Formação Desenvolvedor Moderno
Módulo: Back end

Capítulo: JPA, consultas SQL e JPQL

https://devsuperior.com.br

JPA e entidades

Sessão JPA, estados de entidades, objetos associados

Relembrando: estudos prévios necessários

Revisão Álgebra Relacional e SQL

Pra quê? Para relembrar as operações básicas com SQL.

https://www.youtube.com/watch?v=GHpE5xOxXXI

Super revisão de OO e SQL com Java e JDBC

Pra quê? Para que você compreenda na prática como é consultar os dados de um banco de dados somente com Java e JDBC, sem utilizar uma ferramenta ORM (Mapeamento Objeto-Relacional).

https://www.youtube.com/watch?v=xC_yKw3MYX4

Nivelamento ORM - JPA e Hibernate

Pra quê? Para que você tenha uma introdução teórica e prática sobre ORM com JPA, **antes de ir direto para o Spring** com o Spring Data JPA.

https://www.youtube.com/watch?v=CAP1IPgeJkw

JPA e gerenciamento de entidades

- A JPA faz o **gerenciamento** das entidades do sistema durante a sessão JPA.
- Uma **sessão JPA** corresponde ao contexto em que a JPA está fazendo operações com entidades durante uma conexão com o banco de dados.
  - EntityManager: objeto da JPA que encapsula uma conexão com o banco de dados e que gerencia as entidades durante uma sessão JPA.

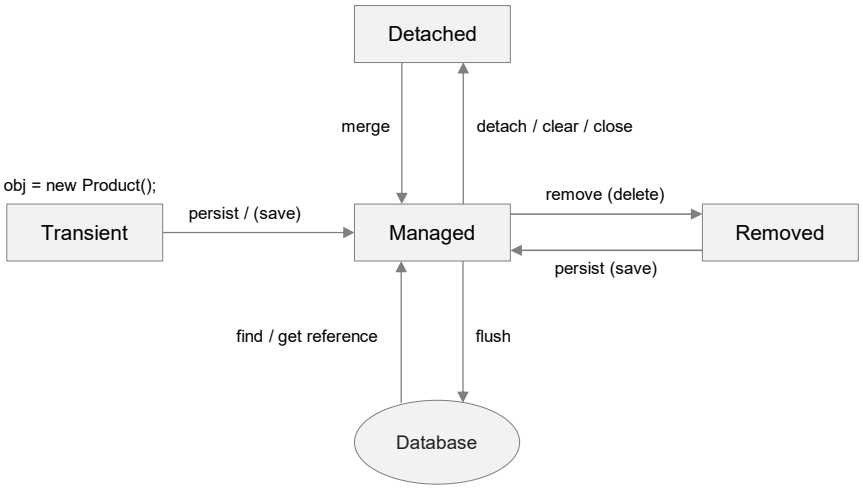
JPA "raiz":

```
EntityManagerFactory emf = ...
EntityManager em = emf.createEntityManager();

Product prod = new ...

em.getTransaction().begin();
em.persist(prod);
em.getTransaction().commit();
```

Spring Data JPA:

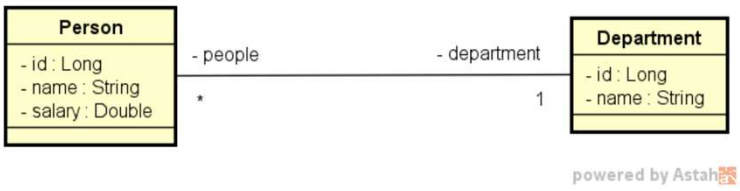
```
@Autowired
private ProductRepository repository;

@Transactional
public void meuMetodo() {
    Product prod = new ...
    repository.save(prod);
```

Estados de uma entidade



Salvar entidades associadas para-um

https://github.com/devsuperior/aula-salvar-para-um




```
POST http://localhost:8080/people

{
    "name": "Nova Pessoa",
    "salary": 8000.0,
    "department": {
        "id": 1
    }
}
```

```
POST http://localhost:8080/people

{
    "name": "Nova Pessoa",
    "salary": 8000.0,
    "departmentId": 1
}
```

Salvar entidades associadas para-muitos


```
POST http://localhost:8080/products

{
    "name": "Produto novo",
    "price": 1000.0,
    "categories": [
        {
            "id": 2
        },
        {
            "id": 3
        }
    ]
}
```

https://github.com/devsuperior/aula-salvar-para-muitos

Evitando degradação de performance

Carregamento lazy, tratativas, Transactional

Grande vilão da JPA

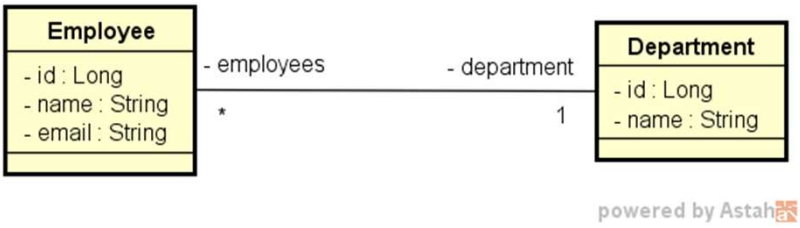
Grande vilão da JPA: idas e vindas desnecessárias ao banco de dados.

Causa comum: entidades associadas **lazy** carregando sob demanda.

Projeto exemplo

Projeto:
https://github.com/devsuperior/aula-lazy

Collection do Postman:
https://www.getpostman.com/collections/d5efb333d1d308d52b7c



Carregamento lazy de entidades associadas

Carregamento padrão de entidades associadas:

- Para-um: EAGER
- Para-muitos: LAZY

Devido ao comportamento lazy, enquanto a sessão JPA estiver ativa, o acesso a um objeto associado pode provocar uma nova consulta ao banco de dados.

Formas de alterar o comportamento padrão

- Atributo fetch no relacionamento da entidade (NÃO RECOMENDADO)
- Cláusula JOIN FETCH
- Consultas customizadas (IDEAL)

Atributo fetch do relacionamento

Cuidado: a mudança desse atributo pode gerar efeitos indesejados! Evite esta abordagem!

Exemplo:

```
@ManyToOne(fetch = FetchType.LAZY)
```

Cláusula JOIN FETCH

Nota: esta cláusula não funciona para buscas paginadas do Spring.

Exemplo:

```
public interface EmployeeRepository extends JpaRepository<Employee, Long> {

        @Query("SELECT obj FROM Employee obj JOIN FETCH obj.department")
        List<Employee> findEmployeesWithDepartments();
}
```

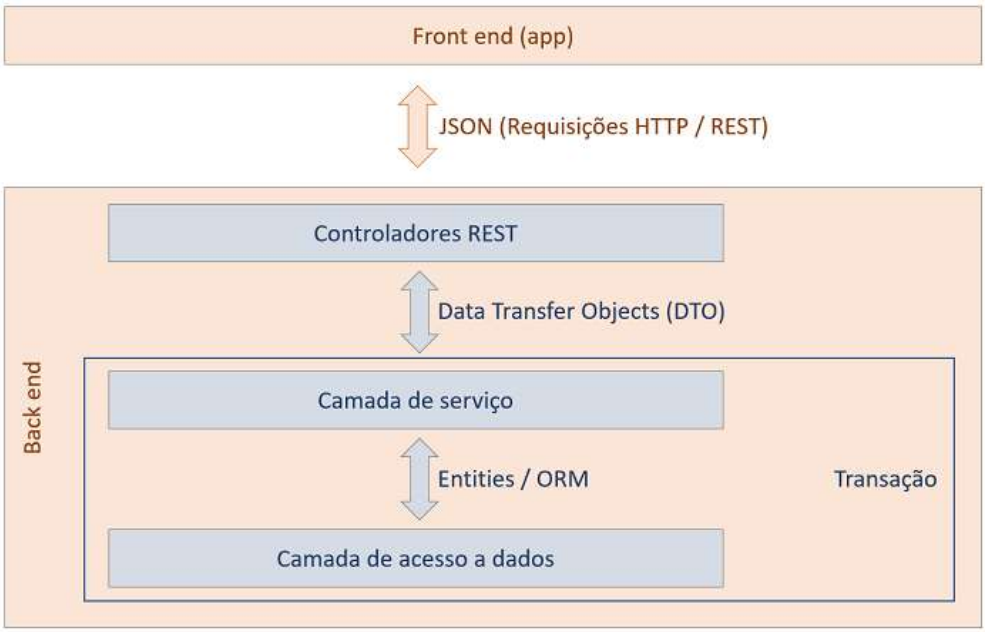

Cache em memória das entidades

- A JPA mantém um "cache" das entidades gerenciadas na mesma sessão JPA (mapa de identidade).
- Se você trouxer entidades para a memória, a JPA não volta ao banco se você precisar novamente delas durante a mesma seção JPA.

Transactional e open-in-view no Spring

- A annotation @Transactional assegura:
  (1) resolução da transação com o banco de dados
  (2) resolução de todas pendências “lazy” com o banco de dados

- A propriedade spring.jpa.open-in-view=false faz com que a sessão JPA seja encerrada antes de voltar para a camada controller (camada web)



Importante: problema N+1 consultas

Aulão no Youtube:
https://www.youtube.com/watch?v=sqbqoR-lMf8

Consultas customizadas

Query methods, SQL, JPQL

Query methods

- No JpaRepository do Spring Data JPA, é possível fazer uma consulta customizada apenas pelo nome do método

https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repositories.query-methods

- Polêmica: vale a pena usar?
  - Para consultas muito simples: ok
  - Para consultas mais complexas: melhor escrever a consulta

JPQL - linguagem de consulta da JPA

Linguagem de consulta específica da JPA.

Geralmente toda ferramenta de ORM possui uma linguagem e/ou ferramentas próprias para realização de consultas ao banco de dados.

A JPQL é parecida com a SQL, porém adaptada para o modelo de acesso a dados JPA.

Exemplo 1

SQL:

```
SELECT *
FROM tb_employee
WHERE UPPER(name) LIKE 'MARIA%'
```

JPQL:

```
SELECT obj
FROM Employee obj
WHERE UPPER(obj.name) LIKE 'MARIA%'
```


Exemplo 2

SQL:

```
SELECT tb_employee.*
FROM tb_employee
INNER JOIN tb_department ON tb_department.id = tb_employee.department_id
WHERE tb_department.name = 'Financeiro'
```

JPQL:

```
SELECT obj
FROM Employee obj
WHERE obj.department.name = 'Financeiro'
```

Polêmica: vale a pena se especializar na JPQL?

- Vantagens da JPQL:
  - Algumas consultas podem ficar mais simples na JPQL
  - Usufrui melhor do Spring Data JPA (paginação, JpaRepository)
  - Os objetos resultantes são entidades gerenciadas pela JPA

- Desvantagens da JPQL:
  - Consultas complexas podem ficar difíceis de escrever e validar (mais fácil escrever e testar a consulta SQL diretamente na ferramenta de banco de dados)
  - Não tem união (JPA 2.x)
  - Curva de aprendizado para uma tecnologia específica

SQL Raiz

Revisão de Álgebra Relacional e SQL

https://www.youtube.com/watch?v=GHpE5xOxXXI

Referências

https://www.w3schools.com/sql/default.asp

Exercícios

https://beecrowd.com.br (antigo https://urionlinejudge.com.br)

Seleção de exercícios

Grupo 1: projeção, restrição
2602, 2603, 2604, 2607, 2608, 2615, 2624

Grupo 2: JOIN
2605, 2606, 2611, 2613, 2614, 2617, 2618, 2619, 2620, 2621, 2622, 2623, 2742

Grupo 3: GROUP BY, subconsultas
2609, 2993, 2994, 2995, 2996

Grupo 4: Expressões na projeção
2610, 2625, 2738, 2739, 2741, 2743, 2744, 2745, 2746, 3001

Grupo 5: Diferença, União
2616, 2737, 2740, 2990

Grupo 6: Difíceis
2988, 2989, 2991, 2992, 2997, 2998, 2999

Preparação do Postgresql

- Opção 1 - instalação direta (mais leve):
  - Instale o Postgresql (versão 12 ou 13 ou 14)
  - Instale o pgAdmin ou o DBeaver
- Opção 2 - Docker:
  https://github.com/devsuperior/fdm-ambiente/blob/main/DOCKER.md

Estudos de caso: SQL e JPQL

Estudos de caso iniciados:
https://drive.google.com/file/d/1aMrWScakLfLUIfRFLXWzqlrnYqQJtDG1/view?usp=sharing

- URI 2602 - busca simples
- URI 2611 - join simples
- URI 2621 - like e between
- URI 2609 - group by
- URI 2737 - união
- URI 2990 - diferença / left join

DISCLAIMER: aulas feitas no STS e não no Intellij

URI 2990 (diagrama)