MAWLANA BHASHANI SCIENCE AND TECHNOLOGY UNIVERSITY

DEPARTMENT OF ICT

Lab Report No : 07

Course Code : ICT-3210

Course Title : Software Engineering Lab

Lab Report name : Eloquent ORM

Submitted by
Md. Faruk Hosen
ID : IT-17035
Session : 2016-2017
Year : 3rd Semester : 2nd

Submitted to
Mr. Tanvir Rahman
Lecturer,
Department of ICT,MBSTU
Santosh,Tangail-1902

Date of Submission : 31 October 2020

Eloquent ORM

1. What do you understand by Eloquent ORM?

Ans : The **Eloquent ORM** present in Laravel offers a simple yet beautiful ActiveRecord implementation to work with the database. Here, each database table offers a corresponding model which is used to interact with the same table. We can create Eloquent models using the make:model command.

Eloquent ORM (Object-Relational Mapping) is one of the main features of the Laravel framework. Eloquent ORM is also responsible for providing the internal methods at the same time when enforcing constraints on the relationship between database objects. Eloquent ORM represents database tables as classes, with their object instances tied to single table rows, while following the active record pattern.

2. List available types of relationships in Laravel Eloquent ?

Ans : Types of relationship in Laravel Eloquent are:

i. One To One
ii. One To Many
iii. Many To Many
iv. Has Many Through, and
v. Polymorphic Relations.

3. How can we create a record in Laravel using eloquent?

Ans : We need to create a new model instance if we want to create a new record in the database using Laravel eloquent. Then we are required to set attributes on the model and call the save() method.

Example :

```
public functionsaveProduct(Request $request )
$product = new product;
$product->name = $request->name;
$product->description = $request->name;
$product->save();
```

4. How to define each type relationships in Laravel ?

Ans : Let's learn how to define each type:

One to One :

A one-to-one relationship is a very basic relation. For example, a User model might be associated with one Phone. To define this relationship, we place a phone method on the User model. The phone method should return the results of the hasOne method on the base Eloquent model class:

```php
<?php
namespace App;
use Illuminate\Database\Eloquent\Model;
class User extends Model
{
    public function phone()
    {
        return $this->hasOne('App\Phone');//hasOne() method
    }
}
```

The first argument passed to the hasOne method is the name of the related model. Once the relationship is defined, we may retrieve the related record using Eloquent's dynamic properties. Dynamic properties allow you to access relationship functions as if they were properties defined on the model:

```php
$phone = User::find(1)->phone;
```

Eloquent assumes the foreign key of the relationship based on the model name. In this case, the Phone model is automatically assumed to have a user_id foreign key. If you wish to override this convention, you may pass a second argument to the hasOne method:

```php
return $this->hasOne('App\Phone', 'foreign_key');
```

One To Many :

A "one-to-many" relationship is used to define relationships where a single model owns any amount of other models. For example, a blog post may have

an infinite number of comments. Like all other Eloquent relationships, one-to-many relationships are defined by placing a function on your Eloquent model:

```php
<?php
namespace App;
use Illuminate\Database\Eloquent\Model;
class Post extends Model
{
    public function comments()
    {
        return $this->hasMany('App\Comment');
    }
}
```

Remember, Eloquent will automatically determine the proper foreign key column on the Comment model. Once the relationship has been defined, we can access the collection of comments by accessing the comments property. Remember, since Eloquent provides "dynamic properties", we can access relationship functions as if they were defined as properties on the model:

```php
$comments = App\Post::find(1)->comments;

foreach ($comments as $comment) {
    //
}
```

Many to Many :

Many-to-many relations are slightly more complicated than hasOne and hasMany relationships. An example of such a relationship is a user with many roles, where the roles are also shared by other users. For example, many users may have the role of "Admin". To define this relationship, three database tables are needed: users, roles, and role_user. The role_user table is derived from the alphabetical order of the related model names, and contains the user_id and role_id columns.

Many-to-many relationships are defined by writing a method that calls the belongsToMany method on the base Eloquent class. For example, let's define the roles method on our User model:

```php
<?php
namespace App;
use Illuminate\Database\Eloquent\Model;
class User extends Model
{
    public function roles()
    {
        return $this->belongsToMany('App\Role');
    }
}
```

Once the relationship is defined, you may access the user's roles using the roles dynamic property:

```php
$user = App\User::find(1);
foreach ($user->roles as $role) {
    //
}
```

Conclusion :

In this lab, we learned how Eloquent ORM relationships works and how laravel implements it. We learned why we should use Eloquent relationships and how to implement it in a real world application.