03. Performance de consultas

Date	@02/01/2023
Categoria	Java e persistência
Curso	Java e JPA: consultas avançadas performance e modelos complexos

Tópicos

- Projeto da aula anterior
- Entendendo Lazy e Eager
- Consultas com Join Fetch
- Consultas com Lazy e Eager
- Vantagens do join fetch
- Faça como eu fiz
- O que aprendemos?

Consultas com Lazy e Eager

Considere as seguintes entidades JPA:

```
@Entity
@Table(name = “clientes”)
public class Cliente {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    @OneToOne(fetch = FetchType.EAGER)
    private Endereco endereco;
    @OneToMany(fetch = FetchType.LAZY, mappedBy = “cliente”)
    private List<Telefone> telefones = new ArrayList<>();

}
```

```
@Entity
@Table(name = “enderecos”)
public class Endereco {

    @Id
    @GeneratedValue(strategy = GenerationType.ID
    private Long id;
    private String logradouro;
    private String cep;
    private String cidade;
    private String uf;
    private String bairro;
    private String numero;

}
```

```
@Entity
@Table(name = “telefones”)
public class Telefone {
```

```
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String ddd;
    private String numero;
    @ManyToOne
    private Cliente cliente;

}
```

E o seguinte trecho de código:

```
Cliente cliente = em.find(Cliente.class, 1l);
System.out.println(cliente.getNome());
```

Que tipo de consulta no banco de dados o trecho de código anterior vai gerar?

- Um `select` na tabela de *clientes* fazendo `join` apenas com a tabela de *enderecos*
- O relacionamento com a entidade `Endereco` é do tipo *eager*, portanto a consulta vai gerar um join com a tabela de *enderecos*

Vantagens do join fetch

Por qual motivo é interessante utilizar o recurso *join fetch* em uma consulta JPQL?

- Para evitar carregar relacionamentos em todas as consultas da aplicação
- O join fetch permite escolher quais relacionamentos serão carregados em determinada consulta, ao invés de sempre os carregar

---

O que aprendemos?

Nessa aula, você aprendeu:

- Como funcionam as estratégias *EAGER* e *LAZY*, em consultas de entidades que possuem relacionamentos;
- Por que a JPA pode lançar a exception `LazyInitializationException` em determinadas situações
    - Caso o EntityManager que e responsavel por fazer a ligação com o banco de dados estiver fechado e você precise acessar uma tabela que não foi carregada antes do seu fechamento, essa exception vai ser lançada
    - Para resolver esse problema utilizamos o *Join Fetch*
- As boas práticas no carrgamento de entidades com relacionamentos;
- Como realizar consultas planajedas com o recruso *join fetch*