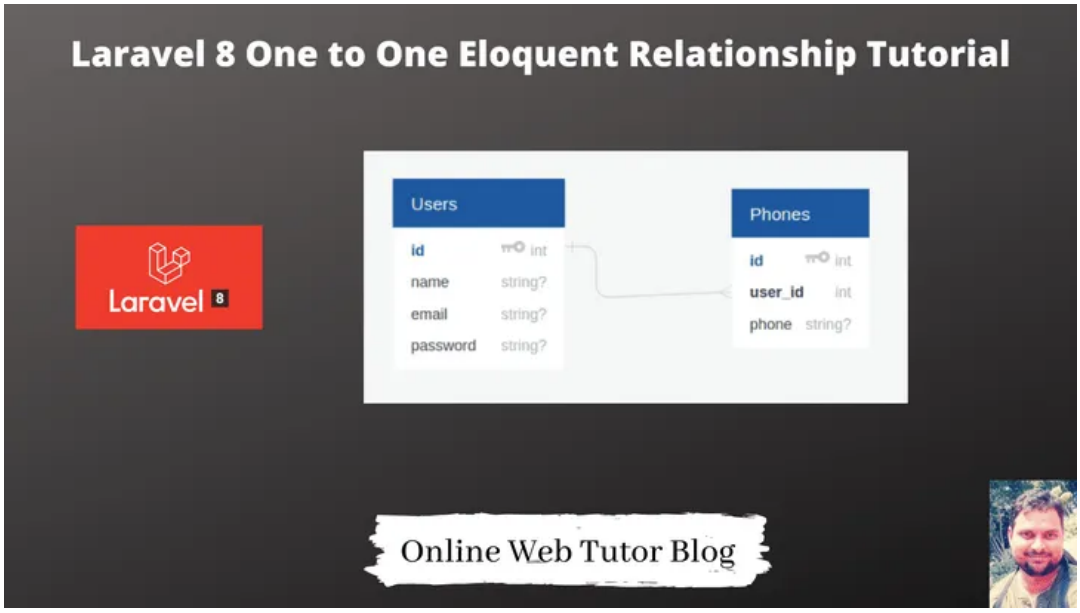

Laravel 8 Tutoriel sur les relations éloquentes de Laravel 8

31 mars 2021 par Sanjay Kumar

Table des matières

Partagez cet article

La relation éloquente de Laravel est une fonctionnalité très importante qui relie une ou plusieurs tables dans une chaîne. C'est le substitut des jointures dans laravel.

Laravel fournit ces relations suivantes -

- **Un par un**
- **Un à plusieurs**
- **Plusieurs à plusieurs**
- Un à plusieurs (inverse) / Appartient à
- **A un travers**
- **A beaucoup à travers**

Les relations Eloquent sont définies comme des méthodes sur vos classes de modèle Eloquent. Dans cet article, nous verrons le concept de relation laravel 8 One to One Eloquent ainsi que nous mettrons en œuvre l'inverse de la relation un à un, c'est-à-dire appartient à.

Cet article vous donnera le concept détaillé de la mise en œuvre d'une relation un à un dans laravel.

Commençons.

Installation de l'application Laravel

L'installation de Laravel peut se faire de deux manières.

- Installateur Laravel
- En utilisant le compositeur

Installateur Laravel

Pour installer Laravel via le programme d'installation de Laravel, nous devons d'abord installer son programme d'installation. Nous devons utiliser le compositeur pour cela.

```
$ composer global nécessite laravel/installer
```

Cette commande installera le programme d'installation de laravel sur le système. Cette installation est à portée globale, vous tapez donc la commande à partir de n'importe quel répertoire du terminal. Pour vérifier, tapez la commande donnée -

```
$ laravel
```

Cette commande ouvrira une palette de commandes de Laravel Installer.

Pour créer un projet ad install laravel dans le système,

```
$ laravel nouveau blog
```

En utilisant le compositeur

Alternativement, nous pouvons également installer Laravel par la commande Composer **create-project** .

If your system doesn't has composer Installed, **Learn Composer Installation Steps**.

Voici la commande complète pour créer un projet laravel-

```
$ composer create-project --prefer-dist blog laravel/laravel
```

Après avoir suivi ces étapes, nous pouvons installer une application Laravel dans le système.

Pour démarrer le serveur de développement de Laravel –

```
$ php service artisanal
```

Cette commande affiche –

Démarrage du serveur de développement Laravel : http://127.0.0.1:8000

En supposant que laravel est déjà installé sur le système.

Créer une base de données et se connecter

Pour créer une base de données, soit nous pouvons créer via l'outil manuel de PhpMyadmin soit au moyen d'une commande mysql.

```
CRÉER UNE BASE DE DONNÉES laravel_app ;
```

Pour connecter la base de données à l'application, **ouvrez le fichier .env à** partir de la racine de l'application. Recherchez **DB_** et mettez à jour vos informations.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel_app
DB_USERNAME=racine
DB_PASSWORD=racine
```

Créer des migrations

Nous avons besoin de deux fichiers de migration. L'un est pour la table des **utilisateurs** et l'autre pour la table des **téléphones** . Par défaut, lorsque nous installons laravel, nous obtiendrons la table de migration pour les utilisateurs.

Nous trouverons la migration **2014_10_12_000000_create_users_table.php** des utilisateurs dans /database/migrations.

Ouvrez le fichier de migration, nous devrions y voir le code suivant.

```
   #Utilisateurs
< ? php

utilisez  Illuminate\Database\Migrations\Migration ;
utilisez  Illuminate\Database\Schema\Blueprint ;
utilisez  Illuminate\Support\Facades\Schema ;

la classe  CreateUsersTable  étend la  migration
{
    /**
     * Exécutez les migrations.
     *
     * @retour vide
     */
     fonction  publique ( )
    {
        Schéma :: créer ( 'utilisateurs' , fonction ( Blueprint  $table ) {
            $table -> identifiant ();
            $table -> chaîne ( 'nom' );
            $table -> string ( 'email' ) -> unique ();
            $table -> horodatage ( 'email_verified_at' ) -> nullable ();
            $table -> chaîne ( 'mot de passe' );
            $table -> RememberToken ();
            $table -> horodatages ();
        });
    }

    /**
     * Inverser les migrations.
     *
     * @retour vide
     */
     fonction  publique vers le bas ()
    {
        Schéma :: dropIfExists ( 'utilisateurs' );
    }
}
```

Ouvrez le projet dans le terminal et exécutez cette commande spark pour **créer une** table de migration pour les téléphones.

```
$ php artisan make:migration CreatePhonesTable
```

Il créera un fichier 2021_03_31_170942_create_phones_table.php à **/database/migrations** selon la valeur d'horodatage.

```
#Téléphone (s
< ? php

utilisez  Illuminate\Database\Migrations\Migration ;
utilisez  Illuminate\Database\Schema\Blueprint ;
utilisez  Illuminate\Support\Facades\Schema ;

la classe  CreatePhonesTable  étend la  migration
{
    /**
     * Exécutez les migrations.
     *
     * @retour vide
     */
     fonction  publique ( )
    {
        Schema :: create ( 'phones' , function ( Blueprint  $table ) {
            $table -> identifiant ();
            $table -> entier ( 'user_id' ) -> non signé ();
            $table -> chaîne ( 'téléphone' , 30 );
            $table -> étranger ( 'user_id' )
                -> références ( 'id' ) -> on ( 'users' )
                -> onDelete ( 'cascade' );
        });
    }

    /**
     * Inverser les migrations.
     *
     * @retour vide
     */
     fonction  publique vers le bas ()
    {
        Schéma :: dropIfExists ( 'téléphones' );
    }
}
```

Exécuter les migrations

Ensuite, nous devons créer des tables dans la base de données.

```
$ php artisan migrer
```

This command will create tables inside database.

Create Model

We need to create few models inside application. By default **User.php** is available inside application setup.

Menu

User.php

```php
<?php

namespace App\Models;

use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use App\Models\Phone;

class User extends Authenticatable
{
    use HasFactory, Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name',
        'email',
        'password',
    ];

    /**
     * The attributes that should be hidden for arrays.
     *
     * @var array
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];

    /**
     * The attributes that should be cast to native types.
     *
     * @var array
     */
    protected $casts = [
        'email_verified_at' => 'datetime',
    ];

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

We will create **Phone.php** model.

Run this command to create model –

```
$ php artisan make:model Phone
```

It will create **Phone.php** file at /app/Models folder. Open file and write this code.

Phone.php

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use App\Models\User;

class Phone extends Model
{
    use HasFactory;

    public $timestamps = false;

    protected $fillable = [
        'user_id',
        'phone'
    ];

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

belongsTo() method is implementing inverse relationship of one to one relationship in laravel.

Create Seeder with One to One Relationship

Open project into terminal and type these artisan command.

```
$ php artisan make:factory UserFactory --model=User

$ php artisan make:factory PhoneFactory --model=Phone
```

It will create two files, **UserFactory.php** and **PhoneFactory.php** at /database/factories folder.

Open **UserFactory.php** file and write this code into it.

UserFactory.php

```php
<?php

namespace Database\Factories;

use App\Models\User;
use Illuminate\Database\Eloquent\Factories\Factory;
use Illuminate\Support\Str;

class UserFactory extends Factory
{
    /**
     * The name of the factory's corresponding model.
     *
     * @var string
     */
    protected $model = User::class;

    /**
     * Define the model's default state.
     *
     * @return array
     */
    public function definition()
    {
        return [
            'name' => $this->faker->name,
            'email' => $this->faker->unique()->safeEmail,
            'email_verified_at' => now(),
            'password' => bcrypt("123456"), // password
            'remember_token' => Str::random(10),
        ];
    }
}
```

PhoneFactory.php

```php
<?php

namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;
use App\Models\User;
use App\Models\Phone;

class PhoneFactory extends Factory
{
    /**
     * The name of the factory's corresponding model.
     *
     * @var string
     */
    protected $model = Phone::class;

    /**
     * Define the model's default state.
     *
     * @return array
     */
    public function definition()
    {
        return [
            "user_id" => \App\Models\User::factory()->create()->id,
            "phone" => $this->faker->phoneNumber
        ];
    }
}
```

Here, we are generating data using One to One Eloquent relationship.

Open terminal and type

```
$ php artisan tinker
```

Inside tinker shell panel, run this command to seed dummy data into database table.

```
>>> App\Models\Phone::factory()->count(10)->create()
```

This command will seed data into users table and phones table as well. In both tables we will have 10 number of records.

Calling Methods at Controller

Open any controller say **SiteController.php** file, we have created two methods in which we used model methods as a property.

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;
utilisez  App\Models\User ;
utilisez  App\Models\Phone ;

la classe  SiteController  étend le  contrôleur
{
     fonction  publique getPhone ( $user_id )
    {
        // Passage de l'identifiant de l'utilisateur dans find()
        return  User :: find ( $user_id ) -> phone ;
    }

     fonction  publique getUser ( $phone_id )
    {
        // Passage de l'identifiant du téléphone dans find()
        return  Phone :: find ( $phone_id ) -> user ;
    }
}
```

- **Utilisateur :: find($user_id)->téléphone ;** Il trouvera la valeur des détails du téléphone par identifiant d'utilisateur. **Un par un**
- **Téléphone :: find($phone_id)->utilisateur ;** Il trouvera les détails de l'utilisateur par identifiant de téléphone. **Inverse de Un à Un / Appartient à**

Créer des itinéraires

Ouvrez **web.php** à partir du dossier /routes et ajoutez-y ces routes.

web.php

```
# Ajouter ceci à l'en-tête
utilisez  App\Http\Controllers\SiteController ;

Route :: get ( 'get-phone/{id}' , [ SiteController :: class , 'getPhone' ]);
Route :: get ( 'get-user/{id}' , [ SiteController :: class , 'getUser' ]);
```

Test d'applications

Ouvrez le projet sur le terminal et tapez la commande pour démarrer le serveur de développement

```
$ php service artisanal
```

Obtenir les détails du téléphone - http://127.0.0.1:8000/get-phone/1

Obtenir les détails de l'utilisateur - http://127.0.0.1:8000/get-user/1

Si vous avez aimé cet article, veuillez vous abonner à notre **chaîne YouTube** pour PHP et ses tutoriels vidéo sur le framework, WordPress, Node Js. Vous pouvez également nous retrouver sur **Twitter** et **Facebook** .

En savoir plus sur les articles de Laravel 8 ici

- Comment créer un site Web multilingue dans Laravel 8
- Comment lire un fichier XML dans Laravel 8 - Exemple
- Comment télécharger et enregistrer des données XML dans Laravel 8
- Tutoriel de demande de publication Laravel 8 Ajax
- Authentification Laravel 8 à l'aide de Jetstream avec Inertia Js
- Authentification Laravel 8 à l'aide de Jetstream avec Livewire
- Tutoriel d'authentification Laravel 8 avec Breeze
- Laravel 8 Effacer le cache de la route, de la vue et de la configuration
- Tutoriel de planification des tâches de la tâche Laravel 8 Cron
- Laravel 8 DataTable Ajax Pagination avec recherche et tri
- Tutoriel de notification push de Laravel 8 Firebase
- Méthodes de validation des formulaires Laravel 8
- Guide d'installation de Laravel 8 – Framework PHP
- Guide complet des mises en page et des vues de Laravel 8
- Tutoriel de routage Laravel 8 Guide étape par étape
- Laravel 8 Envoyer un courrier à l'aide du serveur SMTP Gmail
- Laravel 8 Envoyer une notification push à Android à l'aide de Firebase
- Laravel 8 Envoyer une notification push à IOS à l'aide de Firebase
- Personnalisation du Stub Laravel 8

Articles Similaires:

- Tutoriel sur les relations éloquentes Laravel 8 One to Many
- Laravel 8 a un tutoriel sur les relations éloquentes
- Laravel 8 en a beaucoup grâce au didacticiel sur les relations éloquentes
- Laravel 8 Tutoriel sur les relations éloquentes de plusieurs à plusieurs
- Tutoriel sur les mutateurs et accesseurs éloquents dans Laravel 8
- Journal de magasin Laravel 8 des requêtes SQL éloquentes

Laravel 8
‹ Téléchargement de plusieurs images dans le didacticiel CodeIgniter 4
› Tutoriel sur les relations éloquentes Laravel 8 One to Many

Retrouvez-nous sur Youtube

Tuteur Web en ligne Youtube

Publicité

Articles populaires

Pardon. Pas de données jusqu'à présent.

Catégories

Messages récents

Menu

Menu

Menu

Menu

Menu