Web Programming Framework

Topic 5:

Another Queries as a Basis for Creating Report

WEEK 05

Informatics Engineering

Universitas Surabaya

Outline

1. Query Raw, Query Builder and Eloquent Query
2. Eloquent Relationship
3. Sub-Query
4. Query Inside Controller and Routing
5. Query for Reporting

Raw Queries

Every Query technique, whether Query Builder or Eloquent Model, can accommodate various SQL syntaxes for data filtering, data aggregation, joining between data, data grouping, and data sorting

1. select * from users where name='andi' and password='andi' □ Filter
2. select count(*) from users where name LIKE '%andi%' □ Agregation & Filter
3. select places.name, count(places.id) from places left join types on types.id = places.id group by places.id order by places.id desc □ Agregation, Join, Group & Sort Data

Example of Filter Data

```php
$users = DB::table('users')
                ->where('votes', '>=', 100)
                ->get();

$users = DB::table('users')
                ->where('votes', '<>', 100)
                ->get();

$users = DB::table('users')
                ->where('name', 'like', 'T%')
                ->get();
```

```php
$users = DB::table('users')->where([
    ['status', '=', '1'],
    ['subscribed', '<>', '1'],
])->get();
```

*Select * from users where status = 1 AND subscribed <> 1*

https://laravel.com/docs/10.x/queries#where-clauses

Example of Data Group / Limit Data

groupBy / having

The `groupBy` and `having` methods may be used to group the query results. The `having` method's signature is similar to that of the `where` method:

```
$users = DB::table('users')
                ->groupBy('account_id')
                ->having('account_id', '>', 100)
                ->get();
```

You may pass multiple arguments to the `groupBy` method to group by multiple columns:

```
$users = DB::table('users')
                ->groupBy('first_name', 'status')
                ->having('account_id', '>', 100)
                ->get();
```

https://laravel.com/docs/10.x/queries#grouping

The `skip` & `take` Methods

You may use the `skip` and `take` methods to limit the number of results returned from the query or to skip a given number of results in the query:

```
$users = DB::table('users')->skip(10)->take(5)->get();
```

Alternatively, you may use the `limit` and `offset` methods. These methods are functionally equivalent to the `take` and `skip` methods, respectively:

```
$users = DB::table('users')
                ->offset(10)
                ->limit(5)
                ->get();
```

Example of Data Aggretion

The query builder also provides a variety of aggregate methods such as `count`, `max`, `min`, `avg`, and `sum`. You may call any of these methods after constructing your query:

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

$price = DB::table('orders')->max('price');
```

Select count(*) from users;

Select max(price) from orders;

You may combine these methods with other clauses:

```
$price = DB::table('orders')
                ->where('finalized', 1)
                ->avg('price');
```

Select avg(price) from orders where finalized = 1;

https://laravel.com/docs/10.x/queries#aggregates

Aggregation with Eloquent Model

You may also use the `count`, `sum`, `max`, and other aggregate methods provided by the query builder. These methods return the appropriate scalar value instead of a full model instance:

```
$count = App\Flight::where('active', 1)->count();

$max = App\Flight::where('active', 1)->max('price');
```

https://laravel.com/docs/10.x/queries#aggregates

Example of Joining Data

```php
$users = DB::table('users')
            ->join('contacts', 'users.id', '=', 'contacts.user_id')
            ->join('orders', 'users.id', '=', 'orders.user_id')
            ->select('users.*', 'contacts.phone', 'orders.price')
            ->get();
```

select users., contacts.phone, orders.price*
from users
inner join contacts on users.id=contacts.user_id
inner join orders on users.id = orders.user_id

```php
$users = DB::table('users')
            ->leftJoin('posts', 'users.id', '=', 'posts.user_id')
            ->get();

$users = DB::table('users')
            ->rightJoin('posts', 'users.id', '=', 'posts.user_id')
            ->get();
```

*select **
from users
left join posts on users.id=posts_user_id

*select **
from users
right join posts on users.id=posts_user_id

https://laravel.com/docs/10.x/queries#joins

Example of Sorting Data

orderBy

The `orderBy` method allows you to sort the result of the query by a given column. The first argument to the `orderBy` method should be the column you wish to sort by, while the second argument controls the direction of the sort and may be either `asc` or `desc`:

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

*select *
from users
order by name DESC*

Query with Eloquent Model

When using Eloquent ORM(Model), Laravel provides various kinds of helpers or functions that are used. The syntax is similar to QueryBuilder.

*Select **
from flights
where active = 1
order by name DESC
LIMIT 10;

```
$flights = App\Flight::where('active', 1)
                ->orderBy('name', 'desc')
                ->take(10)
                ->get();
```

Eloquent : Retrieve Only Single Records

Retrieving Single Models / Aggregates

In addition to retrieving all of the records for a given table, you may also retrieve single records using `find`, `first`, or `firstWhere`. Instead of returning a collection of models, these methods return a single model instance:

```php
// Retrieve a model by its primary key...
$flight = App\Flight::find(1);

// Retrieve the first model matching the query constraints...
$flight = App\Flight::where('active', 1)->first();

// Shorthand for retrieving the first model matching the query constraints...
$flight = App\Flight::firstWhere('active', 1);
```

Query with Raw Statement

The concept of Query using RAW method either with DB or Model has an identical way, namely with the select syntax. Inside the select, there is a string that contains the query.

You can use Prepared Statement concept in PHP (similar like in C# or Java)

Prepared Statement

```php
$users = DB::select('select * from users where active = ?', [1]);

return view('user.index', ['users' => $users]);
```

This syntax is similar to ***“select * from active users = 1”***
but stated in more secure form of query, prepared statements also check data types which can help prevent SQL injection attacks.

Sub-Query

Prepared Statement

Remember about Sub-Query Syntax ?

There are 3 types of Sub-Query :

- ‘select clause’,
- ‘from clause’
- ‘where clause’

SubQuery: Select Clause

Sample Case :

show all Supplier ID and Name, number of stock that is already supplied and show the average number of stock from all supplier

```
/*
  select supplier.id, supplier.name, sum(products.stok) as stok,
        ( select avg(stok) from product ) as rerata
     from suppliers inner join products
        on supplier.id = products.supplier_id
     group by supplier.id, supplier.name
*/
$data = DB::table('suppliers')
      ->join('products','supplier.id','=','products.supplier_id')
      ->select('supplier.id', 'supplier.name',DB::raw('sum(products.stok) as stok'))
      ->addSelect(['rerata' => function ($query) {
            $query->from('products')
                  ->avg('stok');
        }])
      ->groupBy('supplier.id, supplier.name');
      ->get();
```

Another Solution with DB::raw()

```
$data = DB::table('suppliers as s')
        ->join ('products as p','s.id','=','p.supplier_id')
        ->select('s.id','s.nama',DB::raw('sum(p.stok) as stok'))
        ->addSelect(DB::raw('(select avg(stok) from products) as rerata'))
        ->groupBy('s.id','s.nama')
        ->get();
```

SubQuery: From Clause

Sample Case :

Show all User Data and the user's latest post from each user that already published

```
/*
"select * from users inner join
  (
    select user_id, max(created_at) as last_post_created_at
    from posts
    where is_published = true
    group by 'user_id'
  ) latest_posts
  on users.id = latest_posts.user_id
*/
```

Query Builder: From Clause

```php
$latestPosts = DB::table('posts')
            ->select('user_id', DB::raw('MAX(created_at) as last_post_created_at'))
            ->where('is_published', true)
            ->groupBy('user_id');

$users = DB::table('users')
        ->joinSub($latestPosts, 'latest_posts', function ($join) {
            $join->on('users.id', '=', 'latest_posts.user_id');
        })->get();
```

$latestPosts is a query statement to the latest post for each user.

SubQuery: Where Clause with DB Table()

Sample Case :

show id, the full name of employee, and salary of the employee that has a salary above the average of all salaries in this company

```
/*SELECT employee_id,first_name,last_name,salary
 FROM employees WHERE salary >
    (SELECT AVG(SALARY) FROM employees);
*/
$data = DB::table('employees')
            ->select('employee_id,first_name,last_name,salary')
            ->where('salary','>',function($query){
              $query->from('employess')
                ->avg('salary');
            })
            ->get();
```

SubQuery: Where Clause with Eloquent

Sample Case :

show id, the full name of employee, and salary of the employee that has a salary above the average of all salaries in this company

```
/*SELECT employee_id,first_name,last_name,salary
 FROM employees WHERE salary >
    (SELECT AVG(SALARY) FROM employees);
*/
$data = Employee::select('employee_id,first_name,last_name,salary')
            ->where('salary','>',function($query){
              $query->from('employess')
                ->avg('salary');
            })->get();
```

Eloquent Relationship
Case : Post and Comment

Eloquent Relationship

One of the benefits of using the Eloquent Model is the simplification of your Query statements.

To realize the benefit, the developer must assign the `relationship` in the **early stage of development** (after the creation of the model or after changes to the table structure).

There are 2 types of basic relationship:
- One-to-many (similar with many-to-one in implementation phase)
- Many-to-Many

This implementation is inside of each Model Class.

Note:
You should create a ModelClass first with
Example*: php artisan make:model Product*

One to Many : hasMany

One-to-many

There are 2 syntaxes:

- **`hasMany` keyword**
- `belongsTo` keyword

If you have a custom name of Identifier (ID/Primary Key) or Foreign Key(FK) columns, You may also override the foreign and local keys by passing additional arguments to the hasMany.

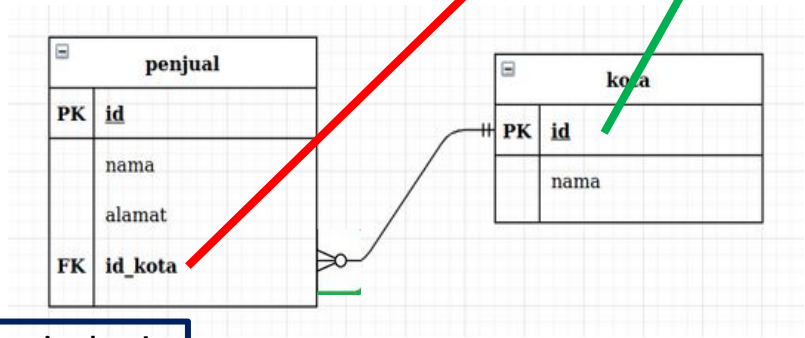
As an example below, the `kota`, `penjual` table, and `id_kota` column isn't a standard name of Laravel Eloquent Model. We should define it in the following way(use overriding if necessary).

Inside the Kota Class

```
Class Kota extends Model {
  protected $table = 'kota';

  public function penjuals() {
    return $this->hasMany(Penjual::class, 'id_kota', 'id');
  }

}
```



Pay attention to the position of the columns (red and green), don't get them mixed up !

One to Many : hasMany (2)

The usage of Eloquent's `hasMany` statement is the same as the picture above.

This syntax can implement inside the Controller.

1. Define or import ModelClass first
2. Call your Model and call the function without a bracket ()
3. **$comments will assign with list of Comment based on Post that has id=1**

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\HasMany;

class Post extends Model
{
    /**
     * Get the comments for the blog post.
     */
    public function comments(): HasMany
    {
        return $this->hasMany(Comment::class);
    }
}
```

```php
use App\Models\Post;

$comments = Post::find(1)->comments;

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

Example : `hasMany` keyword

One to Many : belongsTo

One-to-many

There are 2 syntaxes:

- `hasMany` keyword
- **`belongsTo` keyword**

If you have a custom name of Identifier (ID/Primary Key) or Foreign Key(FK) columns, You may also override the foreign and local keys by passing additional arguments to the belongsTo.

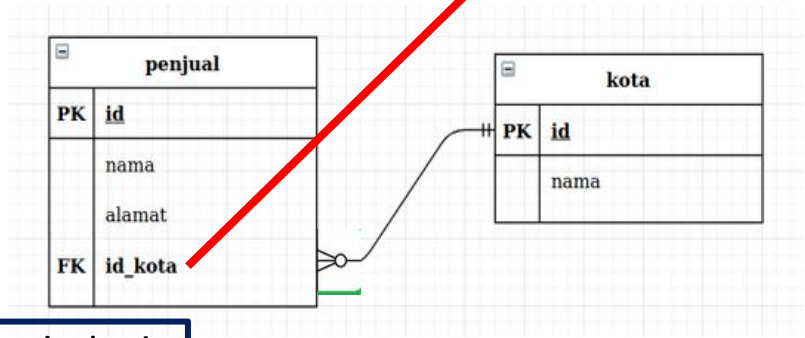
As an example below, the `kota`, `penjual` table, and `id_kota` column isn't a standard name of Laravel Eloquent Model. We should define it in the following way(use overriding if necessary).

Inside the Penjual Class

```
Class Penjual extends Model {
  protected $table = 'penjual';

  public function kota() {
    return $this->belongsTo(Kota::class, 'd_kota');
  }

}
```



Pay attention to the position of the columns (red and green), don't get them mixed up !

One to Many : belongsTo (2)

The usage of Eloquent's `belongsTo` statement is the same as the picture above.
This syntax can implement inside the Controller.

1. Define or import ModelClass first
2. $comment has a Post object that has id=1
3. **You can get the title of post based on number #3, you can type with this format:**

$comment->your_function>your_field_of_object

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;

class Comment extends Model
{
    /**
     * Get the post that owns the comment.
     */
    public function post(): BelongsTo
    {
        return $this->belongsTo(Post::class);
    }
}
```

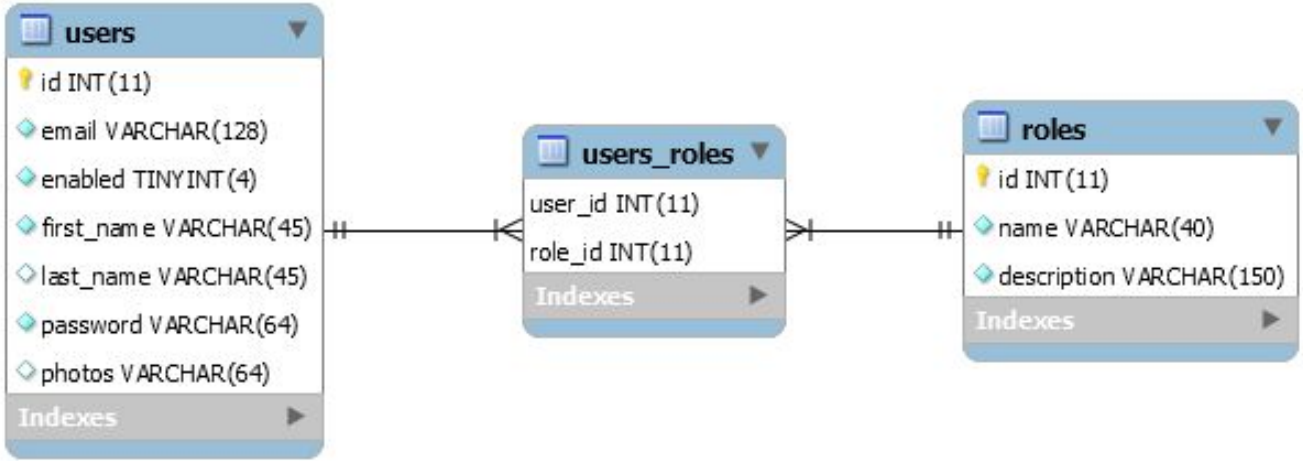
```php
use App\Models\Comment;

$comment = Comment::find(1);

return $comment->post->title;
```

Example : `belongsTo` keyword

Eloquent Relationship : Many to Many



This example is the many-to-many scenario with the standard name of Laravel Eloquent Model

Criteria:

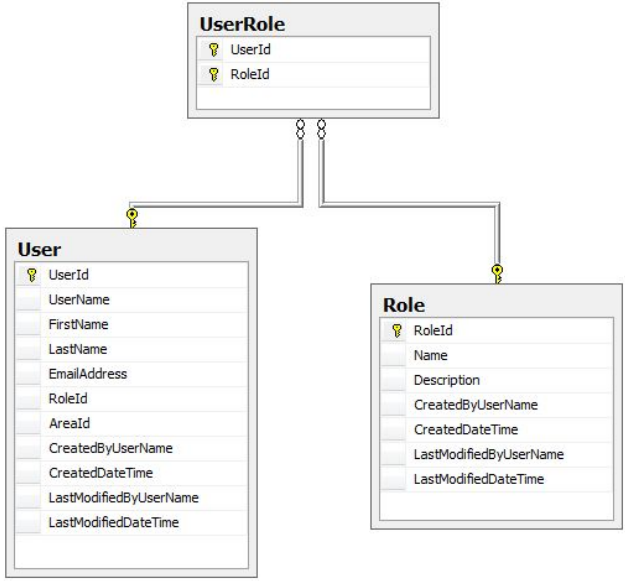
1. Each tables use plural nouns.
2. Intermediated table use singular noun of each many-to-many and order with alphabetic

Many to Many : hasMany

There are 1 syntaxes: `belongsToMany` keyword

If you have a custom name of Identifier (ID/Primary key) and Foreign_key, You may also override the foreign and local keys by passing additional arguments to the belongsToMany.

As an example, the `user`, `role`, `UserRole` table, and `UserId`, `RoleId` column isn't a standard name of Laravel Eloquent Model. You must define with




```
Class User extends Model {
  protected $table = 'User';

  public function roles() {
    return $this->belongsToMany(Role::class,
            'UserRole',
            `UserId`,`RoleId`);
  }
}
```

```
Class Role extends Model {
  protected $table = 'Role';

  public function users() {
    return $this->belongsToMany(User::class,
            'UserRole',
            `RoleId`,`UserId`);
  }
}
```

Many to Many : Get Value

In this example, we want to get the data creation time of role_user data.

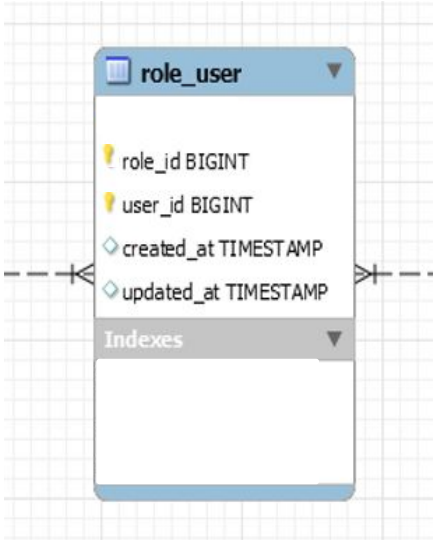
So, we must get value of `created_at` from specific role
You can use ->pivot->

Example
Retrieving creation time data from role_user table for user with user_id=1 and role_id=2

```php
use App\Models\User;

$user = User::find(1);

foreach ($user->roles as $role) {
    echo $role->pivot->created_at;
}
```

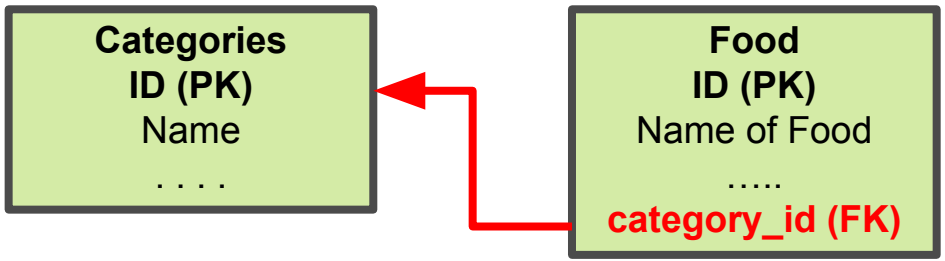


#1 Practice
Our Case Study

Assign your Tickets with Place

Based on our case study, we know that Food as one category but one category can have multiple variant of food.

So the Data Design shown as follow:



One To Many : belongsTo

```php
use Illuminate\Database\Eloquent\Relations\BelongsTo;

class Food extends Model
{
    use HasFactory;

    protected $table = 'foods';
    protected $primaryKey = 'id';
    public $timestamps = true;

    public function category(): BelongsTo{
        return $this->belongsTo(Category::class,'category_id');
    }

```

Discussion in your class:
Can you explain why this example uses `belongsTo` and why does the example decide to give 'category' as function name?

One To Many : hasMany

```php
use Illuminate\Database\Eloquent\Relations\HasMany;


class Category extends Model
{
    use HasFactory;
    protected $table = 'categories';

    public function foods(): HasMany
    {
        return $this->hasMany(Food::class,'category_id','id');
    }
}
```

Discussion in your class:
Can you explain why this example uses `hasMany` and use “foods” in function name?

Show specific Food : Try it !

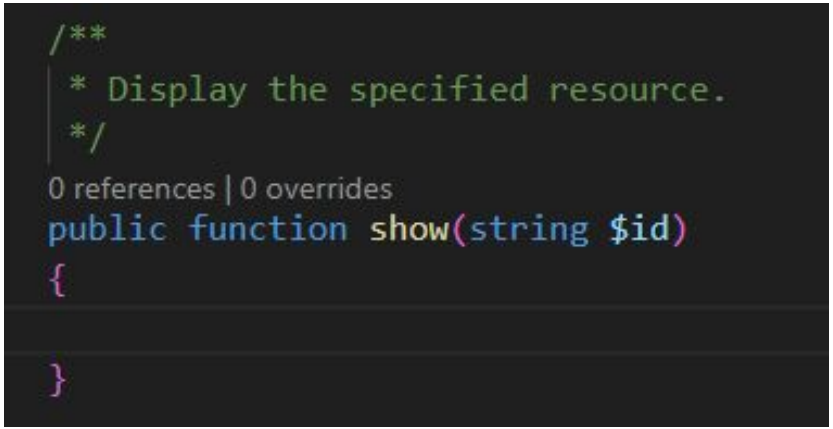
Please open your FoodController !

If you don't have Food Controller, you can run this command:
php artisan make:controller FoodController --resource

show() function inside your Controller is used to inform the details of your object (in this case is Product object)

show() function always required 1 parameter ($id) that represented the value of ID product.

With *php artisan route:list* you can see the required parameter routing and the route name below.



Debugging using dd(dump and die)

DD will '**dump**' your query result and make the apps temporarily '**die**'.

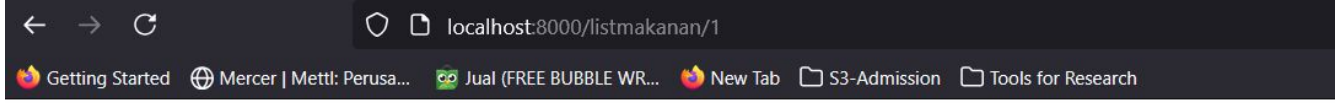
```
app > Http > Controllers > FoodController.php > FoodControlle
 9   class FoodController extends Controller
57       }
58
59       /**
60        * Display the specified resource.
61        */
62       public function show(Food $food)
63       {
64           dd($food);
65       }
66
```

Insert dd function to inspect the specific food data

```
PS C:\laragon\www\example-app> php artisan route:list

GET|HEAD    listmakanan/{listmakanan} ........ listmakanan.show › FoodC
PUT|PATCH   listmakanan/{listmakanan} .... listmakanan.update › FoodCon
DELETE      listmakanan/{listmakanan} .. listmakanan.destroy › FoodCont
GET|HEAD    listmakanan/{listmakanan}/edit ... listmakanan.edit › FoodC
```

Use route list to know the URL format

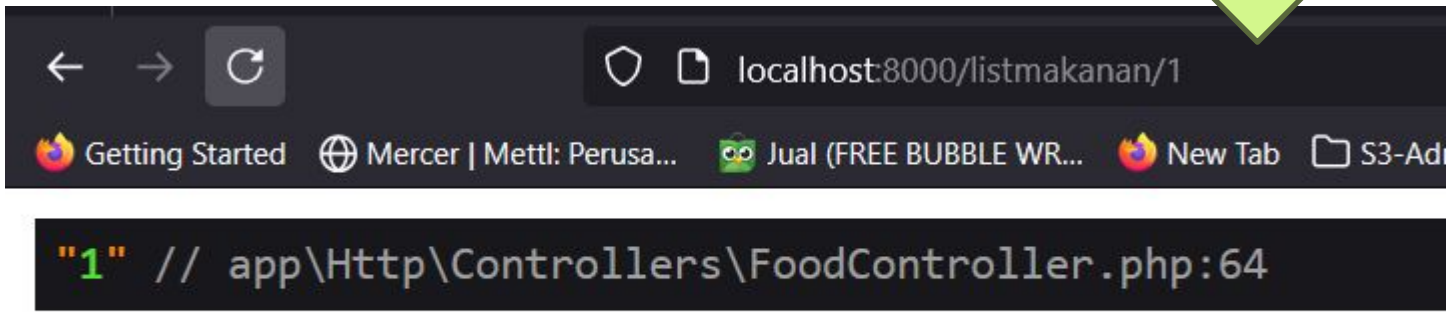



```
App\Models\Food {#3020 ▼ // app\Http\Controllers\FoodController.php:64
  #connection: null
  #table: "foods"
  #primaryKey: "id"
  #keyType: "int"
  +incrementing: true
  #with: []
  #withCount: []
  +preventsLazyLoading: false
  #perPage: 15
  +exists: false
  +wasRecentlyCreated: false
  #escapeWhenCastingToString: false
  #attributes: []
  #original: []
  #changes: []
  #casts: []
  #classCastCache: []
  #attributeCastCache: []
  #dateFormat: null
  #appends: []
  #dispatchesEvents: []
  #observables: []
  #relations: []
  #touches: []
  +timestamps: true
  +usesUniqueIds: false
  #hidden: []
  #visible: []
```

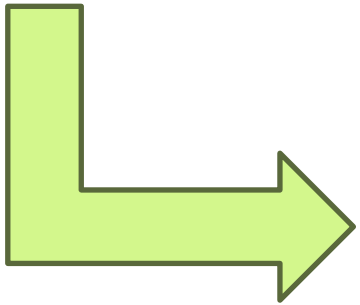
There is no data inside attributes

Remove Model inside parameter function

```
61      */
62     public function show($food)
63     {
64         dd($food);
65     }
66
```



```php
public function show($food)
{
    $current_food = Food::find($food);
    dd($current_food);
}
```



localhost:8000/listmakanan/1

Getting Started | Mercer | Mettl: Perusa... | Jual (FREE BUBBLE WR... | New Tab | S3-Admission | Tools for Research

```
App\Models\Food {#3011 ▼ // app\Http\Controllers\FoodController.php:65
  #connection: "mysql"
  #table: "foods"
  #primaryKey: "id"
  #keyType: "int"
  +incrementing: true
  #with: []
  #withCount: []
  +preventsLazyLoading: false
  #perPage: 15
  +exists: true
  +wasRecentlyCreated: false
  #escapeWhenCastingToString: false
  #attributes: array:8 [▼
    "id" => 1
    "created_at" => null
    "updated_at" => null
    "name" => "Nasi Merah dengan Ayam Panggang Kecap & Tumis Kangkung"
    "nutrition_fact" => """
      Kalori: 400-550 kkal\n
      \t\t                    Protein: 30-40 gram\n
      \t\t                    Lemak: 15-25 gram\n
      \t\t                    Karbohidrat: 50-70 gram\n
      \t\t                    Serat: 5-8 gram
      """
    "description" => """
      Nikmati hidangan sehat dan lezat \n

                              dengan Nasi Merah yang kaya serat, dipadukan dengan Ayar
```

Debugging using dd(dump and die)

```blade
<div class="container">
    <h2>Ticket Table</h2>
    <table class="table">
        <thead>
            <tr>
                <th>Ticket ID</th>
                <th>Report</th>
                <th>Place Name</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>{{ $data->id }}</td>
                <td>{{ $data->report }}</td>
                <td>{{ $data->places->name }}</td>
            </tr>
        </tbody>
    </table>
</div>
```

We can see the `Magic` statement

$data is the variable sent from controller

->places is function of Eloquent Relationship in the Places Model.

->name is an attributed based on our database

Example : http://127.0.0.1:8000/tickets/show/1

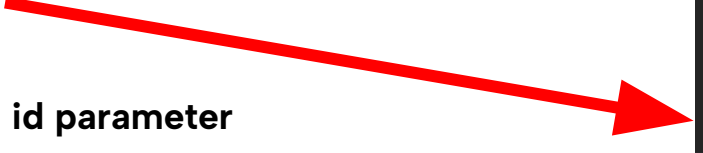
Ticket Table

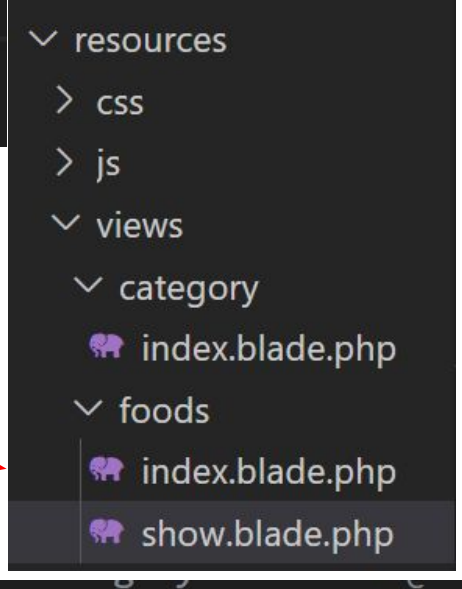
Ticket ID	Report	Place Name
1	CnMF2UGMnKPunofgDUVCIpZ2uC1s5jyotNtAcH9k	Z3A1ZVmGidkC4i38Visq

Show Implementation

```php
public function show($food)
{
    $current_food = Food::find($food);
    return view("foods.show",compact("current_food"));
}
```

Don’t forget to create blade file in resource/view/**product** directory, name it **show.blade.php**





Ensure the route (show) passes with id parameter

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

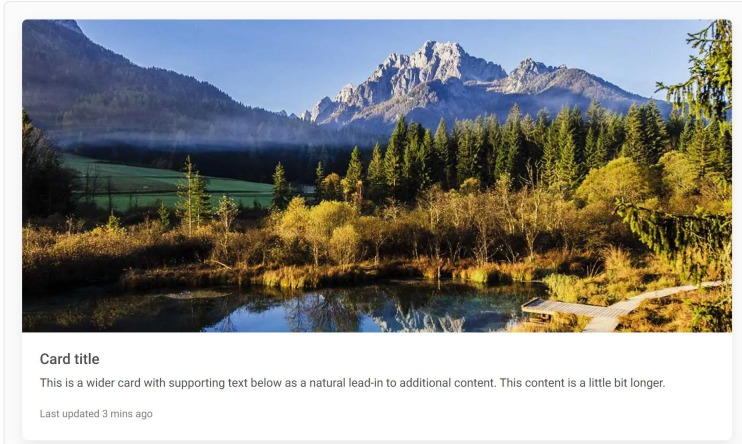
```
GET|HEAD        listkategori/{listkategori} . listkategori.show › CategoryController@s
```

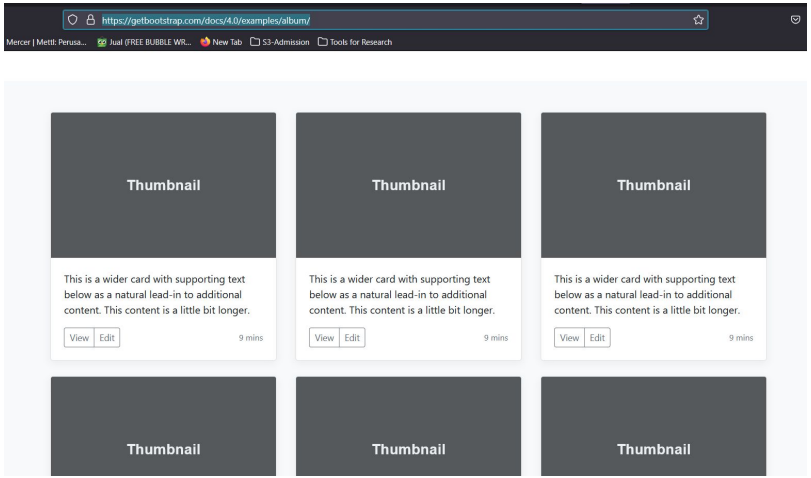
Construction View

Use Boostrap View Template. For catalog or list of data, you can use this example:

https://www.w3schools.com/bootstrap4/bootstrap_cards.asp
https://mdbootstrap.com/docs/standard/components/cards/
https://mdbootstrap.com/docs/standard/extended/gallery/
https://getbootstrap.com/docs/4.0/examples/album/

For the detail page, we recommend use this:
https://bootstrapexamples.com/@andreas-muller/product-details-page

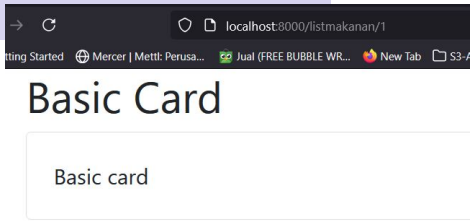




1. Go to example of Boostrap (https://www.w3schools.com/bootstrap4/tryit.asp?filename=trybs_card&stacked=h)
2. Construct HTML, Import CSS/JS from Boostrap
3. Use Basic Card first based on the example

```
<div class="container">
  <h2>Basic Card</h2>
  <div class="card">
    <div class="card-body">Basic card</div>
  </div>
</div>
```

4. Run project first



5. Change the body with another example (https://getbootstrap.com/docs/4.0/examples/album/)

Basic Card

This is a wider card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

View Edit 9 mins

Reports

Reports in an Information System

Reports in an information system are the result of analysis of data in the database.

Data analysis can use previous query concepts such as

- ✔ Data Aggregation
- ✔ Join Data
- ✔ Sort data
- ✔ Filter data

Laravel gives you the freedom to create custom reporting on URLs.

Create Custom URL

Please re-open our Routing Material (from Week 1 or Week 2)
https://laravel.com/docs/10.x/routing#basic-routing

Form Report usually uses the **GET HTTP method** because the report function predominately retrieves data from the database and displays it in the view. So Routing (in routes/web.php) which we will often use has a syntax format

```
Route::get('/user', 'UserController@index');
```

#2 Practice

Practice #2

Please create a new page for showing place list with total category.

Hint :

1. Add custom URL in web.php (url : BASE_URL/category/showTotalFoods
2. Create function in CategoryController
3. In query, use count() to get total category
4. If there is no food in specific category, system will set total food = 0
5. Create view with name totalfood.blade.php inside category folder

Thank You!