Laravel 5 Eloquent ORM

tutorials.kode-blog.com/laravel-5-eloquent-orm

Download: Eloquent ORM Tutorial Files

Previous Tutorial

Next Tutorial

Introduction

In the previous tutorial Laravel Migrations, we created the database schema for the tutorial project Larashop. In this tutorial, we are going to look at how we can use Eloquent ORM to eloquently add and retrieve data from the database.

Topics to be covered

We will cover the following topics in this tutorial

- We will cover the following topics in this tutorial
- Eloquent ORM Models
    - naming conventions
    - table name and primary keys
    - timestamps
- Laravel 5: Run raw SQL statement
- Eloquent ORM INSERT
- Eloquent ORM READ
- Eloquent ORM UPDATE
- Eloquent ORM DELETE
- Eloquent ORM models for Larashop tutorial project
- Larashop: Using models in controllers
- Larashop: Displaying data from models in views

Eloquent ORM Models

Eloquent models are models that extend the `Illuminate\Database\Eloquent\Model` class In the MVC architectural design, models are used to interact with data sources. Eloquent ORM is an Object Relational Mapper developed by Laravel. It implements the active record pattern and is used to interact with relational databases. Eloquent ORM can be used inside Laravel or outside Laravel since it's a complete package on its own. Models are Laravel 5 are located in the `/app` directory.

Naming conventions

The law of the gods state that model names should be singular and table names should be plural. Laravel automatically pluralizes the model name to get the table name. But what if your language is Swahili and Laravel does not understand Swahili?

Eloquent Models provide the above as the default implementation. You can explicitly specify a table name if you

want to. We will now create a model for the categories table. The name of the model will be category. This is in accordance with the law of the gods.

1. Open the command prompt and browser to the project root
2. Run the following artisan command

```
php artisan make:model
Category
```

HERE,

- `php artisan make:model Category` creates a model named Category in `/app/Category.php`

Open the newly created model in `/app/Category.php`

You will get the following

```
<?php

namespace App;

use
Illuminate\Database\Eloquent\Model;

class Category extends Model
{
    //
}
```

HERE,

- `use Illuminate\Database\Eloquent\Model;` imports the Eloquent model class namespace
- `class Category extends Model` defines a model Category that extends Model

Table name and primary key

By default, the plural form of the model name is used as the table name and the primary key field name is assumed to be id. This section shows you how you can explicitly define both the table and primary key field names Add the following lines to Category model

```
protected $primaryKey = 'id';
protected $table =
'categories';
```

HERE,

- `protected $primaryKey = 'id';` explicitly defines the primary key field name
- `protected $table = 'categories';` explicitly defines the name of the table

Record timestamps

By default, Laravel assumes you have added the following fields to all of your database tables

- created_at
- updated_at

These fields are updated whenever you create a new record or update an existing record. If you do not want to use these timestamp fields in your database tables, you can set the following property to turn them off.

```
public $timestamps =
false;
```

HERE,

- `public $timestamps = false;` tells Eloquent ORM model not to consider created_at and updated_at table fields.

Laravel 5: Run raw SQL Statement

In this section, we will create a base model that extends the eloquent model.

ORM frameworks are great but I sometimes find querying data from multiple tables a pain in a quite significant part of my body. The developers for Laravel understand this and as such, they provided mechanisms for executing raw SQL statements.

Our base model will contain methods for retrieving data using raw SQL statements and for executing INSERT, UPDATE & DELETE. I strongly recommend against executing raw INSERT, UPDATE, & DELETE for security reasons. If you have good reason to do so then you should make sure you sanitize all user submitted data before supplying it to the queries.

1. go to the command prompt
2. Run the following artisan command to create the BaseModel

```
php artisan make:model
BaseModel
```

1. Open BaseModel.php in /app/BaseModel.php
2. Modify the contents of BaseModel.php to the following

```
<?php
namespace App;

use Illuminate\Database\Eloquent\Model;
use DB;

class BaseModel extends Model {

    public function selectQuery($sql_stmt) {
        return DB::select($sql_stmt);
    }

    public function sqlStatement($sql_stmt)
{
        DB::statement($sql_stmt);
    }
}
```

HERE,

- `namespace App;` defines the namespace for our base model
- `use Illuminate\Database\Eloquent\Model;` imports the Eloquent ORM model
- `use DB;` imports the DB namespace
- `public function selectQuery($sqlstmt) { return DB::select($sqlstmt); }` defines a public function `selectQuery($sql_stmt)`. `$sqlstmt` is a string parameter that contains the SQL statement to be executed. `DB::select($sqlstmt)` executes the SQL statement
- `public function sqlStatement($sqlstmt) { DB::statement($sqlstmt); }` defines a public function `sqlStatement`. `$sqlstmt` is a string parameter that contains the SQL statement to be executed. `DB::statement($sqlstmt)` executes the SQL statement

We will now use our model to add a record to the database.

1. Open `/app/Category.php` model
2. Modify the code to the following

```
<?php

namespace App;

class Category extends BaseModel {
    protected $primaryKey = 'id';
    protected $table = 'categories';
    protected $fillable = array('name', 'created_at_ip',
'updated_at_ip');
}
```

HERE,

- `namespace App;` defines the model namespace

- `class Category extends BaseModel` defines the category class that extends the `BaseModel`
- `protected $primaryKey = 'id';` explicitly defines the primary key field name. The default is id so in this case it's not really necessary to specify it. Personally I prefer setting the name explicitly.
- `protected $table = 'categories';` explicitly defines the table name. The default is the plural form of the model name.
- `protected $fillable = array('name', 'createdatip', 'updatedatip');` defines field names that can be mass assigned. This is a security measure that ensures only authorized fieldnames are affected.

Eloquent ORM INSERT

In this section, we will use the category model to add data to a record to the database. For the sake of simplicity, we will do this from a route

1. open /app/Http/routes.php
2. add the following code

```
Route::get('/insert', function() {
   App\Category::create(array('name' =>
'Music'));
    return 'category added';
});
```

HERE,

- `App\Category::create(array('name' => 'Music'));` executes the create static function of the model. The array array('name' => 'Music') matches field names to values. Eloquent model will use these values to generate the SQL insert statement and execute it.
- `return 'category added';` outputs category added in the web browser

Open your web browser

Load the following URL

```
http://localhost/larashop/public/insert
```

You will get the following result

```
category
added
```

Eloquent ORM READ

In this section, we will use the category model to retrieve all categories from the database.

1. open `/app/Http/routes.php`

2. add the following code

```
Route::get('/read', function() {
    $category = new App\Category();

    $data = $category-
>all(array('name','id'));

    foreach ($data as $list) {
        echo $list->id . ' ' . $list->name .
'
';
    }
});
```

HERE,

- `$category = new App\Category();` creates an instance variable of Category model
- `$data = $category->all(array('name','id'));` calls the all method of the model. The array parameter array('name','id') is used to specified the column names that the query should return. If left blank, then all columns will be returned.
- `foreach ($data as $list) {…}` loops through the returned results and displays the rows in the browser.

Open your web browser

Load the following URL

```
http://localhost/larashop/public/read
```

You will get the results similar to the following

```
5
CLOTHING
4 FASHION
3 KIDS
1 MENS
16 Music
2 WOMENS
```

Eloquent ORM UPDATE

In this section, we will update a record using the id. Based on the above results, we will update music category with id 16 to HEAVY METAL 1. open `/app/Http/routes.php` 2. add the following code

```
Route::get('/update', function() {
    $category = App\Category::find(16);
    $category->name = 'HEAVY METAL';
    $category->save();

    $data = $category-
>all(array('name','id'));

    foreach ($data as $list) {
        echo $list->id . ' ' . $list->name .
'
';
    }
});
```

HERE,

- `$category = App\Category::find(16);` call find function of the model and pass in 16 as the primary key parameter value. Eloquent will return a record with primary key value 16
- `$category->name = 'HEAVY METAL';` assigns the value HEAVY METAL to the name field
- `$category->save();` saves the changes made to the record
- `$data = $category->all(array('name','id'));` retrieves all the categories
- `foreach ($data as $list) {…}` loops through all the records and display the value in the web browser.

Open your web browser

Load the following URL

```
http://localhost/larashop/public/update
```

You will get the following results.

```
5 CLOTHING
4 FASHION
16 HEAVY
METAL
3 KIDS
1 MENS
2 WOMENS
```

Eloquent ORM DELETE

In this section, we will delete the category CLOTHING. 1. open `/app/Http/routes.php` 2. Add the following code

```
Route::get('/delete', function() {
    $category = App\Category::find(5);
    $category->delete();

    $data = $category-
>all(array('name','id'));

    foreach ($data as $list) {
        echo $list->id . ' ' . $list->name .
'
';
    }
});
```

HERE,

- `$category->delete();` deletes the record that was retrieved via the find method

Open your web browser

Load the following URL

```
http://localhost/larashop/public/delete
```

Eloquent ORM models for Larashop tutorial project

Now that we have covered the basics of Eloquent ORM models, let's create the rest of the models for our tutorial project Open the command prompt Run the following command

```
cd C:\xampp\htdocs\larashop
```

Run the following artisan commands to create models for; 1. brands 2. products 3. posts

```
php artisan make:model Brand
php artisan make:model
Product
php artisan make:model Post
```

Open the newly created models in `/app/model_name` and replace the boiler plate code with the following. The code below extends the `BaseModel` as opposed to Model and defines the;

1. primary key field
2. table name
3. fillable field names

Brand.php

```
<?php

namespace App;

class Brand extends BaseModel {
    protected $primaryKey = 'id';
    protected $table = 'brands';
    protected $fillable = array('name', 'created_at_ip',
'updated_at_ip');
}
```

Product.php

```
<?php

namespace App;

class Product extends BaseModel {
    protected $primaryKey = 'id';
    protected $table = 'products';
    protected $fillable = array('name', 'title',
'description','price','category_id','brand_id','created_at_ip', 'updated_at_ip');
}
```

Post.php

```
<?php

namespace App;

class Post extends BaseModel {
    protected $primaryKey = 'id';
    protected $table = 'posts';
    protected $fillable = array('url', 'title',
'description','content','blog','created_at_ip', 'updated_at_ip');
}
```

Larashop: Using models in controllers

In this section, we will look at how we can use models in controllers to retrieve data and pass it to views for display.

1. open `app/Http/Controllers/Front.php`
2. Modify the code to the following

```
<?php

namespace App\Http\Controllers;

use App\Brand;
use App\Category;
use App\Product;
use App\Http\Controllers\Controller;
```

```
class Front extends Controller {

    var $brands;
    var $categories;
    var $products;
    var $title;
    var $description;

    public function __construct() {
        $this->brands = Brand::all(array('name'));
        $this->categories = Category::all(array('name'));
        $this->products = Product::all(array('id','name','price'));
    }

    public function index() {
        return view('home', array('title' => 'Welcome','description' => '','page'
=> 'home', 'brands' => $this->brands, 'categories' => $this->categories,
'products' => $this->products));
    }

    public function products() {
        return view('products', array('title' => 'Products Listing','description'
=> '','page' => 'products', 'brands' => $this->brands, 'categories' => $this-
>categories, 'products' => $this->products));
    }

    public function product_details($id) {
        $product = Product::find($id);
        return view('product_details', array('product' => $product,'title' =>
$product->name,'description' => '','page' => 'products', 'brands' => $this-
>brands, 'categories' => $this->categories, 'products' => $this->products));
    }

    public function product_categories($name) {
        return view('products', array('title' => 'Welcome','description' =>
'','page' => 'products', 'brands' => $this->brands, 'categories' => $this-
>categories, 'products' => $this->products));
    }

    public function product_brands($name, $category = null) {
        return view('products', array('title' => 'Welcome','description' =>
'','page' => 'products', 'brands' => $this->brands, 'categories' => $this-
>categories, 'products' => $this->products));
    }

    public function blog() {
        return view('blog', array('title' => 'Welcome','description' => '','page'
=> 'blog', 'brands' => $this->brands, 'categories' => $this->categories,
'products' => $this->products));
    }

    public function blog_post($id) {
        return view('blog_post', array('title' => 'Welcome','description' =>
'','page' => 'blog', 'brands' => $this->brands, 'categories' => $this->categories,
'products' => $this->products));
    }

    public function contact_us() {
        return view('contact_us', array('title' => 'Welcome','description' =>
'','page' => 'contact_us'));
```

```
    public function login() {
        return view('login', array('title' => 'Welcome','description' =>
'','page' => 'home'));
    }

    public function logout() {
        return view('login', array('title' => 'Welcome','description' =>
'','page' => 'home'));
    }

    public function cart() {
        return view('cart', array('title' => 'Welcome','description' =>'','page'
=> 'home'));
    }

    public function checkout() {
        return view('checkout', array('title' => 'Welcome','description' =>
'','page' => 'home'));
    }

    public function search($query) {
        return view('products', array('title' => 'Welcome','description' =>
'','page' => 'products'));
    }
}
```

HERE,

- `use App\Brand; use App\Category; use App\Product;` imports the Brand, category and Product models from the App namespace
- `var $brands; var $categories; var $products;` defines variables that will hold data from the respective models. This data is common to all the pages
- `var $title; var $description;` defines the title and meta description for search engine optimization purposes.
- `public function __construct() {…}` defines the controller constructor method. This method is called when the model is initialized and loads the data from models into the respective variables that we defined.
- `return view('home', array('title' => 'Welcome','description' => '','page' => 'home', 'brands' => $this->brands, 'categories' => $this->categories, 'products' => $this->products));` loads the home blade template views and passes in the array variable parameters

Larashop: Displaying data from models in views

In this section, we will look at how we can loop through data from the models and display it in views.

The following code shows the syntax for looping through records

```
@foreach ($brands as $brand)
    <li><a href='{{url("products/brands/$brand->name")}}'> <span class="pull-
right">(50)</span>{{$brand->name}}</a></li>
@endforeach
```

HERE

- `@foreach ($brands as $brand)` loops through the rows returned from the database and displays them as list items

Download the attached tutorial files for examples on how to display data in views.

Summary

Eloquent ORM models are easy to create and use. All you have to do is extend the Eloquent ORM model and you are ready to start using your model. Laravel allows you to execute raw SQL statements via the DB class. This makes Laravel very powerful and flexible.

What's next?

We are almost there with our tutorial project. Read the next tutorial on Laravel 5 Shopping Cart

Tutorial History

Tutorial version 1: Date Published 2015-08-30

Previous Tutorial

Next Tutorial