09

Consulta com filtros

Transcrição

[00:00] Na última aula vimos como fazer a listagem. Ao invés de usar aquela lista estática com tópicos em memória, nós começamos a acessar o banco de dados de verdade, utilizando o padrão `Repository`, que o código fica bem simples. Só criamos uma interface vazia que herda do `JpaRepository` e utilizamos ela no nosso `Controller`.

[00:24] Na aula de hoje vamos fazer uma alteração nesse método, porque do jeito que está no momento, estou trazendo todos os registros. Às vezes queremos filtrar. Imagine que quero filtrar os tópicos pelo status, nome do curso, nome do autor ou por qualquer outra informação.

[00:42] No nosso exemplo, imagine que eu quero filtrar os tópicos pelo nome do curso. Entrando no browser, temos nosso endereço que é http://localhost:8080/topicos (http://localhost:8080/topicos), mas imagine que quero passar um parâmetro de Query, de URL, seguindo aquele formato "?^nomeCurso=" e passar o nome do curso, por exemplo "Spring". Imagine que tem um curso chamado "Spring" e eu quero filtrar pelo nome do curso, então ficaria: http://localhost:8080/topicos?^nomeCurso=Spring (http://localhost:8080/topicos?%5EnomeCurso=Spring). Até funciona, só que ele não filtrou, ele trouxe tudo, porque mandamos um parâmetro que não estamos recebendo no `Controller`. Como faço para receber um parâmetro dessa maneira?

[01:35] Lá no nosso `TopicosController.java`, no método `lista()`, é só receber o parâmetro no método mesmo. Então, `String nomeCurso`. Vamos ver se vai

funcionar. Vou salvar e colocar um `System.out.printLn(nomeCurso)`, só para ver se está chegando o parâmetro.

```
package br.com.alura.forum.controller;

import java.util.List;

@RestController
public class TopicosController {

    @Autowired
        private TopicoRepository topicoRepository;

        @RequestMapping("/topicos")
        public List<TopicoDto> lista(String nomeCurso) {
        System.out.println(nomeCurso);
        List<Topico> topicos = topicoRepository.findAll();
        return TopicoDto.converter(topicos);
    }
}
```

COPIAR CÓDIGO

Vou acessar de novo a URL http://localhost:8080/topicos?^nomeCurso=Spring (http://localhost:8080/topicos?%5EnomeCurso=Spring), pressionar "F5" e apareceu "Spring" no Console.

[02:12] Dessa maneira consigo passar um parâmetro na URL, como Query parâmetro, e chega certinho no `Controller`. Vamos apagar o parâmetro "?nomeCurso=Spring" e ver se do jeito anterior, http://localhost:8080/topicos (http://localhost:8080/topicos), está tudo ok. Funciona e ele manda "*null*", nulo, quando não recebe o parâmetro.

[02:35] Agora que já vi que funciona, vou apagar o `System.out.println(nomeCurso)` e só precisamos alterar o `findAll()`. A ideia é: se estiver vindo o parâmetro `nomeCurso` não vou chamar o `findAll`. Preci

chamar um `findAll()` filtrando pelo nome do curso. E se não estiver vindo um parâmetro aí sim faço um `findAll()` . Dá para fazer um código bem simples, um `if (nomeCurso == null)` , significa que ele não está filtrando. Eu vou chamar o `findAll` , por exemplo e aí eu retorno à lista `return TopicoDto.converter(topicos)` . Seguindo, na próxima linha, escrevo `else` e, se não estiver nulo - se está vindo parâmetro - eu preciso filtrar.

```
package  br.com.alura.forum.controller;

import java.util.List;

@RestController
public class TopicosController {

    @Autowired
        private TopicoRepository topicoRepository;

        @RequestMapping("/topicos")
        public List<TopicoDto> lista(String nomeCurso) {
         if (nomeCurso == null) {
                List<Topico> topicos =
topicoRepository.findAll();
                return TopicoDto.converter(topicos);
         } else {
             List<Topico> topicos =
topicoRepository.findAll();
                 return TopicoDto.converter(topicos);
         }
     }
}
```

COPIAR CÓDIGO

[03:18] O Spring data, naquela interface `Repository` também tem algumas facilidades nesse caso, se você quiser filtrar por algum parâmetro. A ideia é: não vamos mais utilizar o `findAll()` . Vamos utilizar outro método. Aí, o

Spring Data tem um padrão de nomenclatura. Se você seguir esse padrão, ele consegue gerar a Query para você automaticamente. O padrão de nomenclatura seria `findBy()` e o nome do atributo na entidade (do atributo que você quer filtrar). Por exemplo, imagine que quero filtrar pelo `Titulo`, o nome do método seria `findByTitulo(nomeCurso)` e eu passaria o parâmetro que representa esse atributo.

```
            List<Topico> topicos =
topicoRepository.findByTitulo(nomeCurso);
                return TopicoDto.converter(topicos);
```

COPIAR CÓDIGO

[04:04] Dá erro de compilação porque esse método `findByTitulo()`, por ser específico do nosso projeto, não tem naquela interface que herdamos. O `findAll()` vem automaticamente, mas o `findByTitulo()` não existe. Então vou selecionar ele, apertar o comando "Ctrl + 1" e selecionar "Create method 'findByTitulo(String)' in type 'TopicoRepository" e ele vai criar um método no nosso `Repository`. Se você criar um método com essa nomenclatura, o Spring consegue gerar Query automaticamente, não é necessário digitar a Query com JPQL.

```
package br.com.alura.forum.repository;

import java.util.List;

import
org.springframework.data.jpa.repository.JpaRepository;

import br.com.alura.forum.modelo.Topico;

public interface TopicoRepository extends
JpaRepository<Topico, Long> {
```

```
    List<Topico> findByTitulo(String nomeCurso);

    }
```

COPIAR CÓDIGO

[04:50] Só que, no nosso caso, não quero filtrar pelo título, quero filtrar pelo nome do curso. Como eu colocaria? `FindByNomeCurso` ? Não vai funcionar. É até interessante porque, no nosso `Repository` , na hora de rodar o projeto, o Spring data já faz uma validação e se tiver alguma coisa errada, ele já joga *Exception*. Se olharmos aqui a *Exception*, encontraremos "No property nomeCurso found for type Topico!", isto é, na entidade `Topico` não tem uma propriedade chamada `nomeCurso` .

[05:32] Vamos abrir nossa entidade `Topico` e, realmente, o `título` existe, mas `nomeCurso` não. No caso, `Curso` é um relacionamento. Temos um tópico, ele está relacionado com `Curso` . Dentro de `Curso` é que tem o atributo `nome` . Como eu faço para filtrar, não por um atributo da entidade, mas por um atributo que é de um relacionamento que está na minha entidade?

Não tem problema, o Spring também consegue filtrar isso. Só precisamos mudar o padrão para `findbyCurso` sendo que `Curso` é entidade e eu não quero filtrar pelo curso, e eu concateno na sequência `Nome` . Isto, é `findByCursoNome` , onde `Curso` é a entidade de relacionamento e `Nome` é o atributo dentro dessa entidade de relacionamento. Dessa maneira, ele vai conseguir encontrar.

```
package br.com.alura.forum.repository;

import java.util.List;

import
org.springframework.data.jpa.repository.JpaRepository;

import br.com.alura.forum.modelo.Topico;
```

```
public interface TopicoRepository extends
JpaRepository<Topico, Long> {

    List<Topico> findByCursoNome(String nomeCurso);

    }
```

COPIAR CÓDIGO

[06:32] Vou rodar de novo o projeto e agora tudo certo. Vamos testar acessando a URL http://localhost:8080/topicos (http://localhost:8080/topicos). Se eu chamar assim, com "/topicos", sem passar nenhum parâmetro, entra no `if`, está nulo o parâmetro, e chamou o `findAll()`. Se eu passar o parâmetro "?^nomeCurso=Spring", ele me traz os colchetes vazios.

Vamos agora no console do H2 fazer o select, primeiro na tabela "TOPICO", "SELECT FROM TOPICO". Depois, vamos fazer o select na tabela de cursos, "SELECT FROM CURSO". Agora, retorno no navegador e escrevo o parâmetro "?nomeCurso=Spring+Boot", com sinal `+` unindo "Spring Boot", porque colocar espaço no navegador pode gerar problema na URL. Seguindo, o outro era "HTML5", nesse caso, com espaço entre "HTML" e "5", então http://localhost:8080/topicos?^nomeCurso=HTML%205 (http://localhost:8080/topicos?%5EnomeCurso=HTML%205) e veremos que ele carregou corretamente.

Dessa maneira, filtramos por parâmetros usando o Spring Data. Você cria um método seguindo o padrão de nomenclatura do Spring Data, `findBy` e o nome do atributo que você quer filtrar. Se for o atributo de um relacionamento, "Nome do atributo do relacionamento" e, concatenado, o "Nome do atributo", porque ele sabe que é para filtrar pelo relacionamento.

[08:42] Como é uma Query específica do nosso projeto, esse método não existe automaticamente, precisamos criar, mas é só declarar a assinatura do método no nosso `Repository` que o Spring Data gera essa consulta automaticamente para você.

[09:00] Só um detalhe importante. Vamos pensar na seguinte situação. Imagine que na entidade `Topico` tem um atributo chamado `cursoNome` e eu criei o método seguindo o padrão de nomenclatura `findByCursoNome(String nomeCurso)` .

```
@Entity
public class Topico {

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;
        private String cursoNome;
        private String mensagem;
        private LocalDateTime dataCriacao =
LocalDateTime.now();
        @Enumerated(EnumType.STRING)
        private StatusTopico status =
StatusTopico.NAO_RESPONDIDO;
        //...
        }
```

COPIAR CÓDIGO

Mas, agora vai dar um problema, vai gerar ambiguidade. Quando estou falando `CursoNome` , estou falando do atributo ou do relacionamento `Curso` e, lá dentro, do atributo nome? Nesse caso, como faço para diferenciar? Você pode colocar um "underline", `findByCurso_Nome(String nomeCurso)` .

```
package br.com.alura.forum.repository

import java.util.List;

import
org.springframework.data.jpa.repository.JpaRepository;

import br.com.alura.forum.modelo.Topico;
```

```
public interface TopicoRepository extends
JpaRepository<Topico, Long> {

    List<Topico> findByCurso_Nome(String nomeCurso);
    }
```

COPIAR CÓDIGO

Esse underline vai dizer para o Spring que `Curso` vai ser um relacionamento e lá dentro do relacionamento é que tem um atributo `Nome`. Agora ele sabe que não é o atributo `CursoNome`. É o relacionamento `Curso` e, dentro dele, `Nome`. Se você quisesse filtrar pelo atributo, é só não colocar o underline.

[10:45] Você pode navegar dentro dos relacionamentos quantos níveis for. Poderia ter `findbyCursoCategoriaNome(String nomeCurso)` (se categoria fosse uma classe). Ele sai navegando entre os relacionamentos. Dentro do tópico ele vai buscar o curso, dentro do curso buscar a categoria, e dentro da categoria buscar o nome. Funciona muito bem. Ele gera todos os JOINs e monta a Query certinho.

[11:07] Outro detalhe importante é que a vantagem de você usar essa abordagem é que você cria o método com esse nome e o Spring monta a Query para você. A desvantagem é que você tem que seguir o padrão de nomenclatura.

[11:25] Mas às vezes você não gostou do nome de método, não queria que o método tivesse esse nome em inglês. Por exemplo, queria chamar meu método de `carregarPorNomeDoCurso(String nomeCurso)`. Se você criar um método com essa nomenclatura, você não está mais seguindo o padrão, a convenção do Spring data e ele não vai conseguir gerar a Query automaticamente para você. Só que você consegue ensinar para ele quando determinado método for chamado, qual é a Query que você quer executar.

[12:14] Existe uma anotação que você coloca, o `@Query`. E aí você importa do pacote, `org.springframework.data.jpa.repository`. Nos parênteses, entre

aspas, você vai ter que montar a Query na mão, usando o JPQL. Então, ficaria `@Query("SELECT t FROM Topico t WHERE t.curso.nome = :nomeCurso")`

```
package br.com.alura.forum.repository;

import java.util.List;

import
org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;

import br.com.alura.forum.modelo.Topico;

public interface TopicoRepository extends
JpaRepository<Topico, Long> {

    List<Topico> findByCursoNome(String nomeCurso);

        @Query("SELECT t FROM Topico t WHERE t.curso.nome =
:nomeCurso")
        List<Topico> carregarPorNomeDoCurso(String
nomeCurso);

    }
```

COPIAR CÓDIGO

[12:54] A vantagem dessa segunda abordagem é que você coloca o nome do método que você quiser, em português, no estilo que você quiser, só que a desvantagem é que você vai ter que montar a Query manualmente. Você vai ter que colocar o `@Query` e gerar a Query manualmente com JPQL.

[13:10] Tem mais um problema. Como eu tenho um parâmetro `nomeCurso`, o Spring não assume que esse parâmetro é o que está no método. Você precisa colocar mais uma anotação, que é o `@Param`. No `@Param`, você coloca

`"nomeCurso"` (sem os dois pontos ":"). Aí fica a critério do que você definir com a sua equipe, o que vocês vão usar.

```
@Query("SELECT t FROM Topico t WHERE t.curso.nome =
:nomeCurso")
        List<Topico>
carregarPorNomeDoCurso(@Param("nomeCurso")(String
nomeCurso);
```

COPIAR CÓDIGO

[13:50] No nosso caso, vou usar o padrão do próprio Spring, porque é mais fácil, menos código para digitar. Com isso, finalizamos essa aula. Agora conseguimos fazer a busca dos tópicos filtrando pelo nome do curso. Espero vocês na próxima aula, onde vamos fazer a próxima funcionalidade, que é cadastrar um tópico no projeto. Por enquanto estamos focando só na listagem. No próximo vídeo vamos ver como fazer o cadastro, inserir novos tópicos no projeto.