02

Selecionando as vendas do banco de dados

Transcrição

Atualmente, estamos simulando as vendas, e não pegando-as do banco de dados, ou seja, a classe `Venda` não é uma entidade. E um dos nossos objetivos para finalizar a aplicação é justamente colocar as vendas no banco de dados, e não mais produzi-las em memória.

Então vamos lá, a primeira coisa para definir `Venda` como uma entidade é anotar a classe com `@Entity` :

```
@Entity
public class Venda {
```

Uma entidade deve definir uma chave primária, então vamos criar um atributo `id` para ser essa chave. Vamos anotá-lo com `@Id` , justamente para dizer que esse atributo será a chave primária no banco, e com `@GeneretedValue` , para o banco gerenciar o seu valor, assim não precisamos atribui-lo.

Além disso, nessa classe temos um relacionamento, várias vendas são associadas a um livro, então vamos anotar o atributo `livro` com `@ManyToOne` .

Precisamos também criar um construtor padrão para a classe, já que é uma exigência do CDI:

```
@Entity
public class Venda {

    @Id
    @GeneratedValue
    private Integer id;

    @ManyToOne
    private Livro livro;
    private Integer quantidade;

    public Venda() {
    }

    // restante do código
}
```

Pronto, para finalizar precisamos adicionar a nova entidade no arquivo `META-INF/persistence.xml` . Adicione juntamente com as outras entidades:

```
<class>br.com.caelum.livraria.modelo.Venda</class>
```

Podemos agora reiniciar o Tomcat, e fazer login na aplicação, só para inicializar o JPA e a nova tabela ser criada.

Vamos agora entrar no MySQL (`mysql -u root`) e visualizar a tabela:

```
mysql> use livrariadb;
mysql> show tables;
+----------------------+
| Tables_in_livrariadb |
+----------------------+
| Autor                |
| Livro                |
| Livro_Autor          |
| Usuario              |
| Venda                |
| hibernate_sequence   |
+----------------------+
6 rows in set (0.00 sec)

mysql> desc Venda;
+------------+---------+------+-----+---------+----------------+
| Field      | Type    | Null | Key | Default | Extra          |
+------------+---------+------+-----+---------+----------------+
| id         | int(11) | NO   | PRI | NULL    | auto_increment |
| quantidade | int(11) | YES  |     | NULL    |                |
| livro_id   | int(11) | YES  | MUL | NULL    |                |
+------------+---------+------+-----+---------+----------------+
3 rows in set (0.00 sec)
```

Ótimo, a tabela já está lá, mas ela ainda está vazia, vamos então inserir algumas vendas. **ATENÇÃO:** Repare que para inserir uma venda, precisamos do `id` do livro, então os `insert` s só serão válidos com um `id` de um livro válido. Então antes dê um `select` nos livros e veja seus ids. Exemplo:

```
mysql> select * from Livro;
+----+----------------+-------------------+-------+------------------+
| id | dataLancamento | isbn              | preco | titulo           |
+----+----------------+-------------------+-------+------------------+
|  1 | 2016-02-26     | 112-3-54-986321-5 |  49.9 | MEAN             |
|  2 | 2016-02-27     | 156-0-16-963266-3 |  49.9 | Arquitetura Java |
|  3 | 2016-03-11     | 123-1-45-632526-6 |  39.9 | AngularJS        |
|  4 | 2016-03-15     | 151-9-61-911961-9 |  39.9 | TDD              |
|  5 | 2016-03-15     | 136-5-44-515951-5 |  29.9 | Primefaces       |
+----+----------------+-------------------+-------+------------------+
5 rows in set (0.00 sec)

mysql> insert into Venda (quantidade, livro_id) values (126, 1);
Query OK, 1 row affected (0.00 sec)

mysql> insert into Venda (quantidade, livro_id) values (257, 2);
Query OK, 1 row affected (0.00 sec)

mysql> insert into Venda (quantidade, livro_id) values (58, 3);
Query OK, 1 row affected (0.00 sec)

mysql> insert into Venda (quantidade, livro_id) values (199, 4);
Query OK, 1 row affected (0.00 sec)

mysql> insert into Venda (quantidade, livro_id) values (313, 5);
Query OK, 1 row affected (0.00 sec)
```

Você pode inserir uma venda para cada livro cadastrado no seu banco. Com o modelo preparado, precisamos preparar o `VendasBean` para pegar as vendas do banco e montar o gráfico, pois ainda estamos criando as vendas em memória.

Selecionando as vendas do banco de dados

O método que gera as vendas aleatoriamente é o método `getVendas`. Vamos modificá-lo para ele fazer um `select` no banco de dados, para isso vamos injetar o `EntityManager` e criar a query que seleciona todas as vendas do banco:

```
@Named
@ViewScoped
public class VendasBean implements Serializable {

    @Inject
    private EntityManager manager;

    // getVendasModel comentado

    public List<Venda> getVendas(long seed) {

        List<Venda> vendas = this.manager.createQuery("select v from Venda v",
                Venda.class).getResultList();

        return vendas;
    }
}
```

Com o método pronto, não precisamos mais do `seed`, pois ele era utilizado pelo `Random`, então ele não será mais passado por parâmetro para o método `getVendas`, vamos removê-lo.

Não precisamos mais injetar o `LivroDao` também, podemos removê-lo.

No método `getVendasModel()`, nós simulávamos duas séries de vendas. Vamos deixar apenas uma, pois não temos tantas informações no banco de dados para distinguir uma série de outra. Vamos remover a série de 2015:

```
public BarChartModel getVendasModel() {

    BarChartModel model = new BarChartModel();
    model.setAnimate(true);

    ChartSeries vendaSerie = new ChartSeries();
    vendaSerie.setLabel("Vendas 2016");

    List<Venda> vendas = getVendas();
    for (Venda venda : vendas) {
        vendaSerie.set(venda.getLivro().getTitulo(), venda.getQuantidade());
    }

    model.addSeries(vendaSerie);

    return model;
}
```

Podemos visualizar a página de vendas e ver o novo gráfico das vendas, mas dessa vez com as vendas cadastradas no banco de dados!