Chiave primaria tabella countries
        'id' // Chiave primaria tabella users
    );
```

Relationships

Polimorfiche

Dobbiamo implementare una funzionalità commenti. Decidiamo che bisogna creare un modello Comment, che appartiene a un Post definito con belongsTo() e il Post implementerà hasMany(Comment).

Poi riceviamo una richiesta che anche i Video devono avere Comment, e alla fine che anche i commenti devono essere commentabili.

Questo diviene ingestibile con quello che abbiamo imparato fino adesso, perché richiederebbe un PostComment, un VideoComment e un CommentComment. Il belongsToMany non funzionerebbe perché in verità stiamo parlando di 3 relazioni diverse.

Tutto diviene più complicato quando dobbiamo prendere i commenti che un utente ha fatto su post, video e anche come risposte di altri commenti.

Relationships

Polimorfiche

Prendiamo la seguente struttura

```
posts
    id - integer
    title - string
    body - text

videos
    id - integer
    title - string
    url - string

comments
    id - integer
    body - text
    commentable_id - integer
    commentable_type - string
```

I campi da osservare sono commentable_id e commentable_type.

commentable_id contiene l'id dell'istanza commentata, indipendentemente dal tipo (Post, Video o Comment).

commentable_type la classe dell'istanza commentata.

Relationships

Polimorfiche

Alla fine definiamo le relazioni nei modelli nel seguente modo:

```php
class Post extends Model
 {
     //Tutti i commenti per modelli Post
     public function comments()
     {
         return $this->morphMany('App\Comment',
                                 'commentable');
     }
 }
```

```php
class Comment extends Model
 {
     //Tutti i modelli commentati
     public function commentable()
     {
         return $this->morphTo();
     }
 }
```

```php
class Video extends Model
 {
     //Tutti i commenti per modelli Video
     public function comments()
     {
         return $this->morphMany('App\Comment',
                                 'commentable');
     }
 }
```

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

foreach ($post->comments as $comment) {
    // $comment->body
}
```