02

Consultando entidades

Transcrição

Até então, havíamos estudado os métodos de persistência (salvar, atualizar e excluir) e o ciclo de vida. Nesta aula, focaremos na parte de consultas. Vamos continuar na classe `CadastroDeProduto.java` e nela extrairemos todo o código do método `main()` a seguir para um método separado.

```
public static void main(String[] args) {
        Categoria celulares = new Categoria("CELULARES");
        Produto celular = new Produto("Xiaomi Redmi", "Muito
legal", new BigDecimal("800"), celulares );

        EntityManager em = JPAUtil.getEntityManager();
        ProdutoDao produtoDao = new ProdutoDao(em);
        CategoriaDao categoriaDao = new CategoriaDao(em);

        em.getTransaction().begin();

        categoriaDao.cadastrar(celulares);
        produtoDao.cadastrar(celular);

        em.getTransaction().commit();
        em.close();
    }

}
```

COPIAR CÓDIGO

Para isso, apertaremos o botão direito e selecionaremos "Refactor > Extract Method". Na próxima tela, nomearemos o método como "cadastrarProduto". Assim, deixaremos isolada essa parte de cadastro da categoria e do produto. Ao ser chamado, o método `main()` chama o `CadastrarProduto()` e, em seguida, fazemos todo o código que já existia: criamos uma `Categoria`, um `Produto`, um `EntityManager` as DAOS, salvamos e fechamos tudo.

Já fechamos o `EntityManager`, está tudo no banco de dados e no estado detached, nada está gerenciado. Agora, queremos trabalhar diretamente com o banco de dados fazendo consultas, e a JPA possui alguns métodos para isso. Basicamente, podemos consultar uma entidade pelo id ou fazer *select*, uma *query*, uma consulta para carregar várias entidades, a depender do nosso objetivo.

Portanto, para fazer uma consulta por id, considerando `Long id = 1l;`, isto é, que queremos buscar uma entidade de id 1, nós precisaremos, primeiro, de um `EntityManager` e também criar uma classe DAO.

```
public static void main(String[] args) {
    cadastrarProduto();
        Long id = 1l;
        EntityManager em = JPAUtil.getEntityManager();
        ProdutoDao produtoDao = new ProdutoDao(em);
```

COPIAR CÓDIGO

Agora, podemos criar um método na classe DAO,o retorno deste método é um objeto do tipo `Produto`, ele será chamado de `buscarPorId()`, e receberá um parâmetro `Long id`.

```
public Produto buscarPorId(Long id) {

}
```

COPIAR CÓDIGO

Na JPA, para consultar por id, o `EntityManager` possui um método chamado `em.find(null, id)`, com o qual podemos chamar uma única entidade pelo id. O método `find()` recebe dois parâmetros, o primeiro é: quem é a entidade? Isto é, considerando as muitas tabelas que existem no banco de dados, diremos ao `EntityManager` qual delas desejamos buscar. Sendo assim, passamos a classe, `em.find(Produto.class, id)`, e, a partir da entidade `Produto`, será fornecido o mapeamento.

O segundo parâmetro é o id, ou seja, qual é chave primária, o identificador, da entidade que queremos buscar? Logo, passaremos o parâmetro `id`. Também incluiremos `return`, ou seja, `return em.find(Produto.class, id);`, para dizer que o `find` retorna o objeto do tipo da classe que estamos passando, que, no nosso caso, é `Produto`.

Agora, vamos ao `CadastroDeProduto.java`, nós instanciamos e não precisaremos mais da variável `Long id = 1l;`, podemos apagá-la. Queremos carregar `Produto p = produtoDao.buscarPorId(1l)`. Também podemos inserir um `System.out.println(p.getPreco());`.

```
Produto p = produtoDao.buscarPorId(1l);
System.out.println(p.getPreco());
```

COPIAR CÓDIGO

O produto que cadastramos foi o celular `"Xiaomi Redmi"`, e o preço dele é `"800"`, então, se buscarmos pelo id, precisa vir 800. Vamos rodar (apertando o botão direito e, em seguida, "Run As > 1 Java Application") e observar no Console o que acontecerá. Ele carregou e gerou um *select* com alguns *alias* nas colunas, fez um *join* onde era necessário e buscou pelo id corretamente.

Portanto, ele fez a consulta, devolveu o objeto e imprimiu 800. Essa é uma maneira de fazer uma busca pelo id na qual usamos um método `em.find()` c

`EntityManager` que serve para isso. Porém, pode acontecer de desejarmos carregar várias entidades, neste caso, o `find()` não poderá ajudar. Sendo assim, vamos criar mais um método na nossa classe `ProdutoDao.java` .

Agora, teremos um método `List<Produto>` do `java.util` e, em seguida, indicaremos `buscarTodos()` . Nós não queremos mais buscar pelo id e, sim, carregar todos os produtos que estão no banco de dados.

```
public List<Produto> buscarTodos() {

}
```

COPIAR CÓDIGO

A principal maneira de fazer uma busca é utilizar uma linguagem chamada JPQL (Java Persistence Query Language) que é parecida, mas não é SQL. Funciona como se fosse uma SQL orientada a objetos. Assim, com a JPQL, é possível montar a *query* do jeito que quisermos.

Para isso, faremos o método `return em.createQuery()` , passaremos uma String para ele (a JPQL nada mais é do que uma String) e criaremos outra separada: `String jpql = ""` . Nela, motaremos o *select* parecido com o SQL: `String jpql = "SELECT"` .

No SQL, teríamos: `"SELECT * FROM"` e o nome da tabela que é `produtos` , então, `"SELECT * FROM produtos"` . Diferentemente disso, no JPQL, não passaremos a tabela e, sim, o nome da entidade, isto é, `Produto` . Recordando o problema do JDBC referente ao acoplamento com o banco de dados, perceberemos que, tendo o nome da tabela e precisando alterá-la, teríamos que mudar na entidade e em todas as classes DAO do projeto.

Na JPQL no lugar do asterísco, nós podemos utilizar um *alias*: `"SELECT p FROM Produto AS p"` (sendo que o `AS` é opcional), assim dizemos para que o próprio objeto `p` seja carregado, a entidade com todos os atributos. No

`em.createQuery(jpql)`, passamos o `jpql` e, na sequência, um `.getResultList()`. O `em.createQuery(jpql)` não dispara a *query* no banco de dados, apenas monta a *query*, para disparar de fato, chamamos o `getResultList()`.

```
    public List<Produto> buscarTodos() {
        String jpql = "SELECT p FROM Produto p";
            return em.createQuery(jpql).getResultList();
    }

}
```

COPIAR CÓDIGO

Está dando *warning* por causa da tipagem. Ele não sabe que a `em.createQuery(jpql)` retorna um objeto - uma lista - de `Produto`. Mas, para acabar com esse erro, podemos fazer o `createQuery()` em outra versão, na qual passamos o `jpql` e o tipo da classe - da entidade - que será devolvida nessa *query*. Basta passar, `jpql, Produto.class`.

```
    public List<Produto> buscarTodos() {
        String jpql = "SELECT p FROM Produto p";
            return em.createQuery(jpql,
Produto.class).getResultList();
    }

}
```

COPIAR CÓDIGO

Agora, ele infere que o `getResultList()` devolverá um `List` de `Produto` e para de dar *warning*. Retornando ao `CadastroDeProduto.java`, vamos chamar o método `produtoDao.buscarTodos()`, selecionar o comando "Ctrl + 1", pedir para que `buscarTodos` seja jogado em uma variável local, chamar a variável de

`todos`, fazer um `forEach()` e, dado o produto, faremos um `System.out.println()` no `p.getNome();`.

```
Produto p = produtoDao.buscarPorId(1l);
System.out.println(p.getPreco());

List<Produto> todos = produtoDao.buscarTodos();
todos.forEach(p2 -> System.out.println(p.getNome()))
```

COPIAR CÓDIGO

Vamos rodar ("Run As > 1 Java Application") e, analisando o Console, perceberemos que ele fez o *select* do id, imprimiu "800", na sequência fez o nosso *select* e imprimiu o nome "Xiaomi Redmi". Perceberemos também que ele converte o JPQL em um SQL. A consulta que fizemos foi simplificada, mas é possível utilizar *join* e simplificar ainda mais, com SQL puro seria mais complicado.

Então, nesta aula aprendemos a fazer consultas com a JPA. Existe a busca pelo id, `buscarPorId()`, onde usamos `em.find()` e indicamos qual é a entidade e qual é o id.

E existe a busca por JPQL, com a linguagem `SELECT p FROM Produto p"`, onde usamos `em.createQuery()` e indicamos onde está a *quey*, qual a classe da entidade que essa *query* vai devolver e utilizamos `getResultList()` para ele de fato carregar essa *query*, convertê-la para um SQL e disparar carregando as entidades e montando os objetos (sem a necessidade de fazer isso manualmente) com *ResultSet*, conforme era no JDBC.

Espero que tenham gostado. Na próxima aula continuaremos discutindo um pouco mais sobre consultas. Vejo vocês lá!!