url: https://appdividend.com/2022/01/21/laravel-many-to-many-relationship/

Laravel Many To Many Relationship: Complete Guide

[[https://secure.gravatar.com/avatar/e7d1ba5ff04208824be7e0af1282c5cd?s=26&d=mm&r=g](https://appdividend.com/author/kunumj110470116021/)By **Krunal**](https://appdividend.com/author/kunumj110470116021/) Last updated **Jan 21, 2022** [11](https://appdividend.com/2022/01/21/laravel-many-to-many-relationship/#comments)

In Laravel, eloquent relationships are defined as methods on your Eloquent model classes and they are very robust query builders. Eloquent makes managing and working with these relationships effortless, and helps a variety of common relationships. In this tutorial, we will see how to use many to many relationship in Laravel with an example.

**Laravel many to many**

Laravel many to many relationship is slightly more complicated than **hasOne** and **hasMany** relationships. The key in many to many relationship is the join (or pivot) table. The pivot table allows the relationship id from one model to be related to many other models and vice-versa.

**Many-to-many** relationships are defined by writing a method that returns the result of the **belongsToMany.**

Advertisement

In our example, we will define the two models.

1. **Category**
2. **Product**

In our example, **Multiple Categories** have **Multiple Products**, and an inverse relationship will be **Multiple Products** belongs to **Multiple Categories**.

So, what we can do is that, when we go to a particular category, we need to display all the **Products**.

Same, when we see the particular Product, we need to display all the **Categories** that belong to that specific Product.

We start this practical example by installing the **Laravel** project.

**Step 1: Install Laravel.**

I am using **Laravel Valet to** create a new project using the following command.

laravel **new** relationships

If you are not using Laravel Valet, install using the following command.

composer create-project laravel/laravel relationship --prefer-dist

Go into the project.

**cd** relationships

Open the project in your editor.

**code** .

First thing, set up the database.

**Step 2: Create a model and migration.**

We are defining two models for our example.

1. **Category**
2. **Product**

php artisan make:model Category -m

php artisan make:model Product -m

It will create products and categories, tables, and models.

Now, inside **create\_categories\_table,**define the following schema.

*// create\_categories\_table*

*/\*\**

*\* Run the migrations.*

*\**

*\* @return void*

*\*/*

**public** **function** **up**()

{

Schema::create('categories', **function** (Blueprint $table) {

$table->increments('id');

$table->string('title');

$table->timestamps();

});

}

Also, write the following schema inside the **create\_products\_table.**

*// create\_products\_table*

*/\*\**

*\* Run the migrations.*

*\**

*\* @return void*

*\*/*

**public** **function** **up**()

{

Schema::create('products', **function** (Blueprint $table) {

$table->increments('id');

$table->string('name');

$table->float('price');

$table->timestamps();

});

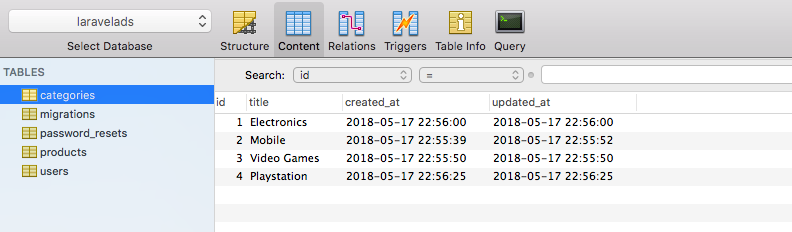
}

Now, go to the terminal and create the tables using the following command.

php artisan migrate

**Step 3: Define random categories manually.**

In this example, we are just focusing on how **many to many relationship** works how we can assign the values and create a relationship with one to another table. So, we make three categories manually inside the database table.

[](https://appdividend.com/wp-content/uploads/2018/05/Laravel-Many-To-Many-Relationship.png)

So, we have manually defined three categories.

**Step 4: Define a Pivot table.**

Many-to-many relations require an intermediary table to manage the relationship.

The most straightforward implementation of the intermediary table, known as a *pivot table*, would consist of just two columns for storing the foreign keys pointing to each related pair of records.

**How to create a Pivot table in Laravel**

1. **The name of the pivot table** should consist of **singular** names of both tables, separated by underscore symbols, and these names should be arranged in **alphabetical** order, so we have to have **category\_product**, not **product\_category**.
2. **To create a pivot table,** we can create the simple migration with **artisan make:migration** or use the community package [Laravel 5 Generators Extended](https://github.com/laracasts/Laravel-5-Generators-Extended" \l "pivot-tables" \t "_blank). For example, we have the command **artisan make:migration:pivot**.
3. **Pivot table fields:** by default, there should be only two fields – the foreign key to each table, in our case **category\_id** and **product\_id**. You can insert more fields if you need, then you need to add them to the relationship assignment.

Let’s use the above rules in our example.

So, we have already defined two table schemas 1) Category 2) Product.

Now, we need to define the third table, which is the Pivot table.

The **Pivot table** is the relationship between two tables.

So the **Pivot table** has these columns.

1. id
2. category\_id
3. product\_id

Now, we are creating **Many to Many relationships; that** is why many products have categories. For example, to create a migration file, type the following command.

php artisan make:migration create\_category\_product\_table --create=category\_product

Now, define the following schema in the migration file.

*// create\_category\_product\_table*

*/\*\**

*\* Run the migrations.*

*\**

*\* @return void*

*\*/*

**public** **function** **up**()

{

Schema::create('category\_product', **function** (Blueprint $table) {

$table->increments('id');

$table->integer('category\_id')->unsigned();

$table->integer('product\_id')->unsigned();

});

}

Now, migrate using the following command.

php artisan migrate

**Step 5: Define Many To Many relationships.**

Now, Multiple Categories belong to Multiple Products. So inside the **Product.php**file, we can define the **belongsToMany**relationship.

*// Product.php*

**<?php**

**namespace** **App**;

**use** **Illuminate**\**Database**\**Eloquent**\**Model**;

**class** **Product** **extends** **Model**

{

**public** **function** **categories**()

{

**return** **$this**->belongsToMany(Category::class);

}

}

Also, the same for the products. Multiple Products belong To Multiple Categories. So inside the **Category.php**file, we can define the **belongsToMany**relationship.

*// Category.php*

**<?php**

**namespace** **App**;

**use** **Illuminate**\**Database**\**Eloquent**\**Model**;

**use** **App**\**Product**;

**class** **Category** **extends** **Model**

{

**public** **function** **products**()

{

**return** **$this**->belongsToMany(Product::class);

}

}

So, we have defined the relationship between them.

Now, let us create a new product and assign the category to the Product.

**Step 6: Create a Product.**

In a real-time scenario, we create a form, and then through a **POST** request, we insert the **Product** data into the table.

However, in this example, we will not define any form; we directly store the data into the database because our goal is to use many to many relationships to the complex scenario.

Now, define a route that saves the Product into the database and assigns the Product to the category using many to many relationships.

Now, we have four(4) categories. So, we create a product and assign the two categories to one Product.

First, create a ProductController using the following command.

php artisan make:controller ProductController

The next step, define the route to store the Product.

Now, I am using a GET request for saving the data because we have not created the form, so we take every data manually.

*// ProductController.php*

Route::get('product/create', 'ProductController@create')->name('product.create');

Import both the models in this **ProductController.php**file.

Also, Now, what we are going to do is we will create one Product that belongs To Many Categories.

In this example tutorial, we will create a **Product** called **God of War.**

Now, the God of War belongs to two categories.

1. Video Games
2. Playstation

So, when we create a product, we also need to fill the pivot table with the two categories.

So, our product\_id will be one but, the category\_id will be different so that it will create two rows in the **category\_product**table.

Now, write the following code inside ProductController’screate() function.

*// ProductController.php*

**<?php**

**namespace** **App**\**Http**\**Controllers**;

**use** **App**\**Category**;

**use** **App**\**Product**;

**use** **Illuminate**\**Http**\**Request**;

**class** **ProductController** **extends** **Controller**

{

**public** **function** **create**(Request $request)

{

$product = **new** Product;

$product->name = 'God of War';

$product->price = 40;

$product->save();

$category = Category::find([3, 4]);

$product->categories()->attach($category);

**return** 'Success';

}

}

**Explanation.**

First, we have created the Product and saved it in the **products**table.

Now, it is time to assign the categories to the newly created Product.

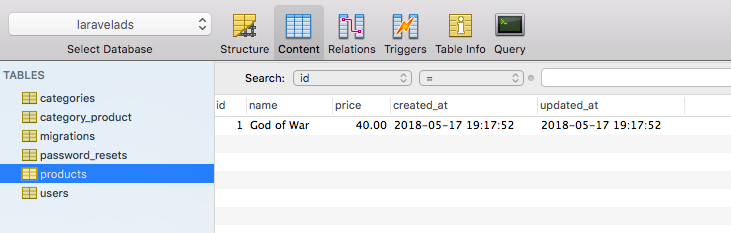
Since we need an ID of categories, so right now, I have coded manually, but in real-time, you have those ids in the form request.

Now, the **attach**() function will assign those category ids to the newly created Product and create two new rows inside the pivot table. Each row has a relationship with its Product and category.

The next step is going to this URL: **http://relationships.test/product/create** or **http://localhost:8000/product/create**

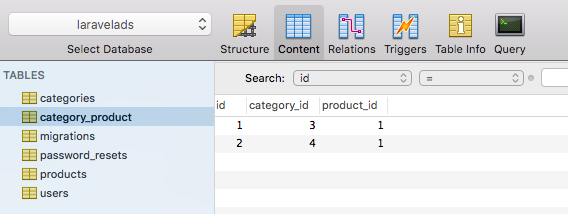
You can see the “Success.”

Now, go to the database and see the **products**table.

[](https://appdividend.com/wp-content/uploads/2018/05/Many-To-Many-Relationship-in-Laravel.png)

Also, you can check the Pivot Table, which is **a create\_product**table.

If we have done it all correctly, we can see the two rows inside the table, where product\_id is the same 1 for both the rows, but category\_id’s are different, which is **3**and **4.**

[](https://appdividend.com/wp-content/uploads/2018/05/Laravel-5.6-Many-To-Many-Relationships.png)

Yayy!!, we have successfully attached the two categories to one Product.

The next step is to display the Product Information and display all the Categories inside that Product.

**Step 7: Display Product Information.**

Define the route that can display all the information.

*// web.php*

Route::get('product/{product}', 'ProductController@show')->name('product.show');

Now, define the ProductController’s show function.

In this function, I am using Routing Model Binding.

*// ProductController.php*

**public** **function** **show**(Product $product)

{

**return** view('product.show', compact('product'));

}

We pass the product id in the get request, we fetch the product details via Route Model Binding directly and give that product detail to the view.

Create a new folder inside the **views**folder called **products**and inside that, create one file called **show.blade.php.**

Write the following code inside the **show.blade.php**file. We display the product name, price, and belongs to categories.

// show.blade.php

<h2>Product Name: </h2>

<p>{{ $product->name }} || ${{ money\_format($product->price, 2) }}</p>

<h3>Product Belongs to</h3>

<ul>

@foreach($product->categories as $category)

<li>{{ $category->title }}</li>

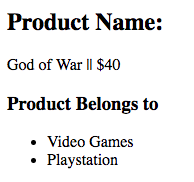
@endforeach

</ul>

As we have defined the relationship, we can directly fetch all the categories from the Product model, which is the magic of Eloquent Relationships in Laravel.

Now, go to this URL: **http://relationships.test/product/1** or **http://localhost:8000/product/1**

As you can see, we have only one Product created to see only one product detail.

[](https://appdividend.com/wp-content/uploads/2018/05/Laravel-attach-function.png)

So, this is how you can add multiple categories to a product and display multiple categories for a product.

Now, you can create as many products as you want and assign multiple categories to them.

You can also delete the relationship between the tables using the detach() function.

**Detach() function.**

*// ProductController.php*

**public** **function** **removeCategory**(Product $product)

{

$category = Category::find(3);

$product->categories()->detach($category);

**return** 'Success';

}

Now, define the following route inside **a web.php**file.

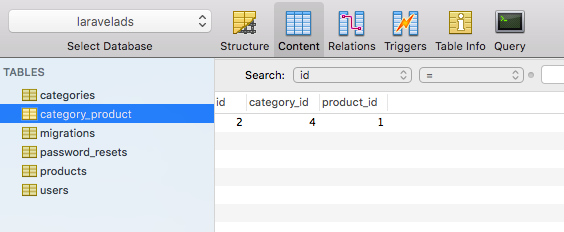
*// web.php*

Route::get('category/product/{product}', 'ProductController@removeCategory')->name('category.product.delete');

Next, type this URL: **http://relationships.test/category/product/1** or **http://localhost:8000/category/product/1.**

Hit enter, and we get success now. Go to the MySQL database and check the **category\_product**table.

You can see that the category\_id = 3 row is deleted, and we no longer belong to category\_id = 3.

[](https://appdividend.com/wp-content/uploads/2018/05/Laravel-detach-Function.png)

So, this is how you can maintain the **many to many relationships in Laravel.**

**Conclusion**

Laravel Many To Many Relationship relates a record on the table to one or many records in another table and vice versa. Unlike Laravel [One to Many](https://appdividend.com/2018/01/04/laravel-one-to-many-relationship-tutorial/) and [One to One](https://appdividend.com/2017/10/12/laravel-one-to-one-eloquent-relationships/), the key is a pivot table in this relationship. The pivot table is a table where you define between two tables to link them. A pivot table allows the relationship **id** from one model to be related to many other models and vice-versa.

One real-life example we can think of is products and categories. A category can have many products, and a product can relate to many categories; simple isn’t.

That’s it for this many to many relationship tutorial.