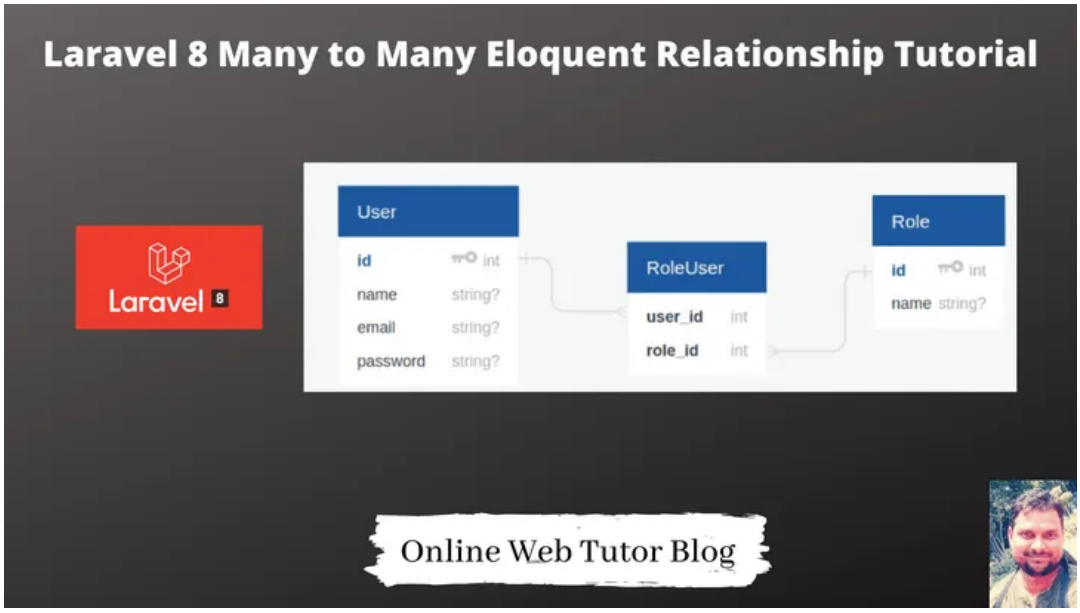

Laravel 8 Many to Many Eloquent Relationship Tutorial

3 avril 2021 par Sanjay Kumar

Table des matières

1. Installation de l'application Laravel
2. Créer une base de données et se connecter
3. Créer des migrations
4. Créer un modèle
5. Exemple d'insertion de données
6. Données pour le tableau des rôles
7. Données pour le tableau des utilisateurs de rôle
8. Méthodes d'appel au contrôleur
9. Créer des itinéraires
10. Test d'applications

Reading Time: 10 minutes 13 849 Vues

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 Many to Many Eloquent ainsi que nous mettrons en œuvre l'inverse de la relation plusieurs à plusieurs, c'est-à-dire appartient à plusieurs.

Cet article vous donnera le concept détaillé de la mise en œuvre de la relation plusieurs à plusieurs 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,

Avec le nom du **blog** un projet laravel sera créé à votre chemin spécifié.

En utilisant le compositeur

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

Si votre système n'a pas installé composer, **Apprenez les étapes d'installation de Composer** .

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 ;
```

```
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 quelques fichiers de migration -

- Migration des utilisateurs pour stocker les données des utilisateurs
- Migration des rôles vers les rôles de magasin
- Migration d'utilisateur de rôle pour stocker l'ID utilisateur et l'ID de rôle

Ouvrez le projet dans le terminal et exécutez ces commandes artisanales.

```
$ php artisan make:migration CreateRolesTable

$ php artisan make:migration CreateRoleUserTable
```

Il créera deux fichiers de migration 2021_04_03_042735 **_create_roles_table.php** et 2021_04_03_042802 **_create_role_user_table.php** à l'emplacement **/database/migrations** .

Nous avons déjà la migration 2014_10_12_000000_create_users_table.php disponible par défaut qui est pour la table des utilisateurs.

Ouvrez 2021_04_03_042735 **_create_roles_table.php** et écrivez-y ce code complet.

```
   #RoleMigration
< ? php

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

la classe  CreateRolesTable  étend la  migration
{
    /**
     * Exécutez les migrations.
     *
     * @retour vide
     */
     fonction  publique ( )
    {
        Schema :: create ( 'roles' , function ( Blueprint  $table ) {
            $table -> identifiant ();
            $table -> chaîne ( 'nom' , 25 );
        });
    }
```

Menu

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

Ouvrez 2021_04_03_042802 **_create_role_user_table.php** et écrivez-y ce code complet.

```
   #RoleUserTable


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

se  CreateRoleUserTable  étend la  migration


Exécutez les migrations.

@retour vide

nction  publique ( )

 Schéma :: créer ( 'role_user' , function ( Blueprint  $table ) {
     $table -> identifiant ();
     $table -> entier non signé ( 'id_utilisateur' );
     $table -> entier non signé ( 'role_id' );

     $table -> étranger ( 'user_id' ) -> références ( 'id' ) -> on ( 'users' ) -> onDelete ( 'cascade' )

     $table -> étranger ( 'role_id' ) -> références ( 'id' ) -> on ( 'roles' ) -> onDelete ( 'cascade' )
 });



Inverser les migrations.

@retour vide

nction  publique vers le bas ()

 Schéma :: dropIfExists ( 'role_user' );
```

Aussi, si nous ouvrons **la migration des utilisateurs** , nous devrions aimer cela.

```
#UserMigration
< ? 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->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('users');
    }
}
```

Run Migrations

Next, we need to create tables inside database.

```
$ php artisan migrate
```

This command will create tables inside database.

Create Model

commands.

```
$ php artisan make:model Role

$ php artisan make:model RoleUser
```

These commands will create two files **Role.php** & **RoleUser.php** at /app/Models folder.

Open **Role.php** and write this complete code into it.

Role.php

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Role extends Model
{
    use HasFactory;

    protected $fillable = ["name"];

    public $timestamps = false;

    /**
     * The users that belong to the role.
     */
    public function users()
    {
        return $this->belongsToMany(User::class, 'role_user');
    }
}
```

Open **RoleUser.php** and write this complete code into it.

RoleUser.php

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class RoleUser extends Model
{
    use HasFactory;

    protected $table = "role_user";

    protected $fillable = ["user_id"];

    public $timestamps = false;
```

Also If we open **User model,** copy and paste this complete code into it.

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\Role;
//use App\Models\RoleUser;

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
     */
```

```
    ];

    /**
     * The roles that belong to the user.
     */
    public function roles()
    {
        return $this->belongsToMany(Role::class, 'role_user');
        // "role_user" is table name
        // OR if we have model RoleUser, then we can use class
        // instead of table name role_user
        //return $this->belongsToMany(Role::class, RoleUser::class);
    }
}
```

- **$this->belongsToMany(Role::class, 'role_user');** It indicates many to many relationship use role_user table
- If we want to use class instead of table name in the above method, use like this **$this->belongsToMany(Role::class, RoleUser::class);**

Sample Data Insertion

Open mysql and run these queries to insert dummy data into posts and comments table.

Data for Users Table

```
Users Table

--
-- Dumping data for table `users`
--

INSERT INTO `users` (`id`, `name`, `email`, `email_verified_at`, `password`, `remember_token`, `created
(1, 'Emilie Mante', 'kwiegand@example.com', '2021-04-02 23:13:37', '$2y$10$92IXUNpkjO0rOQ5byMi.Ye4oKoEa
(2, 'Reyna Stroman II', 'schuster.carlos@example.com', '2021-04-02 23:13:37', '$2y$10$92IXUNpkjO0rOQ5by
(3, 'Prof. Lauriane Yost I', 'greta69@example.com', '2021-04-02 23:13:37', '$2y$10$92IXUNpkjO0rOQ5byMi.
(4, 'Zelma Yundt', 'ricardo.cartwright@example.com', '2021-04-02 23:13:37', '$2y$10$92IXUNpkjO0rOQ5byMi
(5, 'Hayley Lebsack', 'elna.tillman@example.com', '2021-04-02 23:13:37', '$2y$10$92IXUNpkjO0rOQ5byMi.Ye
```

Data for Roles Table

```
Roles Table

--
-- Dumping data for table `roles`
--

INSERT INTO `roles` (`id`, `name`) VALUES
(1, 'Admin'),
(2, 'Editor'),
(3, 'Author'),
(4, 'Blogger');
```

Role User Table

```
INSERT INTO `role_user` (`id`, `user_id`, `role_id`) VALUES
(1, 1, 1),
(2, 1, 3),
(3, 1, 4),
(4, 2, 3),
(5, 2, 4),
(6, 3, 3),
(7, 4, 2);
```

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.

#Controller

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\User;
use App\Models\Role;

class SiteController extends Controller
{
    // To get all roles of a user
    public function getRoles($user_id)
    {
        return User::find($user_id)->roles;
    }

    // To get all users by role
    public function getUsers($role_id)
    {
        return Role::find($role_id)->users;
    }
}
```

- **$roles = User::find($user_id)->roles;** It will find all roles on the basis of user id. **Many to Many**
- **$users = Role::find($role_id)->users;** It will find all users by role id. **Inverse of Many to Many / Belongs To**

Create Routes

Open **web.php** from /routes folder and add these routes into it.

web.php

```
# Add this to header
```

```
Route::get('get-users/{id}', [SiteController::class, 'getUsers']);
Route::get('get-roles/{id}', [SiteController::class, 'getRoles']);
```

Application Testing

Open project to terminal and type the command to start development server

```
$ php artisan serve
```

Get Users by user id – http://127.0.0.1:8000/get-users/1

Get Roles by user id– http://127.0.0.1:8000/get-roles/1

We hope this article helped you to learn about Laravel 8 Many to Many Eloquent Relationship Tutorial in a very detailed way.

If you liked this article, then please subscribe to our **YouTube Channel** for PHP & it's framework, WordPress, Node Js video tutorials. You can also find us on **Twitter** and **Facebook**.

Find More on Laravel 8 Articles here

- How to Create Multi Language Website in Laravel 8
- How To Read XML File in Laravel 8 – Example
- How To Upload And Save XML Data in Laravel 8
- Laravel 8 Ajax Post Request Tutorial
- Laravel 8 Authentication using Jetstream with Inertia Js
- Laravel 8 Authentication using Jetstream with Livewire
- Laravel 8 Authentication with Breeze Tutorial
- Laravel 8 Clear Cache of Route, View & Config
- Laravel 8 Cron Job Task Scheduling Tutorial
- Laravel 8 DataTable Ajax Pagination with Search And Sort
- Laravel 8 Firebase Push Notification Tutorial
- Laravel 8 Form Validation Methods
- Laravel 8 Installation Guide – PHP Framework
- 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:

- Laravel 8 en a beaucoup grâce au didacticiel sur les relations éloquentes
- Laravel 8 One to Many Eloquent Relationship Tutorial

- Tutoriel sur les mutateurs et accesseurs éloquents dans Laravel 8
- Journal de magasin Laravel 8 des requêtes SQL éloquentes

Laravel 8
Tutoriel sur les relations éloquentes Laravel 8 One to Many
Tutoriel sur le paquet de générateur de plan de site Laravel 8

Retrouvez-nous sur Youtube

Tuteur Web en ligne Youtube

Articles populaires

Pardon. Pas de données jusqu'à présent.

Catégories

Bloguer (1)

GâteauPHP 4 (11)

CodeIgniter 4 (156)

Éducation (1)

jQuery et Javascript (39)

Laravel 8 (152)

MySQL (dix)

Noeud J (15)

Divers PHP (33)

Messages récents

Comment lire des données XML en JSON dans CodeIgniter 4

Comment lire des données XML en JSON dans le didacticiel PHP

Somme MySQL avec le tutoriel de condition If

Quels sont les composants fonctionnels de CodeIgniter 4 ?

Concept d'image de chargement paresseux à l'aide de jQuery