Atividade Prática 3 - Models, Seeds e Factories no Laravel

Objetivo

O objetivo desta atividade é entender como criar modelos, configurar relacionamentos, e utilizar seeds e factories no Laravel para popular o banco de dados com dados predefinidos e dados fake. Você aprenderá a criar os modelos necessários, configurar relacionamentos, criar seeds para preencher tabelas com dados iniciais e usar factories para gerar uma grande quantidade de registros com dados aleatórios.

Tarefa: Popular o Sistema de Gerenciamento de Biblioteca

Nesta atividade, você irá criar os modelos para `Book`, `Category` , `Publisher` e `Author`, configurar os relacionamentos entre eles, criar seeds para preencher a tabela de categorias com dados predefinidos e usar factories para gerar milhares de registros de livros e autores utilizando dados fake.

Instruções

Passo 1: Criar os Modelos para Livros, Categorias e Autores

Você já deve ter criado as migrations na “Atividade 2 - Migrations no Laravel”, mas aproveite para verificar se está tudo correto com os arquivos de migrations.

1.1 Verifique o arquivo de migration correspondente ao modelo `Category` em `database/migrations/{timestamp}_create_categories_table.php`.

```php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateCategoriesTable extends Migration
{
    public function up()
    {
        Schema::create('categories', function (Blueprint $table) {
            $table->id();
            $table->string('name')->unique();
            $table->timestamps();
        });
    }
```

```
    public function down()
    {
        Schema::dropIfExists('categories');
    }
}
```

1.2 Verifique o arquivo de migration correspondente ao modelo `Author` em `database/migrations/{timestamp}_create_authors_table.php`.

```
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateAuthorsTable extends Migration
{
    public function up()
    {
        Schema::create('authors', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->date('birth_date')->nullable();
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('authors');
    }
}
```

1.3 Verifique o arquivo de migration correspondente ao modelo `Publisher` em `database/migrations/{timestamp}_create_publishers_table.php`.

```
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreatePublishersTable extends Migration
{
    public function up()
```

```
    {
        Schema::create('publishers', function (Blueprint $table) {
            $table->id();
            $table->string('name')->unique();
            $table->string('address')->nullable();
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('publishers');
    }
}
```

1.4 Verifique o arquivo de migration correspondente ao modelo `Book` em `database/migrations/{timestamp}_create_books_table.php`.*(acrescentei um campo published_year direto na migration de books, mas na atividade 2 fizemos um update da tabela books)*

```
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;


public function up()
{
    Schema::create('books', function (Blueprint $table) {
        $table->id();
        $table->string('title');
   $table->foreignId('author_id')->constrained()->onDelete('cascade');
   $table->foreignId('category_id')->constrained()->onDelete('cascade');
  $table->foreignId('publisher_id')->constrained()->onDelete('cascade');
        $table->integer('published_year')->nullable();
        $table->timestamps();

    });
}

public function down()
{
    Schema::dropIfExists('books');
}
```

1.5 Caso tenha feito alguma alteração então execute as migrations com a opção *refresh* para limpar o banco de dados e criar as tabelas novamente.

```
php artisan migrate:refresh
```

Passo 2: Criar os Modelos para Livros, Categorias e Autores

2.1 Na linha de comando, execute os comandos para criar os modelos `Category`, `Author, Publisher` e `Book`, juntamente com as migrations correspondentes.

```
php artisan make:model Category
php artisan make:model Author
php artisan make:model Publisher
php artisan make:model Book
```

Passo 3: Configurar Relacionamentos nos Modelos

3.1 Abra o modelo `Category` em `app/Models/Category.php` e configure o relacionamento `hasMany` com o modelo `Book`.

```php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Category extends Model
{
    use HasFactory;

    protected $fillable = ['name'];

    public function books()
    {
        return $this->hasMany(Book::class);
    }
```

```
}
```

3.2 Abra o modelo `Author` em `app/Models/Author.php` e configure o relacionamento `hasMany` com o modelo `Book`.

```
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Author extends Model
{
    use HasFactory;

    protected $fillable = ['name', 'birth_date', 'email'];

    public function books()
    {
        return $this->hasMany(Book::class);
    }
}
```

3.3 Abra o modelo `Publisher` em `app/Models/Publisher.php` e configure o relacionamento `hasMany` com o modelo `Book`.

```
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Publisher extends Model
{
    use HasFactory;

    protected $fillable = ['name', 'address'];

    public function books()
```

```
    {
        return $this->hasMany(Book::class);
    }
}
```

3.4 Abra o modelo `Book` em `app/Models/Book.php` e configure os relacionamentos `belongsTo` com os modelos `Author` e `Category`.

```
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Book extends Model
{
    use HasFactory;

    protected $fillable = ['title', 'author_id', 'category_id',
'publisher_id', 'published_year'];

    public function author()
    {
        return $this->belongsTo(Author::class);
    }

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

    public function publisher()
    {
        return $this->belongsTo(Publisher::class);
    }

}
```

Passo 4: Criar Seeds para Categorias

4.1 Na linha de comando, execute o comando para criar um seeder para a tabela de categorias.

```
php artisan make:seeder CategorySeeder
```

4.2 Abra o arquivo gerado na pasta `database/seeders/CategorySeeder.php`.

4.3 Edite o arquivo `CategorySeeder.php` para incluir as categorias predefinidas.

```php
namespace Database\Seeders;

use Illuminate\Database\Seeder;
use App\Models\Category;

class CategorySeeder extends Seeder
{
    public function run()
    {
        $categories = [
            'Ficção',
            'Não-ficção',
            'Fantasia',
            'Ciência',
            'Biografia',
            'História',
            'Tecnologia',
            'Arte',
            'Culinária',
            'Viagem'
        ];

        foreach ($categories as $category) {
            Category::create(['name' => $category]);
        }
    }
}
```

4.4 Para popular a tabela de categorias, execute o comando para rodar o seeder.

```
php artisan db:seed --class=CategorySeeder
```

4.5 Verifique no banco de dados se as categorias foram inseridas corretamente (*select * from categories;*).

Passo 5: Criar Factories para Livros, Autores e Publishers

5.1 Na linha de comando, execute o comando para criar uma factory para a tabela de autores.

```
php artisan make:factory AuthorFactory --model=Author
```

5.2 Abra o arquivo gerado na pasta `database/factories/AuthorFactory.php`.

5.3 Edite o arquivo `AuthorFactory.php` para gerar dados fake de autores.

```
<?php

namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;
use App\Models\Author;

class AuthorFactory extends Factory
{
    protected $model = Author::class;

    public function definition()
    {
        return [
            'name' => $this->faker->name(),
            'birth_date' => $this->faker->date()
        ];
    }
}
```

5.2 Na linha de comando, execute o comando para criar uma factory para a tabela de publishers.

```
php artisan make:factory PublisherFactory --model=Publisher
```

5.3 Abra o arquivo gerado na pasta `database/factories/PublisherFactory.php`.

5.4 Edite o arquivo `PublisherFactory.php` para gerar dados fake de editoras.

```
<?php

namespace Database\Factories;

use App\Models\Publisher;
use Illuminate\Database\Eloquent\Factories\Factory;
```

```
class PublisherFactory extends Factory
{
    protected $model = Publisher::class;

    public function definition()
    {
        return [
            'name' => $this->faker->unique()->company, // Gera um nome
de empresa único
            'address' => $this->faker->address,
        ];
    }
}
```

5.5 Na linha de comando, execute o comando para criar uma factory para a tabela de categories.

```
php artisan make:factory CategoryFactory --model=Category
```

5.6 Abra o arquivo gerado na pasta `database/factories/CategoryFactory.php`.

5.7 Edite o arquivo `CategoryFactory.php` para gerar dados fake de editoras.

```
<?php

namespace Database\Factories;

use App\Models\Category;
use Illuminate\Database\Eloquent\Factories\Factory;

class CategoryFactory extends Factory
{
    protected $model = Category::class;

    public function definition()
    {
        return [
            'name' => $this->faker->unique()->word,
        ];
    }
}
```

5.8 Na linha de comando, execute o comando para criar uma factory para a tabela de livros.

```
php artisan make:factory BookFactory --model=Book
```

5.9 Abra o arquivo gerado na pasta `database/factories/BookFactory.php`.

5.10 Edite o arquivo `BookFactory.php` para gerar dados fake de livros.

```
<?php
namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;
use App\Models\Book;
use App\Models\Author;
use App\Models\Category;
use App\Models\Publisher;

class BookFactory extends Factory
{
    protected $model = Book::class;

    public function definition()
    {
        return [
            'title' => $this->faker->sentence(),
            'author_id' => Author::factory(),
            'category_id' => Category::factory(), // Agora usa a
CategoryFactory
            'publisher_id' => Publisher::factory(),
            'published_year' => $this->faker->year
        ];
    }
}
```

Passo 6: Criar Seeder para Popular o Banco com Autores, Publishers e Livros

6.1 Na linha de comando, execute o comando para criar um seeder para a tabela de autores e livros.

```
php artisan make:seeder AuthorPublisherBookSeeder
```

6.2 Abra o arquivo gerado na pasta `database/seeders/AuthorAndBookSeeder.php`.

6.3 Edite o arquivo `AuthorPublisherBookSeeder.php` para criar milhares de registros de autores e livros.

```php
<?php
namespace Database\Seeders;

use Illuminate\Database\Seeder;
use App\Models\Author;
use App\Models\Book;
use App\Models\Category;
use App\Models\Publisher;

class AuthorPublisherBookSeeder extends Seeder
{

    public function run()
    {
        // Gera 100 autores, cada um com 10 livros
        Author::factory(100)->create()->each(function ($author) {
            // Gera uma editora para cada autor
            $publisher = Publisher::factory()->create();

            // Cria 10 livros para cada autor, associando uma categoria existente
            $author->books()->createMany(
                Book::factory(10)->make([
                    'category_id' => Category::inRandomOrder()->first()->id,
                    'publisher_id' => $publisher->id,
                ])->toArray()
            );
        });
    }
}
```

Passo 7: Popular Todo o Banco de Dados

7.1 Caso deseje rodar todas as seeds de uma vez, você pode adicionar os seeders criados no método `DatabaseSeeder` localizado em :

```php
<?php
namespace Database\Seeders;

use Illuminate\Database\Seeder;
use Faker\Factory as FakerFactory;
class DatabaseSeeder extends Seeder
{

    public function run()
    {
        FakerFactory::create()->unique(true);
        $this->call([
            CategorySeeder::class,
            AuthorPublisherBookSeeder::class,
        ]);
    }
}
```

7.2 Execute o comando para rodar todas as seeds e popular o banco de dados completo.

```
php artisan migrate:refresh
php artisan db:seed
```

7.3 Verifique no banco de dados se os registros de autores, editoras e livros foram inseridos corretamente (acesse o mysql e verifique..).

Conclusão

Nesta atividade, você aprendeu a criar os modelos necessários, configurar relacionamentos, criar seeds para inserir dados predefinidos no banco de dados e usou factories para gerar uma grande quantidade de registros fake de maneira eficiente. Essas técnicas são essenciais para o desenvolvimento e testes de aplicações no Laravel, permitindo que você popule rapidamente seu banco de dados para desenvolvimento e testes.