Laravel's Query Builder and Eloquent ORM

December 14th, 2022

An web application always needs to interact with a database and Laravel makes this task hassle free. A few tools that make Laravel an awesome framework is the inclusion of "Query Builder and Eloquent ORM". Through this blog I intend to share few quick pointers on these concepts.

Query Builder:

In Laravel the database query builder provides an easy interface to create and run database queries. It can be used to perform all the database operations in your application, from basic DB Connection, CRUD, Aggregates, etc. and it works on all supported database systems like a champ.

The notable factor about query builder is that, since it uses the PHP Data Objects (PDO), we need not worry about SQL injection attacks (Make sure we don't inadvertently remove this protection). We can avoid all those lines of code to sanitize the data before feeding it to the DB.

So, how do we create a simple select query to fetch all values from the users table?

```
$users = DB::table('users')->get();
```

DB::table is responsible to begin a fluent query against a database table. The table from which the value has to be selected is mentioned inside the brackets within quotes and finally the get() method gets the values. Similarly to fetch a single row we can modify the above code by adding a where clause

```
$user = DB::table('users')->where('name', 'John')->first();
```

Here, we are trying to fetch a row that has the value John in its name column. The first() method will only return the first find. What if we need only the user id of John. Instead of returning the entire result array we can simply pluck out that specific column?

```
$user_id = DB::table('users')->where('name', 'John')->pluck('id');
```

For specifying more than one column we can use the select clause

```
$users = DB::table('users')->select('name', 'email')->get();
```

I now believe you are getting the grip. Things get more interesting further down.

We often write queries against certain 'where conditions'. So how do we fetch the list of users whose user_id is less than 10?

```
$users = DB::table('users')->where(id, '<', 10)->get();
```

Yes, we split up the operator and the operands as three parameters and feed it to the where conditions. Now we have a situation, we need to fetch all those users whose user_id falls between 10 and 20.

```
$users = DB::table('users')->whereBetween('id', array(10, 20))->get()
```

Laravel has the whereBetween(), whereNotBetween(), wherein() and whereNotIn() methods to which we can pass values as an array.

Why are we passing values as an array and not as comma separated parameters?

At the start of this blog I did mention about SQL injection attacks. Let's say that the values 10 and 20 are taken as user inputs. As programmers we cannot trust what the user types into the input field. He can be a valid user who enters proper values or someone trying to enter false values and crash your DB.

```
$users = DB::table('users')->whereBetween('id', array($from, $to))->get()
```

Here $from and $to are user inputs. If we look in to the Laravel's database connection class for the select() method this array is wrapped around PDO connection and it is responsible to sanitize the data before the query is executed. So you have clean queries!!

Using Query builder we can also write raw SQL queries

```
DB::select(DB::raw("SELECT * FROM `users` WHERE name = '$name' "));
```

Here $name is obtained from user input. $name may contain malicious code therefore we need to alter the above code to make it SQL friendly.

```
:(DB::raw("SELECT * FROM `users` WHERE `name` = :username"), array('username' => $name));
```

So the array value when passed through the PDO connection gets sanitized

Connect with us

- Find office locations
- Email us

Contact us for business enquiries

So the array value when passed through the PDO connection gets sanitized.

Eloquent ORM:

We start by answering few questions.

What is ORM?

ORM or Object Relational Mapper is a technique to access objects without having to consider how those objects are related to their source.

What is Eloquent?

The ORM included in Laravel is called Eloquent and it enables us to work with the database objects and relationships using an expressive syntax. It is similar to working with objects in PHP. In Laravel, each database table has a corresponding "Model". Eloquent ORM provides Active Record implementation which means that each model we create in our MVC structure corresponds to a table in our database.

Creating an Eloquent model is similar to creating a class. All the model files should be in the app/models folder.

```
class Group extends Eloquent { }
```

All Eloquent models extend from the Eloquent class. The lower-case, plural name of the class will be used as the table name unless another name is explicitly specified. Eloquent will also assume that each table has a primary key column named "id" unless specified. We can specify a table as follows:

```
class Group extends Eloquent

{

protected $table = 'group_list';   // will point to group_list table if mentioned

}
```

Here, Group model will correspond to groups table (by default). We can access the data in the groups table using the basic CRUD operations.

By default Eloquent models will have auto-incrementing keys.

Create:

```
$new_group = new Group;
$new_group->name = 'NewGroup';
$new_group->description = 'Awesome Group';
$new_group->save();
<read:< pre="">

// To get all groups

Group::all();

// To find a group by passing the group id etc.

Group::find($id);
```

// Try to retrieve a model by primary key else throw an exception

```
$model = User::findOrFail(1);
$model = User::where('id', '>', 5)->firstOrFail();
```

Update:

// Retrieve and update

```
$group = Group::find(1);
$group->name = 'Group01';
$group->save();
```

// Using a WHERE clause

```
Group::where('name', '=', 'Group01')-date(array('name' => 'Group1'));
```

// Delete one record

```
$group = Group::find(1);
$group->delete();
```

// Delete several

```
Group::destroy(1, 2, 3);
Group::where('id', '<', 10)->delete();
```

There are more topics to cover under Eloquent, hopefully in the next blog.

So we have come to an end!!

We discussed on Laravel's query builder and how it makes our DB interaction hassle free. We wrote a basic select query, saw few ways to fetch values depending on our web application development needs and how to avoid SQL injection while writing raw SQL queries using query builder. Next, we tried to get a grasp on Eloquent ORM, how to create a model and few quick examples on the CRUD operations. I hope these quick pointers could benefit a few of you and help you get started. Feel free to drop in your comments and also point out if any typos.

Happy Coding!!

Know More About This Topic

Latest Post

Insights and Highlights from Day 2 of Laracon IN – Jithin T C

I had the chance to attend the Laracon IN conference, which took place in Ahmedabad, Gujarat, last month, along with...

Learn More..

Insights and Highlights from Day 1 of Laracon IN – Vishal Vijayan

As a project manager at Cubet and a Laravel enthusiast, I recently had the opportunity to attend the much-awaited Laracon...

Learn More..

How to Reduce Development Costs by Leveraging Open-Source Solutions?

Software development is a complex task, and you can hire a software development company for the development process. Besides, the...

Learn More..

© 2023 Cubet

Have any questions?

Business enquiries
sales@cubettech.com

OR

BOOK A FREE CONSULTATION

Follow us: