NOW IN TECHNICOLOR!
From
FORM-TO-TABLE
With
LARAVEL

Level 5

Relationships

Farms, Pivots, & CRUD

Farms, Pivots, & CRUD

Once we have built out our Markets, we need to add some Farms to the Markets.

Look at Farms & CRUD

Establish a relationship between Farms & Markets

Create a pivot table for database associations

Building out the Farm

Many steps that we took for the Markets can be duplicated for Farms.

Create a Farm Model with Artisan

Create the migration with the same fields as our Market

Create a Farm Controller

Create the index, show, create, and save methods on the controller

FROM FORM TO TABLE WITH LARAVEL

Reviewing Our Controller

Our farm controller should have the following methods:

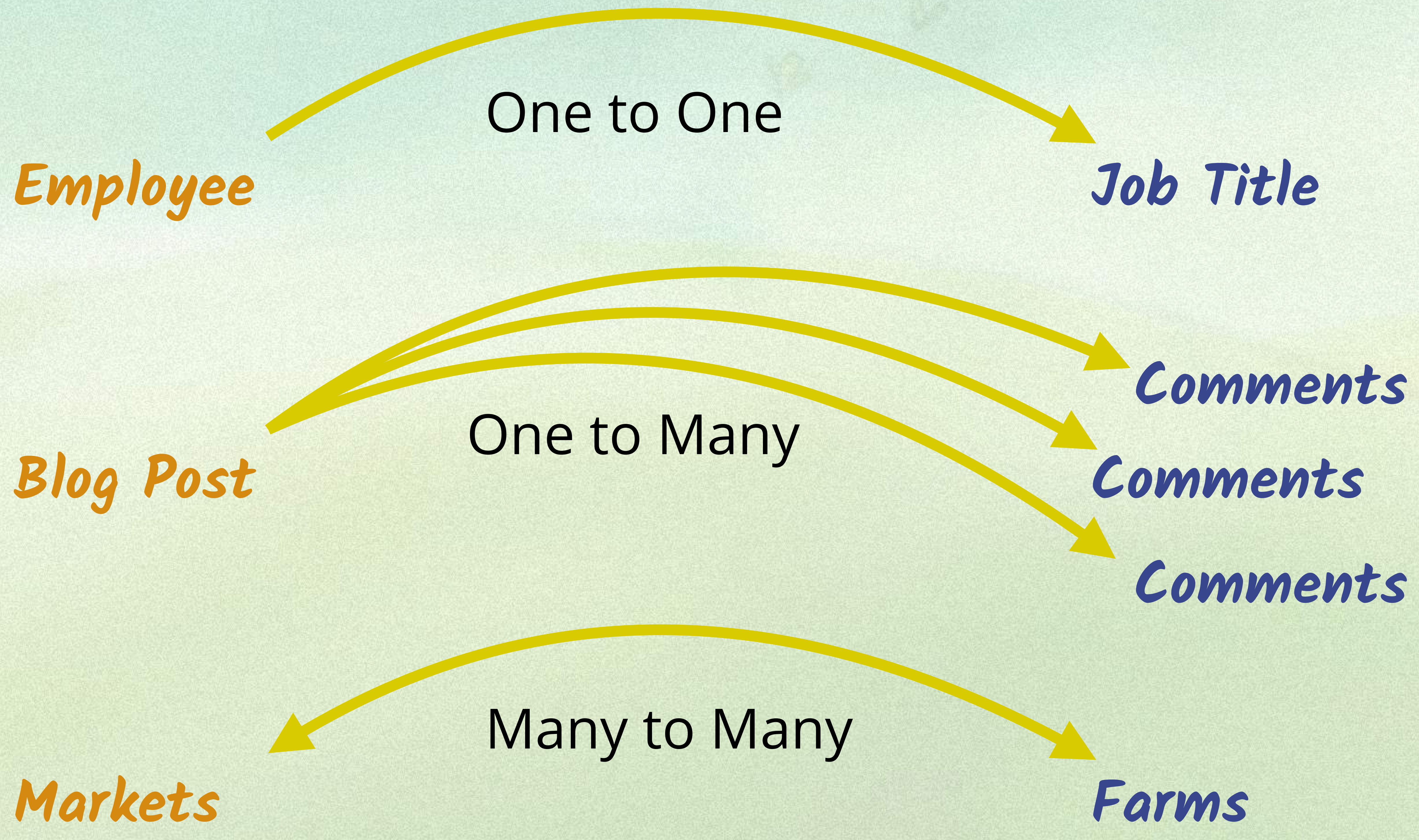
app/Http/Controllers/FarmController.php

```php
<?php
class FarmController extends Controller
{
    public function index()
    …
    public function show(Farm $farm)
    …
    public function create()
    …
    public function save(Request $request)
    …
…
```

FROM FORM TO TABLE WITH LARAVEL

Linking Our Models Together

Relationships within Laravel help tie our models together.



FROM FORM TO TABLE WITH LARAVEL

Defining Our Many to Many

A pivot table links our farms table to our markets table by establishing a link through the id.

farms

id	name
1	Thomas Farms
2	Slow Foods
3	Pig Party
4	Sprouts Farm

markets

id	name
1	Audubon Market
2	Orlando Farmers Market
3	Maitland Farmers Market
4	Cair Paravel Market

farm_market

farm_id	market_id
1	4
1	2
2	1
3	1

Pivot Table!

FROM FORM TO TABLE WITH LARAVEL

Create a Migration for a Pivot Table

terminal.app

```
~/market $  php artisan make:migration create_farm_market_pivot_table
            --create farm_market


Migration created successfully.
Created Migration: 2017_02_24_193452_create_farm_market_pivot_table

~/market $
```

the `--create` flag will define the name of the table and some basic fields too

verbose naming helps us understand what our task for each migration is

FROM FORM TO TABLE WITH LARAVEL

Add the Pivot Columns to the Migration

database/migrations/2017_02_24_193452_create_farm_market_pivot_table.php

```php
<?php
class CreateFarmMarketPivotTable extends Migration
{
    public function up()
    {
      Schema::create('farm_market', function (Blueprint $table) {
        $table->timestamps();
      });

    } …
```

the table name must be a combo of both tables in alphabetical order

This code is pre-generated by artisan, but we need to add more!

Add the Pivot Columns to the Migration

database/migrations/2017_02_24_193452_create_farm_market_pivot_table.php

```
<?php
class CreateFarmMarketPivotTable extends Migration
{
    public function up()
    {
      Schema::create('farm_market', function (Blueprint $table) {
        $table->integer('farm_id')->unsigned()->index();
        $table->foreign('farm_id')->references('id')
            ->on('farms')->onDelete('cascade');

        $table->integer('market_id')->unsigned()->index();
        $table->foreign('market_id')->references('id')
            ->on('markets')->onDelete('cascade');

        $table->timestamps();
      });

    } …
```

the table name must be a combo of both tables in alphabetical order

This line deletes references in our pivot table if the market gets deleted

Add the References to the Models

In the Market and Farm Models we will create a new action to return the relationships.

app/Market.php

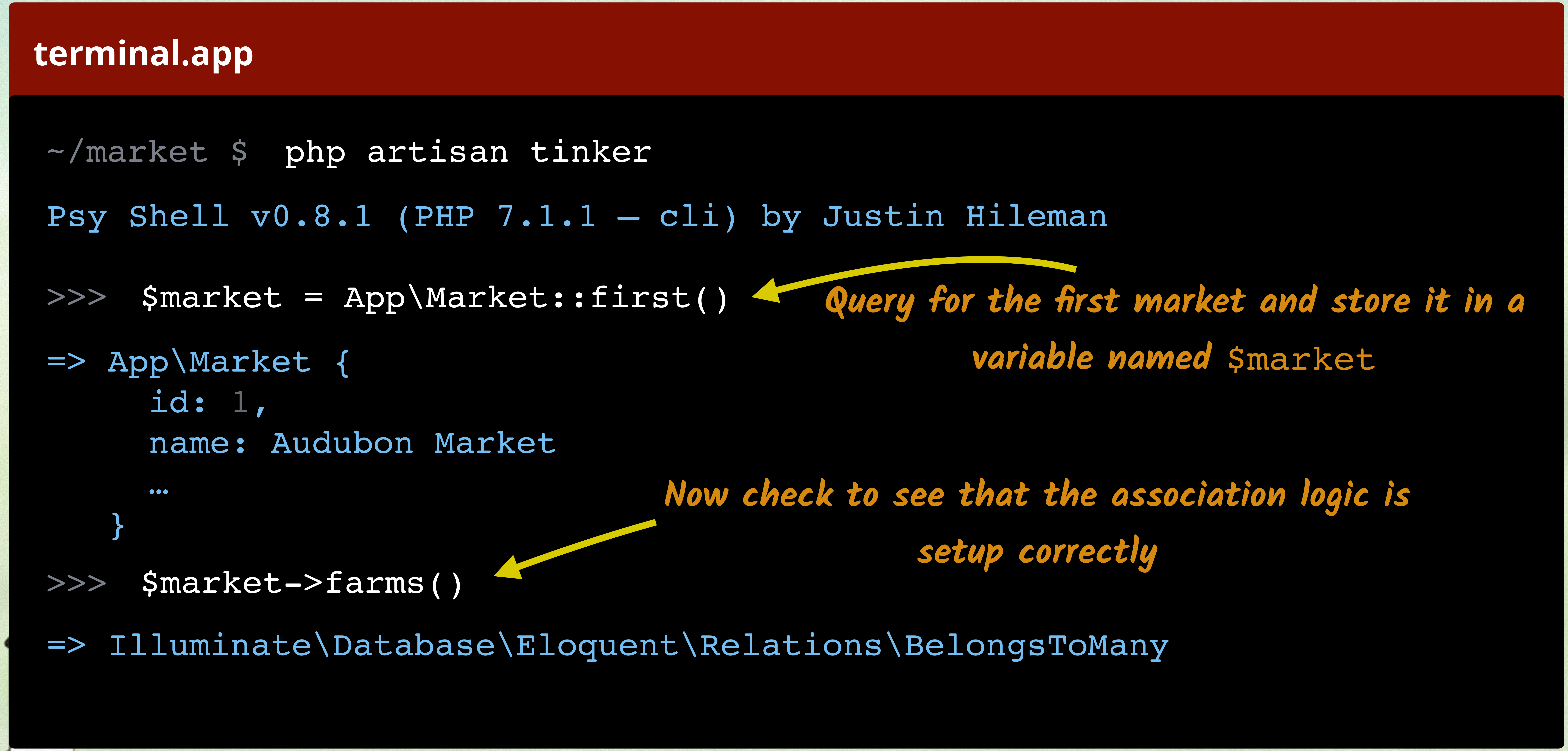
```php
<?php
class Market extends Model
{
   public function farms()
   {
     return $this->belongsToMany('App\Farm')->withTimestamps();
```

app/Farm.php

```php
<?php
class Farm extends Model
{
   public function markets()
   {
     return $this->belongsToMany('App\Market')->withTimestamps();
```

Using Tinker to Modify Our Relationships

Using Tinker, we will tie a single farm to a single market in a few steps.



terminal.app

```
~/market $  php artisan tinker
Psy Shell v0.8.1 (PHP 7.1.1 — cli) by Justin Hileman

>>>  $market = App\Market::first()
=> App\Market {
      id: 1,
      name: Audubon Market
      …
   }
>>>  $market->farms()
=> Illuminate\Database\Eloquent\Relations\BelongsToMany
```

Query for the first market and store it in a variable named $market

Now check to see that the association logic is setup correctly

Using Tinker to Modify Our Relationships

Query for a single farm that we will associate to the market object.

terminal.app

```
>>>  $market->farms()
=> Illuminate\Database\Eloquent\Relations\BelongsToMany

>>>  $market->farms()->first()

=> null

>>>  $farm = App\Farm::first()

=> App\Farm {
     id: 1,
     name: "Thomasville Farms", …
```

The relationship is setup, but is empty!

Query for the first farm and store it in a variable named `$farm`

FROM FORM TO TABLE WITH LARAVEL

Using Tinker to Modify Our Relationships

Using Tinker, we can interact with our development environment, DB, & Model.

terminal.app

```
>>>  $market->farms()->save($farm)
=> App\Farm {
       id: 1,
       name: "Thomasville Farms",…

>>>  $market->farms()->count()
=> 1

>>>  $farm->markets()->count()
=> 1
```

Using the farms method on the market model and then the save method, we can associate the `$farm` to the `$market`

If we test our market and farm association in either direction the result should be the same!

Farms, Pivots, & CRUD Overview

What have we accomplished in this section with our Farms and relationships?

Created a Farm Model and generic CRUD methods

Established a relationship between Farms & Markets

Created a pivot table for database associations

Used Tinker to search and associate a farm to a market in both directions

FROM FORM TO TABLE WITH LARAVEL

Level 5

Relationships

Association Through Forms

Association Through Forms

Now that we have Farms and Markets, lets connect them with Forms.

Create an Update method to add the Markets to the Farm

Create a Form for updating our Farms

Use the Form to associate Markets to the Farm

Keep these associations in sync through our controller

A Default Edit Action for Farms

Without a relationship, the farm edit action will query the farm id and pass along the object.

app/Http/Controllers/FarmController.php

```php
<?php
class FarmController extends Controller
{
    public function edit(Farm $farm)
    {
      return view('farms.edit', ['farm'=> $farm]);
    }
…
```

Modifying the Edit Action for Relationships

We need to add markets to our query and pass along both the farm and markets.

app/Http/Controllers/FarmController.php

```
<?php
class FarmController extends Controller
{
    public function edit(Farm $farm)
    {

      $markets = App\Market::get()->pluck('name', 'id')->sortBy('name');


    }
…
```

`get` *works much like* `all()`

`pluck` *lets us define what fields we want in our collection results and sorted by name*

FROM FORM TO TABLE WITH LARAVEL

Modifying the Edit Action for Relationships

We need to add markets to our query and pass along both the farm and markets.

app/Http/Controllers/FarmController.php

```php
<?php
class FarmController extends Controller
{
    public function edit(Farm $farm)
    {

       $markets = App\Market::get()->pluck('name', 'id')->sortBy('name');
       return view('farms.edit', compact('farm', 'markets'));

    }
...
```

```php
compact('farm', 'markets')
```

is a shorthand way of writing

```php
['farm' => $farm, 'markets' => $markets]
```

FROM FORM TO TABLE WITH LARAVEL

Using the Edit Form for Associations

Using a for loop, we can list out markets to associate with a farm.

resources/views/farms/edit.blade.php

```
<form action="{{ route('markets.update', $farm) }}" method="post">
  {{ method_field('patch') }}
…
</form>
```

`method_field` *with patch lets the laravel routes know we will be updating an existing object or market*

Using the Edit Form for Associations

Using a for loop, we can list out markets to associate with a farm.

resources/views/markets/edit.blade.php

```
<form action="{{ route('markets.update', $market) }}" method="post">
  {{ method_field('patch') }}



  @foreach ($markets as $id => $market)

  @endforeach
```

when we used `pluck()` *we created a collection where the key is the id of the market and the* `$market->name` *is the calue*

```
…
</form>
```

Using the Edit Form for Associations

Using a for loop, we can list out markets to associate with a farm.

resources/views/markets/edit.blade.php

```
<form action="{{ route('markets.update', $market) }}" method="post">
  {{ method_field('patch') }}

  @foreach ($markets as $id => $market)
  <div>
    <label for="{{ $market }}">
      <input type="checkbox" name="markets[]" value="{{ $id }}">
        {{ $market }}
    </label>
  </div>
  @endforeach

…
</form>
```

markets[] will return an array of checked values to our request object in the controller

here, we will create a checkbox with the name of the market and the value of the market id

Using the Edit Form for Associations

Using a for loop, we can list out markets to associate with a farm.

resources/views/markets/edit.blade.php

```
<form action="{{ route('markets.update', $market) }}" method="post">
  {{ method_field('patch') }}
  @foreach ($markets as $id => $market)
  <div>
    <label for="{{ $market }}">
      <input type="checkbox" name="markets[]" value="{{ $id }}">

        {{ $farm->markets()
                ->allRelatedIds()
                ->contains($id) ? "checked" : "" }}>
        {{ $market }}
    </label>
  </div>
… @endforeach
</form>
```

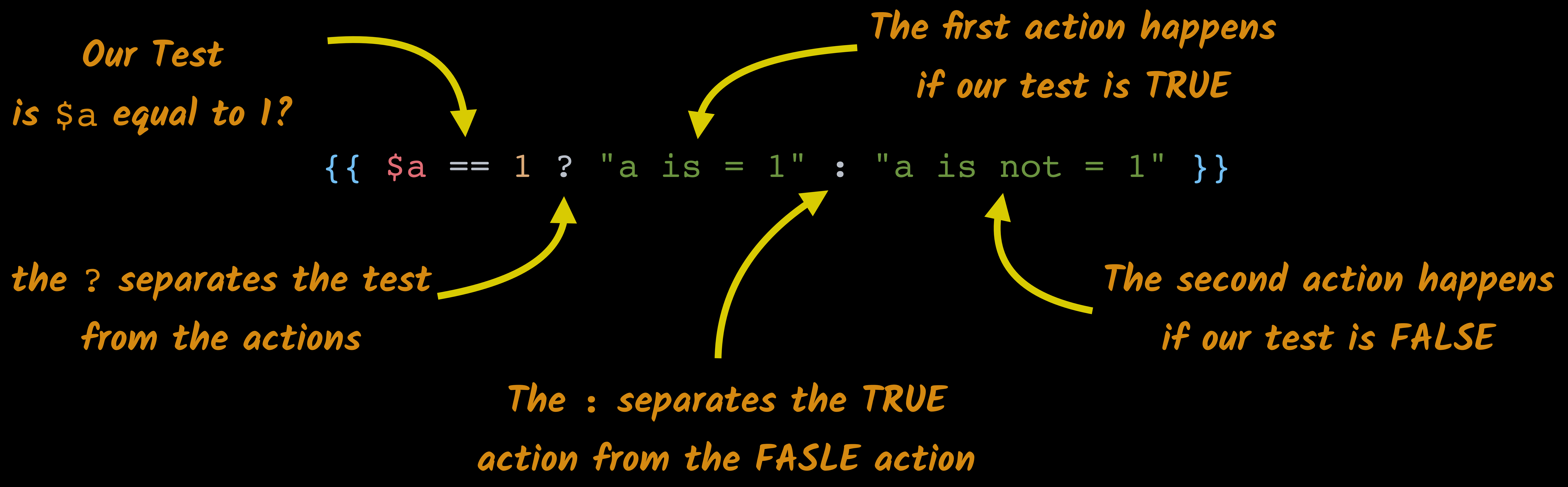
this ternary if statement will return the text `'checked'` *if our* `farm->markets` *has the* `$id` *present in a collection*

Understanding a Ternary IF Statement

Using a Ternary IF statement, we can echo a simple string for our checkbox.

resources/views/markets/edit.blade.php

```
{{ $farm->markets()
        ->allRelatedIds()
        ->contains($id) ? "checked" : "" }}
```



```
{{ $a == 1 ? "a is = 1" : "a is not = 1" }}
```

A Generic Update Action for Farms

Here, we are only updating our farm and not concerned with market relationships.

app/Http/Controllers/FarmController.php

```
<?php
class FarmController extends Controller
{
    public function update(Request $request)
    {

      $farm->update($request->all());
      return redirect('farms');

    }
...
```

we need to add code here to store our market associations

FROM FORM TO TABLE WITH LARAVEL

Using the Update Action to Add Markets

Adding market relationships is a simple process using the sync method.

app/Http/Controllers/FarmController.php

```php
<?php
class FarmController extends Controller
{
    public function update(Request $request)
    {

       $farm->update($request->all());
       $farm->markets()->sync($request->markets);
       return redirect('farms');

    }
...
```

FROM FORM TO TABLE WITH LARAVEL

NOW IN TECHNICOLOR!
From
FORM-TO-TABLE
With
LARAVEL