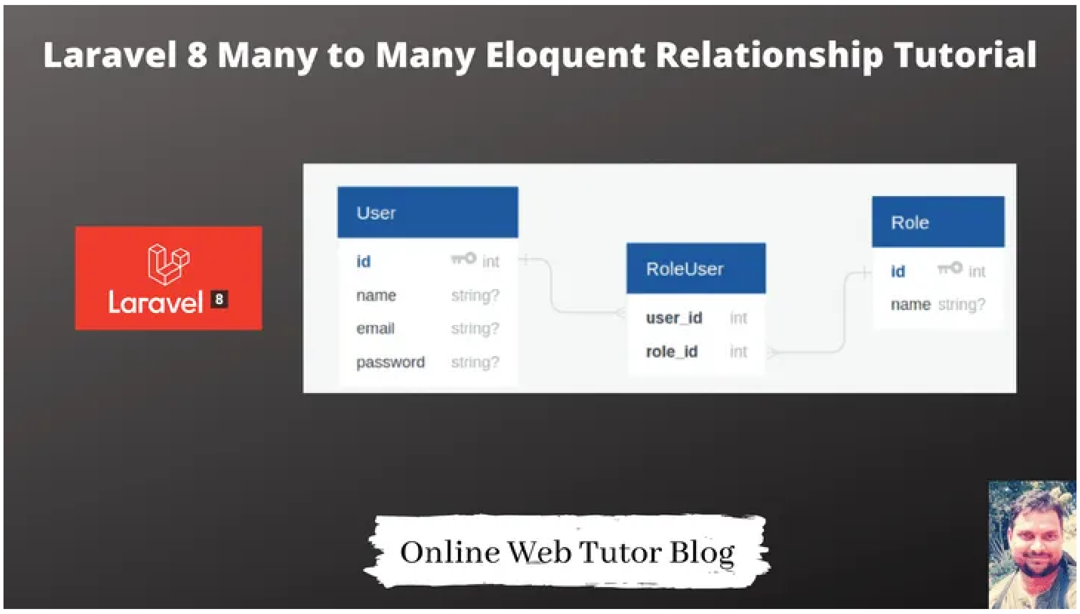

Laravel 8 Many to Many Eloquent Relationship Tutorial

April 3, 2021 by Sanjay Kumar

Table of Contents

Menu

Reading Time: 10 minutes 13,849 Views

Laravel eloquent relationship is a very important feature which connects one or more tables in a chain. This is the substitute of joins in laravel.

Laravel provides these following relationships –

- **One To One**
- **One To Many**
- **Many to Many**
- One To Many (Inverse) / Belongs To
- **Has One Through**
- **Has Many Through**

Eloquent relationships are defined as methods on your Eloquent model classes. Inside this article we will see the concept of laravel 8 Many to Many Eloquent relationship as well as we will implement inverse of many to many relationship i.e belongsToMany.

This article will give you the detailed concept of about implementation of many to many relationship in laravel.

Let’s get started.

Installation of Laravel Application

Laravel Installation can be done in two ways.

- Laravel Installer
- By using composer

Laravel Installer

To install Laravel via Laravel installer, we need to install it’s installer first. We need to make use of composer for that.

```
$ composer global require laravel/installer
```

This command will install laravel installer at system. This installation is at global scope, so you type command from any directory at terminal. To verify type the given command –

```
$ laravel
```

This command will open a command palette of Laravel Installer.

To create ad install laravel project in system, With the name of **blog** a laravel project will be created at your specified path.

By using composer

Alternatively, we can also install Laravel by Composer command **create-project**.

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

Here is the complete command to create a laravel project-

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

After following these steps we can install a Laravel application into system.

To start the development server of Laravel –

```
$ php artisan serve
```

This command outputs –

Starting Laravel development server: http://127.0.0.1:8000

Assuming laravel already installed at system.

Create Database & Connect

To create a database, either we can create via Manual tool of PhpMyadmin or by means of a mysql command.

```
CREATE DATABASE laravel_app;
```

To connect database with application, Open **.env file** from application root. Search for **DB_** and update your details.

```
DB_PORT=3306
DB_DATABASE=laravel_app
DB_USERNAME=root
DB_PASSWORD=root
```

Create Migrations

We need few migration files –

- User migration to store user data
- Role migration to store roles
- Role User Migration to store user id and role id

Open project into terminal and run these artisan commands.

```
$ php artisan make:migration CreateRolesTable

$ php artisan make:migration CreateRoleUserTable
```

It will create two migration files 2021_04_03_042735_**create_roles_table.php** and 2021_04_03_042802_**create_role_user_table.php** at location **/database/migrations**.

Already we have 2014_10_12_000000_create_users_table.php migration available by default which is for users table.

Open 2021_04_03_042735_**create_roles_table.php** and write this complete code into it.

```
#RoleMigration
<?php

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

class CreateRolesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('roles', function (Blueprint $table) {
            $table->id();
            $table->string('name', 25);
        });
    }

    /**
     * Reverse the migrations.
     *
    public function down()
    {
        Schema::dropIfExists('roles');
    }
}
```

Open 2021_04_03_042802_**create_role_user_table.php** and write this complete code into it.

```
#RoleUserTable
<?php

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

class CreateRoleUserTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('role_user', function (Blueprint $table) {
            $table->id();
            $table->unsignedInteger('user_id');
            $table->unsignedInteger('role_id');

            $table->foreign('user_id')->references('id')->on('users')->onDelete('cascade');

            $table->foreign('role_id')->references('id')->on('roles')->onDelete('cascade');
        });
    }

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

Also If we open **users migration**, we should like this.

```
#UserMigration
<?php
```

```
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $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

Next, we need to create two models and already we have a User model. Back to terminal and run these artisan commands.

```
$ php artisan make:model Role

$ php artisan make:model RoleUser
```

Menu

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
     */
    protected $casts = [
        'email_verified_at' => 'datetime',
    /**
     * 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');
```

Data for Role User Table

```
(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
use App\Http\Controllers\SiteController;

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

Application Testing

Prefinished Trim at Menards

Over 1,000 Profiles In-Stock. Shop Prefinished Moulding & Trim Now at Menards!

Menards

SHOP NOW

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
- Laravel 8 Layouts And Views Complete Guide
- Laravel 8 Routing Tutorial Step by Step Guide
- Laravel 8 Send Mail using Gmail SMTP Server
- Laravel 8 Send Push Notification to Android Using Firebase
- Laravel 8 Send Push Notification to IOS Using Firebase
- Laravel 8 Stub Customization

Related Posts:

- Laravel 8 Has Many Through Eloquent Relationship Tutorial
- Laravel 8 One to Many Eloquent Relationship Tutorial
- Laravel 8 One to One Eloquent Relationship Tutorial
- Laravel 8 Has One Through Eloquent Relationship Tutorial
- Eloquent Mutators and Accessors in Laravel 8 Tutorial

Find Us on Youtube

Online Web Tutor Youtube

Menu

ezoic report this ad

Popular Posts

Sorry. No data so far.

Categories

Blogging (1)

CakePHP 4 (11)

CodeIgniter 4 (156)

Education (1)

jQuery & Javascript (39)

Laravel 8 (152)

MySQL (10)

Node Js (15)

PHP Miscellaneous (33)

Wordpress (22)

Recent Posts

How To Read XML Data to JSON in CodeIgniter 4

How To Read XML Data to JSON in PHP Tutorial

MySQL Sum with If Condition Tutorial

What are Working Components of CodeIgniter 4?

Concept of Lazy Loading Image Using jQuery

Menu

Menu

Menu

Menu

Menu

Menu

Menu

Menu

Menu