Les relations Laravel Eloquent

[sommaire]

1. ## Objectifs

- Etre capable d'utiliser les relations des modèles avec Eloquent Laravel

- ## Présentation

  - Pour les bases de données qui fonctionnent sur la base de relations, vous devez définir la structure de votre base de données avant d'y insérer des données.
  - MySQL le plus connu au monde est une base de données relationnelle. Les autres SGBDR célèbres sont MSSQL, Oracle, Microsoft Access et SQLite. Dans ces bases de données, une collection de données similaires est appelée une table.
  - La clé primaire d'une table est la clé étrangère d'une autre table sont reliées pour faire la relation entre ces deux tanles.
  - Il existe trois types de base de relations **un-à-un**, **un-à-plusieurs** et **plusieurs-à-plusieurs**.
  - Dans PHP personnalisé ou dans des Frameworks qui fonctionnent sur la base du modèle Active Record, vous devez gérer les relations à l'aide de jointures tandis que dans Laravel, vous avez également cette option, mais vous pouvez gérer toutes ces relations de manière très simple en utilisant des modèles éloquents .

- ## Relation un à un

  - Dans une relation **un-à-un**, deux tables sont liées de telle sorte qu'un enregistrement d'une table ne peut avoir qu'un seul enregistrement dans l'autre table.
  - Une relation **un à un** est un type très basique de relation de base de données. Par exemple, un modèle **User** peut être associé à un modèle **Phone**.

    > *Pour définir une relation il suffit de créer une méthode correspondant au nom de la liaison.*

  - Pour définir cette relation, nous placerons une méthode **phone** sur le modèle **User**.
  - La méthode **phone** doit appeler la méthode `hasOne` et renvoyer son résultat.
  - La méthode `hasOne` est disponible pour votre modèle via la classe de base du modèle `Illuminate\Database\Eloquent\Model` :

apcpedagogie.com

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * Get the phone associated with the user.
     */
    public function phone()
    {
        //return $this->hasOne(Phone::class);
        return $this->hasOne('App\Phone');
    }
}
```

- **Définir l'inverse de la relation**
    - Ainsi, nous pouvons accéder au modèle **Phone** à partir de notre modèle **User**. Ensuite, définissons une relation sur le modèle **Phone** qui nous permettra d'accéder à l'utilisateur propriétaire du téléphone. Nous pouvons définir l'inverse d'une relation `hasOne` en utilisant la méthode: `belongsTo`

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Phone extends Model
{
    /**
     * Get the user that owns the phone.
     */
    public function user()
    {
        //return $this->belongsTo(User::class);
        return $this->belongsTo('App\User');
    }
}
```

apcpedagogie.com

- Lors de l'appel de la méthode **user**, Eloquent tentera de trouver un modèle **User** qui a un **id** correspondant à la colonne **user_id** du modèle **Phone**.

- ## Relation un à plusieurs

    - Dans une relation à plusieurs entre deux tables dans la base de données, un enregistrement de table peut avoir de nombreux enregistrements liés dans une autre table,
    - Une relation **un-à-plusieurs** est utilisée pour définir des relations dans lesquelles un modèle unique est le parent d'un ou plusieurs modèles enfants.
    - Par exemple, un article de blog peut contenir un nombre infini de commentaires. Comme toutes les autres relations Eloquent, les relations un-à-plusieurs sont définies en définissant une méthode sur votre modèle Eloquent:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    /**
     * Get the comments for the blog post.
     */
    public function comments()
    {
        //return $this->hasMany(Comment::class);
        return $this->hasMany('App\Posts');
    }
}
```

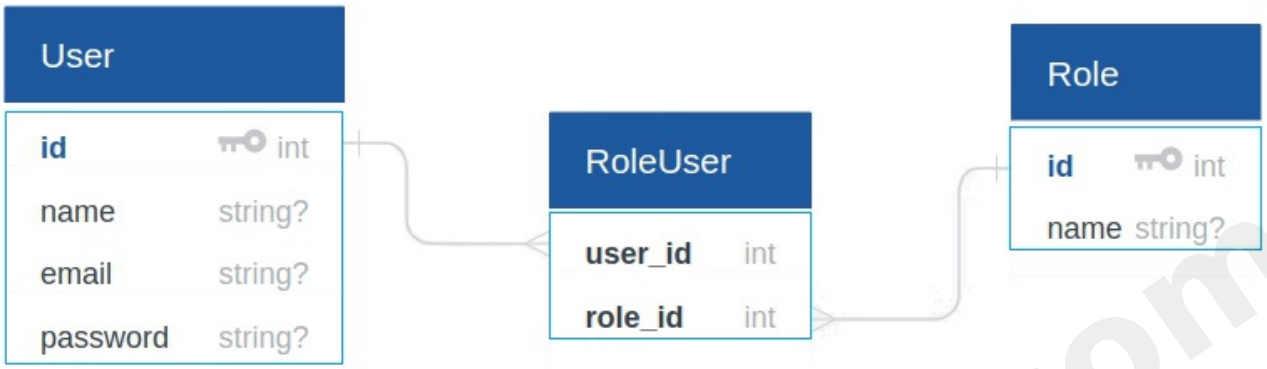
- **Un à plusieurs (inversé)/appartient à**

    - Maintenant que nous pouvons accéder à tous les commentaires d'un article, définissons une relation pour permettre à un commentaire d'accéder à son article parent.
    - Pour définir l'inverse d'une relation `hasMany`, définissez une méthode de relation sur le modèle enfant qui appelle la méthode `belongsTo`
    - Voici une schématisation de la relation avec les deux méthodes :

apcpedagogie.com

- ## Relation plusieurs à plusieurs (many-to-many)

  - Les relations **n..n (many-to-many)** permettent de définir une relation entre plusieurs objets d'un côté et plusieurs objets de l'autre.



  - C'est le cas par exemple dans les relations entre des utilisateurs et des rôles, ou des articles et des catégories. Un utilisateur peut avoir plusieurs rôles ("admin" et "modérateur") et un rôle peut être donné à plusieurs utilisateurs ("Donovan" ou "Vincent"). Ou un article peut avoir plusieurs catégories ("programmation" et "bien-être") et une même catégorie peut avoir plusieurs articles ("Le coût des interruptions").
  - Les exemples suivants seront basés sur une relation entre articles (à travers un modèle Article) et catégories (avec un modèle Category).

1. **Création des modèles**

  - Deux appels à Artisan permettent de créer les modèles et leurs migrations.

```
//Création des modèles
php artisan make:model Article --migration
php artisan make:model Category --migration
```

- **Structure des tables**

  - Les relations **n..n** sont légèrement plus compliquées à représenter en base que les relations **1..1** et **1..n**. En effet, elles nécessitent une table "pivot" qui contient une référence aux clés primaires des deux autres tables.
  - Ainsi, il faudra donc créer une table articles, une table categories et une table article_category. Cette dernière contiendra une liste des identifiants aux deux autres tables sous la forme de clés étrangères.

apcpedagogie.com

- Si l'article d'identifiant 12 appartient aux catégories 35 et 80, alors la table pivot contiendra deux entrées avec les couples 12-35 et 12-80. De la même manière, si la catégorie 44 contient deux articles d'identifiants 22 et 33, alors la table pivot contiendra deux entrées 22-44 et 33-44.
- Il faut donc au préalable créer également une migration pour la table article_category.

```
//Création de la table de pivot
php artisan make:migration create_table_article_category
```

> *Par convention, le nom de la table pivot est composé des noms des deux tables à relier au singulier dans l'ordre alphabétique. Cela permet à Laravel de faire directement le lien entre les tables sans avoir à préciser le nom de la table de relation. Donc, pour une table articles et une table categories, il faudra créer une table pivot article_category.*

- La table pivot contient uniquement deux colonnes qui sont deux clés étrangères vers les tables qui sont mises en relation. Éditez le fichier de migration pour définir le schéma du pivot.
- **Structure de la table `article_category`**

```
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Migrations\Migration;

class CreateTableArticleCategory extends Migration
{
    /**
     * Lancement des migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('article_category', function ($table) {

            $table->integer('article_id')
                  ->unsigned()
                  ->index();
            $table->foreign('article_id')
                  ->references('id')
                  ->on('articles')
                  ->onDelete('cascade');
```

apcpedagogie.com

```
            $table->integer('category_id')
                  ->unsigned()
                  ->index();
            $table->foreign('category_id')
                  ->references('id')
                  ->on('categories')
                  ->onDelete('cascade');

            $table->primary(['article_id', 'category_id']);
        });
    }

    /**
     * Annulation de la migration.
     *
     * @return void
     */
    public function down()
    {
        Schema::drop('article_category');
    }
}
```

- **Structure de la table `articles`**

```
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Migrations\Migration;

class CreateArticlesTable extends Migration
{
    /**
     * Lancement de la migration.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('articles', function ($table) {
            $table->increments('id');
            $table->string('title')->index();
```

apcpedagogie.com

```
            $table->text('text');
            $table->timestamps();
        });
    }

    /**
     * Annulation de la migration.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('articles');
    }
}
```

- **Structure de la table `categories`**

```
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Migrations\Migration;

class CreateCategoriesTable extends Migration
{
    /**
     * Lancement de la migration.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('categories', function ($table) {
            $table->increments('id');
            $table->string('title')->index();
            $table->timestamps();
        });
    }

    /**
     * Annulation de la migration.
     *
     * @return void
```

apcpedagogie.com

```
     */
    public function down()
    {
        Schema::dropIfExists('categories');
    }
}
```

- Lancez ensuite les migrations avec Artisan pour créer les trois tables en base de données: `php artisan migrate`

- **Création des modèles**

    - Pour déclarer les relations n..n dans les modèles, il faut ajouter une méthode articles dans la classe Category et une méthode categories dans la classe Article.
    - Les noms de méthodes sont au pluriel dans les deux sens dans le cas d'une relation n..n. En effet, un article appartient potentiellement à plusieurs catégories et une catégorie peut également appartenir à plusieurs articles.
    - Déclarez dans le corps de chacune de ces méthodes qu'un article appartient à plusieurs catégories (**Article belongsToMany Category**) et qu'une catégorie appartient à plusieurs articles (**Category belongsToMany Article**). Contrairement aux autres relations, les deux relations sont en miroir, c'est donc la même méthode **belongsToMany** qui est utilisée dans les deux classes de modèle. On dit que les deux modèles peuvent "appartenir à plusieurs" instances de modèles.

      Classe Article

```
<?php
namespace App;

use Illuminate\Database\Eloquent\Model;

class Article extends Model
{
    public function categories()
    {
        // Un article appartient à une ou plusieurs catégories.
        return $this->belongsToMany(Category::class);
    }
}
```

Classe Category

apcpedagogie.com

```php
<?php
namespace App;

use Illuminate\Database\Eloquent\Model;

class Category extends Model
{
    public function articles()
    {
        // Une catégorie appartient à un ou plusieurs articles.
        return $this->belongsToMany(Article::class);
    }
}
```

- Source:
- Document officielle Lravel
- Laravel - Un framework efficace pour développer vos applications PHP

Cours Laravel

Source : https://apcpedagogie.com/les-relations-laravel-eloquent · téléchargé le 03/11/2026 · © APC Pédagogie · apcpedagogie.com

apcpedagogie.com