

Web Programming Framework

Topic 4: Query in the Controller

WEEK 04

Informatics Engineering
Universitas Surabaya



Outline

1

Types of query methods in laravel

2

Introduction to raw queries

3

Introduction to query builder

4

Introduction to Eloquent ORM

5

Controller & View Concepts

6

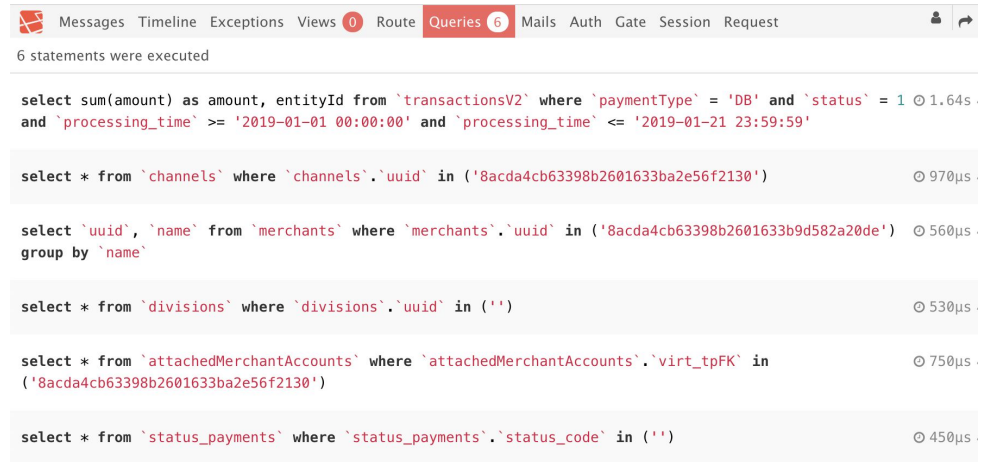
Using Route, Controller, and View in online reporting / ticketing system

Query Communication

There are 3 variants of query syntax to manage data from the database, namely:

- Raw Queries
- Query Builder
- Eloquent ORM

All techniques will generate queries based on DBMS language (as shown in the image)



The screenshot shows a web-based interface for a database management system. At the top, there is a navigation bar with tabs: Messages, Timeline, Exceptions, Views (0), Route, Queries (6), Mails, Auth, Gate, Session, and Request. Below the navigation bar, it states "6 statements were executed". The main area displays six SQL queries, each with its execution time in microseconds.

```
select sum(amount) as amount, entityId from `transactionsV2` where `paymentType` = 'DB' and `status` = 1 and `processing_time` >= '2019-01-01 00:00:00' and `processing_time` <= '2019-01-21 23:59:59' 1.64s
```

```
select * from `channels` where `channels`.`uuid` in ('8acda4cb63398b2601633ba2e56f2130') 970µs
```

```
select `uuid`, `name` from `merchants` where `merchants`.`uuid` in ('8acda4cb63398b2601633b9d582a20de') group by `name` 560µs
```

```
select * from `divisions` where `divisions`.`uuid` in ('') 530µs
```

```
select * from `attachedMerchantAccounts` where `attachedMerchantAccounts`.`virt_tpFK` in ('8acda4cb63398b2601633ba2e56f2130') 750µs
```

```
select * from `status_payments` where `status_payments`.`status_code` in ('') 450µs
```

Raw Queries

Raw Query is a method that EXPLICITLY uses "native" SQL to manage data from a DBMS.

- (Advantages) Raw Query is easier to use if we are already familiar with queries in the Native SQL.
- (Disadvantages for long-term development) If you need to migrate your schema into another DBMS (i.e : from MySQL to PostgreSQL), then you need to adjust each query so it can run in the new DBMS.

Source : <https://laravel.com/docs/10.x/queries#raw-expressions>

Example : There are 2 tables: **foods** and **categories**. The query to display the all foods and its category is presented below:

```
DB::raw("select f.name, c.name from foods f inner join categories c on. c.id=f.category_id");
```

Query Builder

- Query Builder is a **query method provided by Laravel that uses PHP language** to access the database.
- With this method, Laravel will automatically adjusts query into a raw format (native SQL form) that matches the target DBMS .
- The advantage is, if our project utilizes multiple DBMS, then we don't need to re-adjust the query. Query Builder will automatically adjust it for us.

Example : There are 2 tables: **fasum** and **ticket**. The query to display the all foods and its category is presented below:

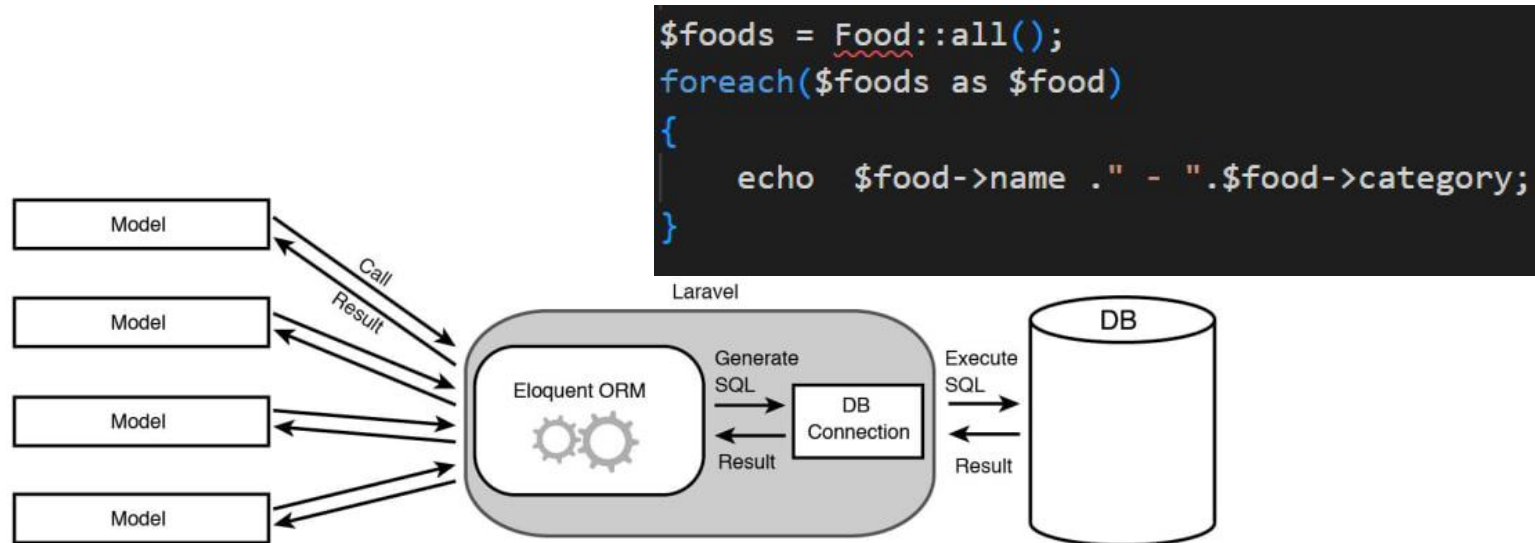
Source : <https://laravel.com/docs/10.x/queries>

```
DB::table("foods as f")  
    ->join("categories c")  
    ->select("f.name", "c.name")  
    ->get();
```

Eloquent ORM

Query Method that implements “Eloquent ORM” Model Laravel to manage Single table Query and Query on table relation.

Source : <https://laravel.com/docs/10.x/eloquent>



Where is Query syntax used?

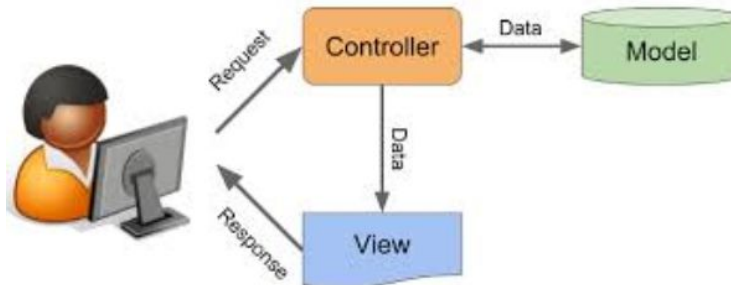
Lots of Query Syntax can be found in the **Controller file** or **Model file** to manage data requested by User.

User requests are processed by Laravel application in the Routing File and then directed to a specific Controller file.

Controllers

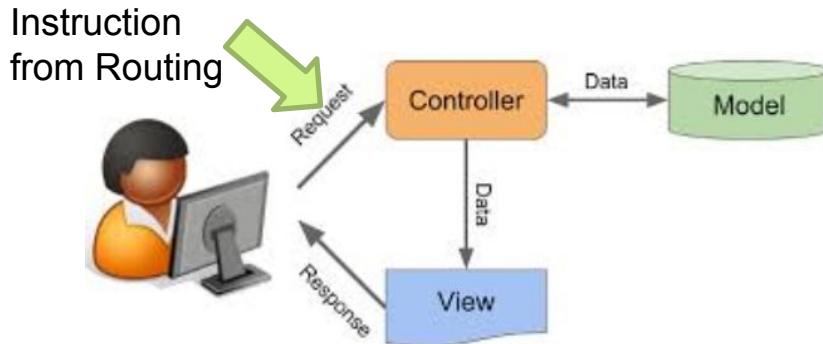
Controllers

- Controllers in Laravel are PHP files located in the App/Http/Controllers directory.
- The controller contains PHP script with Object Oriented structure.
- The Controller class name will be the same as the File name of the controller. Please note that “all names are case sensitive”.
- Usually, controller receives requests from users and then handle it by performing some logic statements. The output will be returned as responses for a specific view.
- We can access database from a Controller by using either Query Builder or Eloquent ORM.



Controller Relationship with Routing

- Controllers and Routing work together to handle requests.
- Routing manages HTTP commands, URLs and map in the Routing table to direct data handling.
- The results of the instructions from Routing will be the initial data for a function on a particular controller. In general, the initial data is a parameter.



Contents of The Controller

- Generally, the controller handles data display and data manipulation.
- Data display: displaying all the data in the database, displaying certain data in accordance with criteria
- Data manipulation: adding, editing and deleting data.
- The above functions are usually called CRUD functions (Create, Retrieve, Update and Delete)

Resource Controller

Laravel gives convenient syntaxes to help developers create "CRUD" on the controller with one single line of code.

To use it, we can use the following Artisan command:

```
php artisan make:controller PhotoController --resource
```

When using resources, you only need to create one method in routing files (web.php)

```
use App\Http\Controllers\PhotoController;  
  
Route::resource('photos', PhotoController::class);
```

That single method will manage CRUD in the Photo table (on case above) .

Source : <https://laravel.com/docs/10.x/controllers#resource-controllers>

Resource Controller

To manage CRUD, we must follow the existing Action Handler rules determined by Laravel Resource Controller.

Actions Handled By Resource Controller

Verb	URI	Action	Route Name
GET	/photos	index	photos.index
GET	/photos/create	create	photos.create
POST	/photos	store	photos.store
GET	/photos/{photo}	show	photos.show
GET	/photos/{photo}/edit	edit	photos.edit
PUT/PATCH	/photos/{photo}	update	photos.update
DELETE	/photos/{photo}	destroy	photos.destroy

Resource Controller Explanation

If your routing file contains

```
use App\Http\Controllers\PhotoController;  
  
Route::resource('photos', PhotoController::class);
```

Then your website will get the following routes :

1. GET method with url BASE_URL/photos redirects to PhotoController index() function and named as 'photos.index '
2. GET method with url BASE_URL/photos/create redirected to PhotoController create() function and named as 'photos.create '
3. POST method with url BASE_URL/photos redirects to PhotoController store() function and named as 'photos.store '
4. (etc until)
5. DELETE method with url BASE_URL/photos/{photo} redirected to PhotoController destroy() function and named as 'photos.destroy '

NB: /photos/{photo} will contain the ID of {photo} photo to be deleted

Example : if we want to delete photo id=1, then url must be BASE_URL/photos/1 with the HTTP DELETE method

PRACTICE #1

(20 minutes)

1. Create 2 controllers in your 2 tables, namely: ``FoodController`` and ``CategoryController``
2. Name those controllers with **a singular** name
3. Use additional parameter `--resource`

Try: if you create a controller **without `--resources`**, what happens to the contents of the controller file? Discuss in class !

Changing the HTTP Method

- The implementation of HTTP methods usually done in a View (basically is a HTML form)
- The GET method is the default HTTP method used when we access the browser. GET Method transferred data parameters between open URLs, those parameters could be seen on the URL.
- The POST method is the HTTP method used to secure the transferred data from seen on the URL. To change the method, just like on Web Programming course, we need to change the method on the form element into "POST".

```
<form action="/foo/bar" method="POST">
```



.....

```
</form>
```


Changing the HTTP Method

- For methods other than GET and POST, you have to do "HTTP method spoofing" based on <https://laravel.com/docs/10.x/controllers#resource-controllers> in Spoofing Form Methods subchapter (<https://laravel.com/docs/10.x/routing#form-method-spoofing>)
- The following example shows a Form Method Spoofing for the PUT method. Fill the form method with "POST" and include "@method('PUT')" blade syntax

For convenience, you may use the `@method` [Blade directive](#) to generate the `_method` input field:

```
<form action="/example" method="POST">
    @method('PUT')
    @csrf
</form>
```

HTML forms do not support `PUT`, `PATCH`, or `DELETE` actions. So, when defining `PUT`, `PATCH`, or `DELETE` routes that are called from an HTML form, you will need to add a hidden `_method` field to the form. The value sent with the `_method` field will be used as the HTTP request method:

```
<form action="/example" method="POST">
  <input type="hidden" name="_method" value="PUT">
  <input type="hidden" name="_token" value="{{ csrf_token() }}">
</form>
```

For convenience, you may use the `@method` [Blade directive](#) to generate the `_method` input field:

```
<form action="/example" method="POST">
  @method('PUT')
  @csrf
</form>
```

Getting to know Eloquent ORM

Eloquent ORM is model class in the Laravel Framework. To make it easier, next we will use "model" terminology for this.

The Laravel model is in the app folder

There are 2 ways to make a Laravel model:

1. Make the model directly.

Artisan Syntax:

php artisan make:model ModelName

2. Because the models are related tightly with the controller, we can create model and controller simultaneously in a single syntax:

Artisan Syntax:

php artisan make:controller ControllerName --resource -model= ModelName

Filling in the Model

Model Class Structure consists of:

- Table Name
- Timestamps for created_at and updated_at status
- Primary Key column name
- Primary Key Column Type

(source : <https://laravel.com/docs/10.x/eloquent#eloquent-model-conventions>)

Eloquent Function that define relation with other tables (either one-to-one, one-to-many, many-to-many)

(source : <https://laravel.com/docs/10.x/eloquent-relationships#introduction>)

More in about Models

Table Name

Model name is singular, but database name is plural. Example: if the model is named **Product**, then the table name in the database is **products**. If the database name is not following the Laravel's naming convention, then we can override the configuration later.

Primary Key

The primary key column default name is **'id'**. Primary Key default type is **Unsigned Big Integer** and **has AUTO_INCREMENT** element. If we use different name and type for the primary key, we can also override it in the Model class.

Timestamps

By default all tables in the Laravel framework have **created_at** and **updated_at** columns. We can implement it in migrations by using `timestamps()`. If we used pre-designed schema (i.e : ERD from MySQL Workbench), then we need to re-adjust it later by overriding the configuration..

Overriding Model Defaults

- From the example image, we can identify that the database structures of "Soal" model is :
- Table name: 'soal'
- Primary Key(PK) column: 'soal_id '
- Primary Key(PK) not using auto_increment
- Primary Key(PK) type is varchar/string
- "Soal" table has timestamps, timestamps column's name is not 'created_at' & 'updated_at' but rather 'creation_date' and 'last_update'

```
class Soal extends Model
{
    protected $table = 'soal';
    protected $primaryKey = 'soal_id';
    public $incrementing = false;
    protected $keyType = 'string';
    public $timestamps = true;
    const CREATED_AT = 'creation_date';
    const UPDATED_AT = 'last_update';
}
```

Practice #2

(10 minutes)

1. Create 2 model classes for `Food` and `Category` in your database
2. Use singular names according to the name conventions

More about Eloquent ORM (Relation)

[https://laravel.com/docs/10.x/eloquent-relationships#
main-content](https://laravel.com/docs/10.x/eloquent-relationships#main-content)

Relationships in ORM

```
$foods = Food::all();  
foreach($foods as $food)  
{  
    echo $food->name . " - ".$food->category;  
}
```

Implementing **RELATION** on Eloquent.

1. Relation isn't automatically creating by system. Programmer **have to** define inside Model Eloquent so we can perform queries with ORM
2. The definition is only necessary once at the beginning when a new FK is installed

Relationships in ORM (Example)

Students are related (only) to 1 Unit

```
class Mahasiswa extends Model
{
    public function unit()
    {
        return $this->belongsTo('App\Unit', 'unit_id');
    }
}
```

The unit has many MKs taught

```
class Unit extends Model
{
    public function matakuliah()
    {
        return $this->hasMany('App\Matakuliah', 'mk_id');
    }
}
```

Controller with Models

You can integrate the Controller with a specific Model by using parameter **--model= ModelName** when creating the controller.

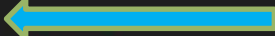
```
• (base) fikri@Fikris-MacBook-Air laravel10 % php artisan make:controller Product1Controller --resource --model=Product
```

```
A App\Models\Product model does not exist. Do you want to generate it?  
Yes
```

```
INFO Model [app/Models/Product.php] created successfully.
```

```
INFO Controller [app/Http/Controllers/Product1Controller.php] created successfully.
```

```
5 use Illuminate\Http\Request;  
6  
7 class ProductController extends Controller  
8 {  
9     /**  
10      * Display a listing of the resource.  
11      *  
12      * @return \Illuminate\Http\Response  
13      */  
14     public function index()  
15     {  
16         //  
17     }  
18 }
```

```
5 use App\Product;   
6 use Illuminate\Http\Request;  
7  
8 class Product1Controller extends Controller  
9 {  
10     /**  
11      * Display a listing of the resource.  
12      *  
13      * @return \Illuminate\Http\Response  
14      */  
15     public function index()  
16     {  
17         //  
18     }  
19 }
```

Notes

When performing a query with the DB class (namely: RAW and Query Builder), use import DB with the image syntax below.
(same as OOP concept) Place it above the class definition

```
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6  use Illuminate\Support\Facades\DB;
7  class ProductController extends Controller
8  {
```

Notes

2. When performing a query with ELOQUENT QUERY, don't forget to use Import on each Class Models.
(same as OOP concept) Place it above the class definition

```
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6  use Illuminate\Support\Facades\DB;
7  use App\Models\Product;
8
9  class ProductController extends Controller
10 {
11
```

Route List Additional Notes

You can see the installed routing (either manually or with the route list)

```
• (base) fikri@Fikris-MacBook-Air laravel10 % php artisan route:list

GET|HEAD / ..... ignition.executeSolution > Spatie\LaravelIgnition > ExecuteSolutionController
POST _ignition/execute-solution ..... ignition.executeSolution > Spatie\LaravelIgnition > ExecuteSolutionController
GET|HEAD _ignition/health-check ..... ignition.healthCheck > Spatie\LaravelIgnition > HealthCheckController
POST _ignition/update-config ..... ignition.updateConfig > Spatie\LaravelIgnition > UpdateConfigController
GET|HEAD api/user ..... sanctum.csrf-cookie > Laravel\Sanctum > CsrfCookieController@show
GET|HEAD sanctum/csrf-cookie ..... sanctum.csrf-cookie > Laravel\Sanctum > CsrfCookieController@show

Showing [6] routes
```

Viewing Route List Syntax

php artisan route:list

```
PS C:\xampp74\htdocs\wfp\mystore> php artisan route:list
```

Error

Call to undefined function resource()

at C:\xampp74\htdocs\wfp\mystore\routes\web.php:21
17 | Route::get('/', function () {

If you encounter this of error, then check the file and the error line of codes. There is a chance that you haven't implement "use" or not importing the controller correctly.

```
• (base) fikri@Fikris-MacBook-Air laravel10 % php artisan route:list
```

```
GET|HEAD / ..... ignition.executeSolution > Spatie\LaravelIgnition > ExecuteSolutionController
POST _ignition/execute-solution ..... ignition.executeSolution > Spatie\LaravelIgnition > ExecuteSolutionController
GET|HEAD _ignition/health-check ..... ignition.healthCheck > Spatie\LaravelIgnition > HealthCheckController
POST _ignition/update-config ..... ignition.updateConfig > Spatie\LaravelIgnition > UpdateConfigController
GET|HEAD api/user .....
GET|HEAD product ..... product.index > ProductController@index
POST product ..... product.store > ProductController@store
GET|HEAD product/create ..... product.create > ProductController@create
GET|HEAD product/{product} ..... product.show > ProductController@show
PUT|PATCH product/{product} ..... product.update > ProductController@update
DELETE product/{product} ..... product.destroy > ProductController@destroy
GET|HEAD product/{product}/edit ..... product.edit > ProductController@edit
GET|HEAD sanctum/csrf-cookie ..... sanctum.csrf-cookie > Laravel\Sanctum > CsrfCookieController@show
```

Showing [13] routes



View

View

View is a file with extension “.php” and containing html script, css, jquery, and php. In Laravel, it is known as blade template file.

Location of View: resources/views folder

View File Extension:

- When we want to use blade syntax then name it “<file_name>.blade.php ”
- When we don't use blade syntax then name it “<filename>.php”

Practice #3: Create a Controller to display all your Product Food data

Steps

1. Use Artisan Command to create a Controller and its model. Model name is singular of table name in the database.

```
php artisan make:controller FoodController --resource --model=Food
```

2. Fill the `index()` in the Controller file with a command to retrieve all data in the ticket table. The `index()` function is selected because `index` is usually used to display list of data.

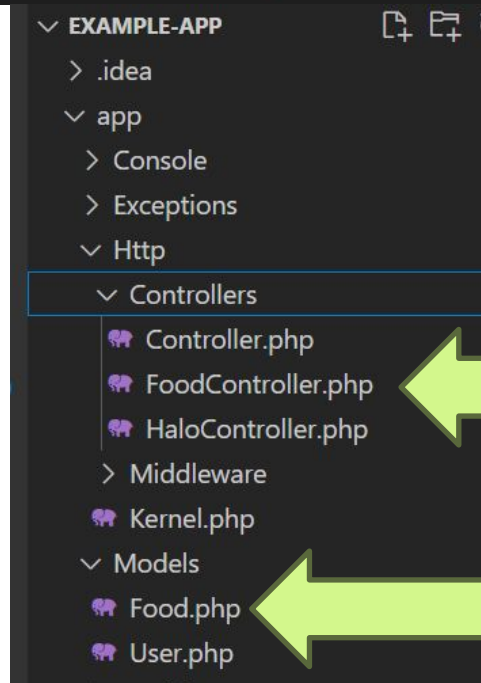
```
PS C:\laragon\www\example-app> php artisan make:controller FoodController --resource --model=Food
```

```
A App\Models\Food model does not exist. Do you want to generate it? (yes/no) [yes]
```

```
> yes
```

```
INFO Model [C:\laragon\www\example-app\app\Models\Food.php] created successfully.
```

```
INFO Controller [C:\laragon\www\example-app\app\Http\Controllers\FoodController.php] created successfully.
```



```
web.php FoodController.php X
app > Http > Controllers > FoodController.php > ...
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use App\Models\Food;
6 use Illuminate\Http\Request;
7
8 class FoodController extends Controller
9 {
10     /**
11      * Display a listing of the resource.
12      */
13     public function index()
14     {
15         //
16     }
17
18     /**
```

```
web.php FoodController.php Food.php X
app > Models > Food.php > ...
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 class Food extends Model
9 {
10     use HasFactory;
11 }
12
```

Add additional configuration of Food Model

```
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 class Food extends Model
9 {
10     use HasFactory;
11
12     protected $table = 'foods';
13     protected $primaryKey = 'id';
14     public $timestamps = true;
15
16 }
```

```
3 namespace App\Http\Controllers;
4
5 use App\Models\Food;
6 use Illuminate\Support\Facades\DB;
7 use Illuminate\Http\Request;
```

```
public function index()
{
    //Raw Query
    $allFoods = DB::raw("select * from foods");

    //Query Builder
    $allFoods = DB::table("foods")->get();

    //Eloquent Model
    $allFoods = Food::all();
}
```

```
3 use App\Http\Controllers\FoodController;
```

```
23 Route::resource('listmakanan', FoodController::class);
```

Warning

When an error appears like image below

Error

Class 'App\Http\Controllers\DB' not found

So, you need to import the DB in that class (see Notes)
Remember, all static classes that are called must be imported before being executed.

Its characteristic is `classname :: function ()`;
This means that in the import there must be “`use package\classname`”

3. Passing the data to a specific view.

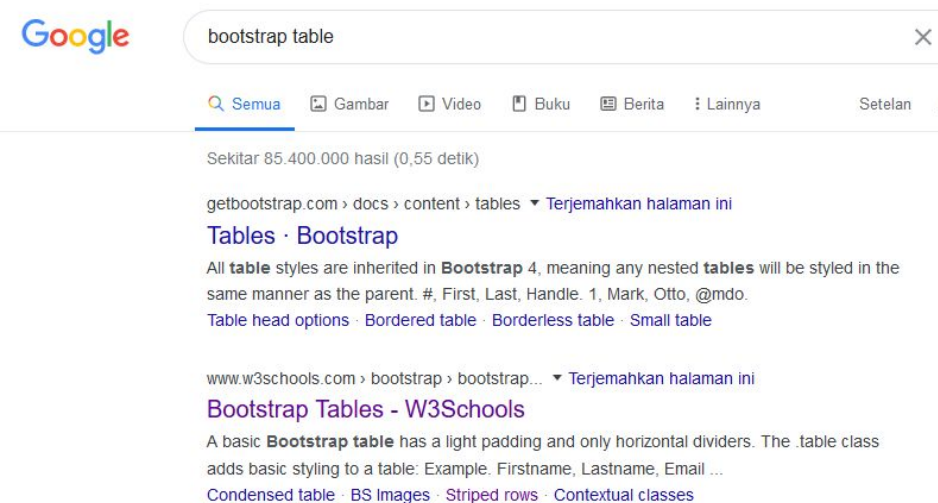
Naming pattern is 'foldername.viewfilename '. (i.e : 'foods.index' means that we are passing the data into 'index.blade.php' that is located in folder 'resource/view/foods')

Create call View and create a View

```
9  class FoodController extends Controller
14      public function index()
25
26          //method #1
27          return view('foods.index',compact('allFoods'));
28          //method #2
29          return view('foods.index',['foods'=>$allFoods]);
30
31      }
```

In the example above, we should create the view in:
/resources/views/**foods/index**.blade.php

Using Bootstrap to Make Pretty Tables



Google

bootstrap table

Semua Gambar Video Buku Berita Lainnya Setelan

Sekitar 85.400.000 hasil (0,55 detik)

getbootstrap.com > docs > content > tables ▾ [Terjemahkan halaman ini](#)
[Tables · Bootstrap](#)
All **table** styles are inherited in **Bootstrap 4**, meaning any nested **tables** will be styled in the same manner as the parent. #, First, Last, Handle. 1, Mark, Otto, @mdo.
[Table head options](#) · [Bordered table](#) · [Borderless table](#) · [Small table](#)

www.w3schools.com > bootstrap > bootstrap... ▾ [Terjemahkan halaman ini](#)
[Bootstrap Tables - W3Schools](#)
A basic **Bootstrap table** has a light padding and only horizontal dividers. The `.table` class adds basic styling to a table: Example. Firstname, Lastname, Email ...
[Condensed table](#) · [BS Images](#) · [Striped rows](#) · [Contextual classes](#)

Bootstrap Basic Table

A basic Bootstrap table has a light padding and only horizontal dividers.

The `.table` class adds basic styling to a table:

Example

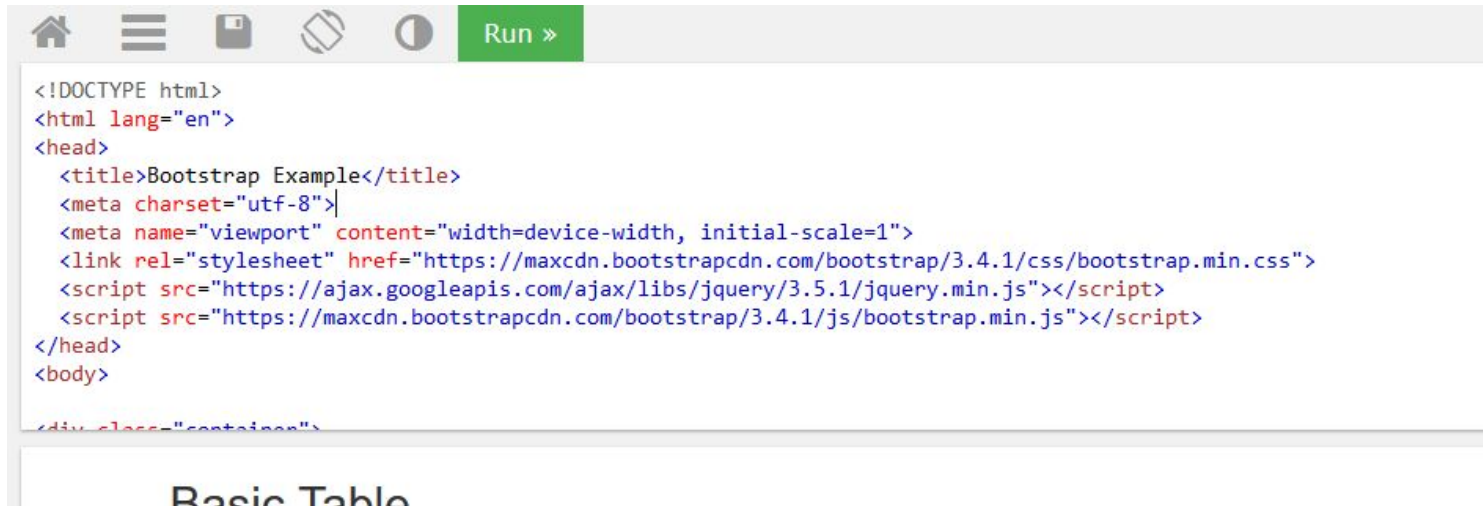
Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

Try it Yourself »

https://www.w3schools.com/bootstrap/bootstrap_tables.asp

Using Bootstrap to Make Pretty Tables

Copy and paste the example bootstrap table codes then paste it into your view(blade) file.



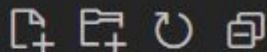
```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Bootstrap Example</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
</head>
<body>
  <div class="container">
```

Basic Table

EXPLORER



EXAMPLE-APP



resources

> css

> js

views

foods

index.blade.php

welcome.blade.php

routes

api.php

channels.php

console.php

web.php

> storage

index.blade.php X

Food.php

resources > views > foods > index.blade.php

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta name="viewport" content="width=device-width, initial-scale=1">
5   <link rel="stylesheet" href="/css/app.css">
6   <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js">
7   <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js">
8 </head>
9 <body>
10
11 <div class="container">
12   <h2>Table</h2>
13   <p>The .table-responsive class wraps tables to make them act like responsive, mobile first viewports.
14   <div class="table-responsive">
15     <table class="table">
16       <thead>
```

Modifying the Table

```
<div class="container">
  <h2>Basic Table</h2>
  <p>The .table class adds basic styling</p>
  <table class="table">
    <thead>
      <tr>
        <th>Firstname</th>
        <th>Lastname</th>
        <th>Email</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>John</td>
        <td>Doe</td>
        <td>john@example.com</td>
      </tr>
      <tr>
        <td>Mary</td>
```



```
<thead>
  <tr>
    <th>#</th>
    <th>Name</th>
    <th>Category</th>
    <th>Description</th>
    <th>Nutrition Facts</th>
    <th>Price</th>
  </tr>
</thead>
```

Generating Table Body

Method #1

```
//method #1  
return view('foods.index',compact('allFoods'));
```

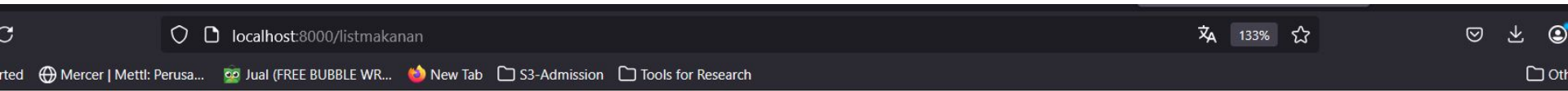
```
<tbody>  
  @foreach($allFoods as $f)  
    <tr>  
      <td>{{ $f->id }}</td>  
      <td>{{ $f->name }}</td>  
      <td>{{ $f->category_id }}</td>  
      <td>{{ $f->description }}</td>  
      <td>{{ $f->nutrition_fact }}</td>  
      <td>{{ $f->price }}</td>  
    </tr>  
  @endforeach  
</tbody>
```

Method #2

```
//method #2  
return view('foods.index',['foods'=>$allFoods]);
```

```
<tbody>  
  @foreach($foods as $f)  
    <tr>  
      <td>{{ $f->id }}</td>  
      <td>{{ $f->name }}</td>  
      <td>{{ $f->category_id }}</td>  
      <td>{{ $f->description }}</td>  
      <td>{{ $f->nutrition_fact }}</td>  
      <td>{{ $f->price }}</td>  
    </tr>  
  @endforeach  
</tbody>
```


Open the Browser to Test the Result



Table

The `.table-responsive` class creates a responsive table which will scroll horizontally on small devices (under 768px). When viewing on anything larger than 768px wide, there is no difference:

#	Name	Category	Description	Nutrition Facts	Price
1	Nasi Merah dengan Ayam Panggang Kecap & Tumis Kangkung	2	Nikmati hidangan sehat dan lezat dengan Nasi Merah yang kaya serat, dipadukan dengan Ayam Panggang Kecap yang manis gurih dan Tumis Kangkung yang segar. Kombinasi sempurna untuk santapan yang mengenyangkan dan bergizi.	Kalori: 400-550 kkal Protein: 30-40 gram Lemak: 15-25 gram Karbohidrat: 50-70 gram Serat: 5-8 gram	35000
2	Nasi Hitam dan Tumis Ca Kailan	2	Nikmati hidangan sehat dan lezat dengan Nasi Hitam yang kaya serat.	Kalori: 400-550 kkal Protein: 30-40 gram Lemak: 15-25 gram Karbohidrat: 50-70 gram Serat: 5-8 gram	30000

Practice #4: Add List View of “Category”, “Customer”, “Transaction”

Working steps

1. Use Controller Resource with Model
2. Write Route with `Route::Resource` to simplify your process
3. Write View with Boostraps Table
4. Write View inside `Resources/Views/` and make a directory first with the Entities Name from Database.

Thank You!