Atividade Prática 4: Uso do Eloquent para Criação de registro, Consultas e Manipulação de Relacionamentos no Sistema de Biblioteca

---

Introdução:

Nesta atividade, você irá construir e manipular um sistema de gerenciamento de biblioteca utilizando o Laravel e seu ORM, o Eloquent. A atividade inclui a criação das tabelas no banco de dados, a implementação dos relacionamentos entre as entidades e a manipulação de dados por meio de consultas avançadas e funções de agregação.

Além disso, você aprenderá a utilizar o **Tinker**. O **Tinker** é uma ferramenta interativa do Laravel que permite executar comandos diretamente no ambiente da aplicação. Ele é útil para testar comandos e interagir com o banco de dados em tempo real, especialmente para práticas rápidas e experimentação de funcionalidades. Nesta prática, vamos usar o Tinker para manipular o banco de dados do sistema de biblioteca que inclui as entidades `User`, `Author`, `Book`, `Category`, `Publisher` e `Borrowing`.

Para abrir o Tinker, utilize o comando:

```
php artisan tinker
```

Objetivo:

- Empregar funções Eloquent e definir corretamente os relacionamentos entre as entidades.
- Realizar consultas avançadas
- Comparar a performance das técnicas de Eager Loading Lazy Loading
- Verificar e validar as operações de manipulação de dados utilizando o Tinker.

Passo 1: Consultas Básicas com Eloquent

1.1 Listar Todos os Usuários

No Tinker, para listar todos os usuários, utilize o comando:

```
User::all();
```

Este comando retorna todos os registros da tabela `users`.

1.2 Trazer um Livro Específico pelo ID

Para consultar um livro específico pelo seu `id`, use:

```
use App\Models\Book;
Book::find(1); // Substitua 1 pelo ID desejado
```

Esse comando retorna o livro com o ID especificado, permitindo acessar seus dados e, se necessário, realizar manipulações ou obter informações relacionadas, como autor e editora.

Passo 2: Contar Quantos Livros Cada Publisher Possui

Usaremos o `withCount` no Eloquent para incluir a contagem de livros relacionados a cada `Publisher`. Isso ajuda a ver quantos livros estão associados a cada editora.

No Tinker, execute o comando:

```
use App\Models\Publisher;
Publisher::withCount('books')->take(20)->get();
```

Esse comando:

1. Filtra os livros publicados após o ano 2000.
2. Limita o resultado a 20 livros.

Passo 3: Usar Funções de Agregação para Analisar os Anos de Publicação dos Livros

Neste passo, vamos utilizar as funções de agregação do Eloquent para obter informações sobre os anos de publicação dos livros no sistema. Esses comandos serão executados no Tinker para calcular:

1. A **média** dos anos de publicação.
2. O **somatório** dos anos de publicação.
3. O ano de publicação do **livro mais antigo**.
4. O ano de publicação do **livro mais recente**.

3.1 Calcular a Média dos Anos de Publicação dos Livros

Para calcular a média dos anos de publicação, execute o comando abaixo no Tinker:

```
$mediaAno = Book::avg('published_year');
```

Esse comando calcula e exibe a média dos anos de publicação dos livros.

3.2 Calcular o Somatório dos Anos de Publicação dos Livros

Para obter a soma dos anos de publicação, execute o seguinte comando:

```
$somaAno = Book::sum('published_year');
```

Este comando calcula o total somado de todos os anos de publicação dos livros.

3.3 Encontrar o Livro Mais Antigo

Para descobrir o ano de publicação do livro mais antigo, execute:

```
$anoMaisAntigo = Book::min('published_year');
```

Esse comando retorna o menor valor de `published_year`, representando o livro mais antigo no sistema.

3.4 Encontrar o Livro Mais Recente

Para obter o ano de publicação do livro mais recente, utilize o comando:

```
$anoMaisRecente = Book::max('published_year');
```

Esse comando retorna o maior valor de `published_year`, representando o livro mais recente no sistema.

Essas consultas oferecem uma visão geral sobre a distribuição dos anos de publicação dos livros e auxiliam na análise de dados agregados no sistema de biblioteca.

Passo 4: Salvar e Imprimir a Contagem de Livros por Categoria

No Tinker, execute o comando para armazenar os dados em uma variável `$categorias`:

```
use App\Models\Category;
$categorias = Category::withCount('books')->get();
```

1. Em seguida, use um `foreach` para iterar sobre cada categoria e exibir o nome e a quantidade de livros:

```
foreach ($categorias as $categoria) {
    echo "{$categoria->name}: {$categoria->books_count} livros\n";
}
```

2. **Explicação**

- O primeiro comando armazena todas as categorias com suas contagens de livros em `$categorias`.
- O `foreach` percorre cada categoria em `$categorias`, exibindo o nome da categoria e a contagem de livros associados a ela.

Esse método permite ver rapidamente a quantidade de livros em cada categoria.

Passo 5: Entender o Problema do N+1, Eager Loading e Lazy Loading

O Problema do N+1

O problema do **N+1** ocorre em consultas de banco de dados quando o código executa uma consulta inicial para buscar registros (como uma lista de livros) e, em seguida, faz uma nova consulta para cada item individual (como buscar o autor de cada livro). Esse padrão resulta em múltiplas consultas ao banco, uma para o conjunto inicial e uma para cada registro relacionado, o que pode ser extremamente ineficiente para grandes conjuntos de dados.

Lazy Loading

Lazy Loading é o comportamento padrão no Eloquent. Com Lazy Loading, os relacionamentos são carregados apenas quando acessados explicitamente. Isso pode levar ao problema do N+1, pois cada vez que se acessa o autor de um livro em uma lista, uma nova consulta é feita.

Exemplo de Lazy Loading: Trazer todos os livros e exibir o nome do autor de cada um.

```
$books = Book::all();

foreach ($books as $book) {
    echo "{$book->title} - {$book->author->name}\n";
}
```

Neste caso, para cada livro, uma consulta é feita para buscar o autor, o que resulta no problema do N+1, pois teremos 1 consulta para os livros e N consultas para os autores.

Eager Loading

Eager Loading é uma técnica para carregar todos os dados relacionados em uma única consulta, minimizando o número de consultas ao banco e resolvendo o problema do N+1. No Eloquent, o Eager Loading é implementado com o método `with()`, que permite pré-carregar relacionamentos.

Exemplo de Eager Loading: Trazer todos os livros e exibir o nome do autor de cada um.

```
$books = Book::with('author')->get();

foreach ($books as $book) {
    echo "{$book->title} - {$book->author->name}\n";
}
```

Com o Eager Loading, o Eloquent executa uma única consulta para obter os livros e uma consulta adicional para carregar todos os autores. Isso reduz o número de consultas a 2, independentemente do número de livros, eliminando o problema do N+1.

Código para Comparação de Tempo: Lazy Loading vs Eager Loading

Execute o código a seguir no Tinker para realizar as operações e medir o tempo:

```php
// Medindo o tempo para Lazy Loading
use App\Models\Book;
use App\Models\Author;
$inicioLazy = microtime(true);

$booksLazy = Book::all();
foreach ($booksLazy as $book) {
    echo "{$book->title} - {$book->author->name}\n";
}

$tempoTotalLazy = microtime(true) - $inicioLazy;
echo "Tempo total para Lazy Loading: {$tempoTotalLazy} segundos\n";
```

```php
// Medindo o tempo para Eager Loading
use App\Models\Book;
use App\Models\Author;
$inicioEager = microtime(true);

$booksEager = Book::with('author')->get();
foreach ($booksEager as $book) {
    echo "{$book->title} - {$book->author->name}\n";
}

$tempoTotalEager = microtime(true) - $inicioEager;
echo "Tempo total para Eager Loading: {$tempoTotalEager} segundos\n";
```

Explicação

1. **Lazy Loading**: Capturamos o tempo antes e depois de buscar todos os livros e iterar para acessar o nome do autor de cada um. O resultado é armazenado em `$tempoTotalLazy`.

2. **Eager Loading**: Repetimos o processo, mas com Eager Loading (`with('author')`). O tempo total é armazenado em `$tempoTotalEager`.

Interpretação

Após a execução, o Tinker exibirá o tempo total para cada abordagem, permitindo comparar o desempenho e ver a diferença no tempo de execução entre **Lazy Loading** e **Eager Loading**. Essa comparação é útil para verificar como o Eager Loading pode reduzir o tempo de execução ao evitar o problema do N+1.

Passo 6: Criar um Novo Livro com Todos os Seus Relacionamentos

Instruções

Criar o Autor (Author): Primeiro, vamos criar o autor do livro. Substitua os dados pelo que desejar.

```
use App\Models\Author;
$author = Author::create([
    'name' => 'John Doe',
    'birth_date' => '1975-08-15'
]);
```

Criar a Categoria (Category): Em seguida, criamos a categoria do livro:

```
$category = Category::create([
    'name' => 'Fiction'
]);
```

Criar a Editora (Publisher): Agora, criamos a editora do livro:

```
$publisher = Publisher::create([
    'name' => 'Best Publisher',
    'address' => '123 Publisher Lane'
]);
```

Criar o Livro (Book): Finalmente, vamos criar o livro e associá-lo aos relacionamentos criados. Certifique-se de preencher as informações do livro conforme necessário.

```
$book = Book::create([
    'title' => 'The Great Adventure',
    'author_id' => $author->id,
    'category_id' => $category->id,
    'publisher_id' => $publisher->id,
    'published_year' => 2024
]);
```

Verificar o Livro e Seus Relacionamentos: Para garantir que tudo foi criado corretamente, execute:

```
echo "Livro: {$book->title}\n";
```

```
echo "Autor: {$book->author->name}\n";
```

```
echo "Categoria: {$book->category->name}\n";
```

```
echo "Editora: {$book->publisher->name}\n";
```

Esse processo cria um novo livro e define suas relações com `Author`, `Category`, e `Publisher`, integrando o livro ao sistema de biblioteca com todas as informações associadas.