Java Persistence API (JPA)
Mapeamento de Relacionamentos

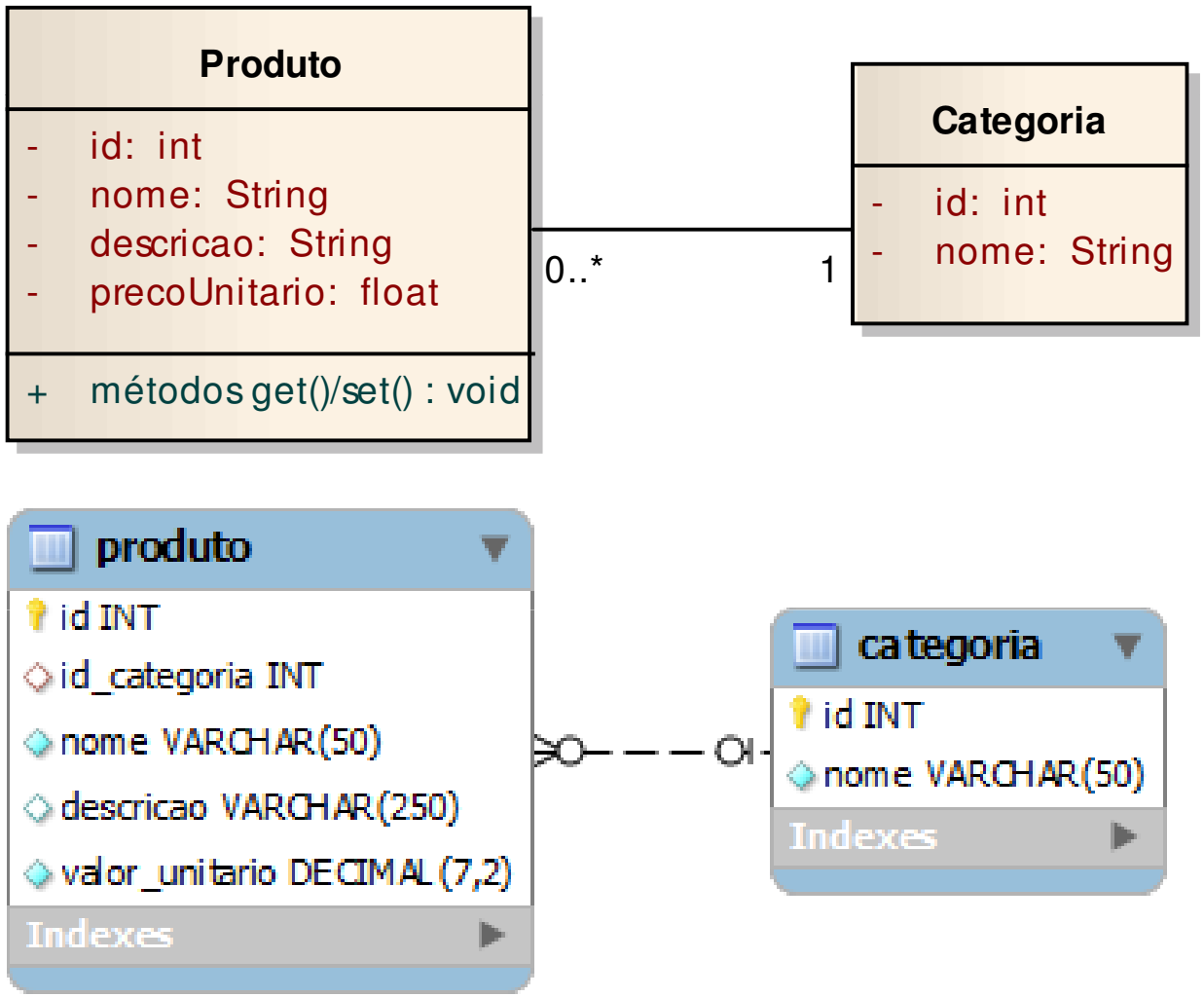
Fernando dos Santos
fernando.santos@udesc.br

UDESC
Ibirama

Mapeamento de Relacionamentos

Associação, Agregação, Composição

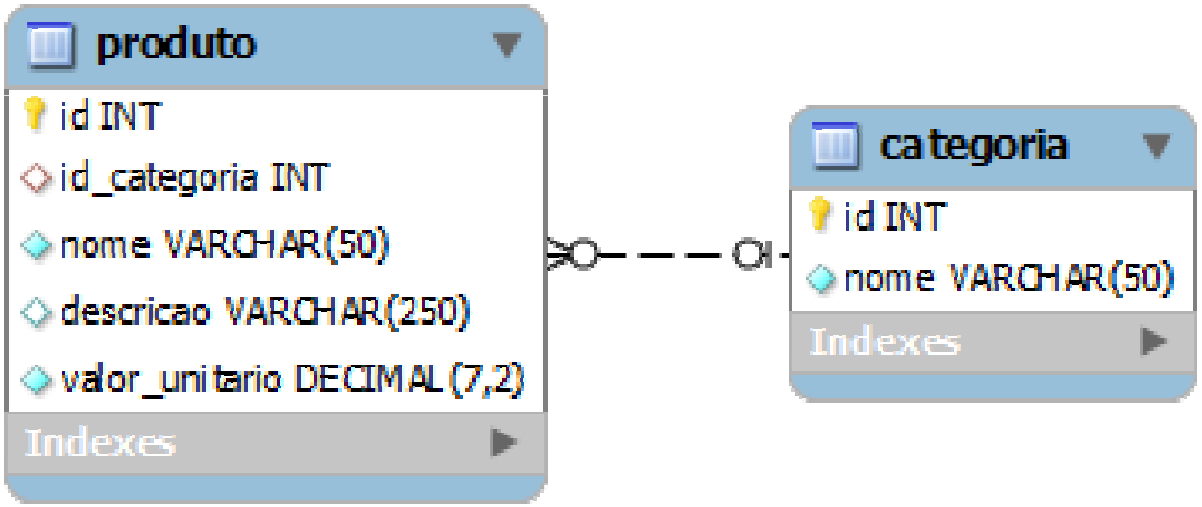
- Tipos de Relacionamentos mapeáveis:
  - Um para Um
  - Muitos para Um
  - **Um para Muitos**



Relacionamento **Um para Muitos**

Mapeamento

- Uma categoria está associada a muitos produtos.



- Categoria possui como atributo uma **coleção** (lista) de Produtos.

- Se houver um mapeamento **Muitos para Um** na entidade "muitos" (Produto), o mapeamento na entidade "um" (Categoria) apenas faz referência a este mapeamento Muitos para Um.

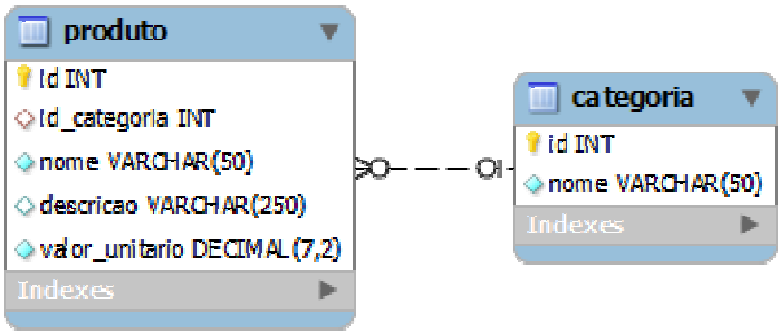
Relacionamento **Um para Muitos**

Mapeamento

```
@Entity
@Table(name="categoria")
public class Categoria implements Serializable{
    // outros atributos

    @OneToMany(mappedBy="categoriaProduto", cascade= CascadeType.ALL)
    private List<Produto> produtos = new ArrayList<Produto>();

    public List<Produto> getProdutos() {
        return produtos;
    }

    public void setProdutos(List<Produto> produtos) {
        this.produtos = produtos;
    }

    // demais métodos get() e set()
}
```

nome do atributo que está mapeado na classe “muitos” (Produto)

tipos de operações que serão propagadas para as entidades do lado “muitos”: ALL / MERGE / PERSIST / REFRESH / REMOVE



Relacionamento **Um para Muitos**

Mapeamento – Cascade de Operações (1)

- PERSIST
  - persist() da entidade “um” é propagado para as entidades “muitos”
    - ao persistir um, as outras também serão persistidas
    - exemplo: *ao fazer persist() de uma categoria, os produtos desta categoria (se forem novos) também serão persistidos.*
  - requer que as entidades “muitos” já estejam associadas com a “um”
    - os produtos já devem ter sido adicionados na lista de produtos da categoria.

- REMOVE
  - remove() da entidade “um” é propagado para as entidades “muitos”
    - ao remover um, as outras também serão removidas
    - exemplo: *ao fazer remove() de uma categoria, os produtos desta categoria (se existirem) também serão removidos.*

Relacionamento **Um para Muitos**

Mapeamento – Cascade de Operações (2)

- MERGE
  - merge() da entidade “um” é propagado para as entidades “muitos”
    - ao fazer merge em um, as outras também sofrerão merge
    - exemplo: *ao fazer merge() de uma categoria, os produtos desta categoria (novos ou já existentes) também sofrerão merge.*
  - requer que as entidades “muitos” já estejam associadas com a “um”
    - os produtos devem estar na lista de produtos da categoria
- REFRESH
  - idem ao merge(), porém, com a operação de refresh()

- ALL
  - propaga às entidades “muitos”, qualquer operação feita na entidade “um”
    - PERSIST, REMOVE, MERGE e REFRESH

Relacionamento **Um para Muitos**

Manipulações (1)

- Criar novo produto **(1)** e nova categoria **(2)**;
- Associar categoria no produto **(3)**, e adicionar produto na categoria **(4).**

```
Produto prod = new Produto();  // (1)
prod.setNome("Bicicleta");
prod.setDescricao("Bicicleta 18 marchas");
prod.setPrecoUnitario(350);

Categoria cat = new Categoria();  // (2)
cat.setNome("Esporte Lazer");

prod.setCategoriaProduto(cat);   // (3)
cat.getProdutos().add(prod);      // (4)

em.getTransaction().begin();
em.persist(cat);    // (5)
em.getTransaction().commit();
```

- **Basta persistir categoria (Cascade ALL propaga para o produto) (5)**

Relacionamento **Um para Muitos**

Manipulações (2)

- Criar novo produto **(1)**; Associar em categoria já existente **(2) (3)**;

```
Produto prod = new Produto();  // (1)
prod.setNome("Mochila");
prod.setDescricao("Mochila 10 litros");
prod.setPrecoUnitario(95);

// busca a categoria 2(Malas) do banco
Categoria cat = em.find(Categoria.class, 2);

prod.setCategoriaProduto(cat);   // (2)
cat.getProdutos().add(prod);     // (3)

em.getTransaction().begin();
em.merge(cat);    // (4)
em.getTransaction().commit();
```

- **Basta merge na categoria (Cascade.ALL propaga para produto) (4)**

Relacionamento **Um para Muitos**

Manipulações (3)

- Trocar a categoria de um produto.

```
// No banco de dados, o produto 1 (Bicicleta)
// possui a categoria 1 (Esporte Lazer)
Produto prod = em.find(Produto.class, 1);

 // Armazenar categoria atual em uma variável para depois trocar
Categoria catAtual = prod.getCategoriaProduto();
// Buscar uma outra categoria (3-Brinquedos)
Categoria outraCat = em.find(Categoria.class, 3);

em.getTransaction().begin();
catAtual.getProdutos().remove(prod); // (1)
outraCat.getProdutos().add(prod);      // (2)
prod.setCategoriaProduto(outraCat);  // (3)
em.getTransaction().commit();
```

- É necessário:
  - **remover o produto da categoria antiga (1)**
  - **adicioná-lo na categoria nova (2)**
  - **setar a nova categoria no produto (3)**

Relacionamento **Um para Muitos**

Manipulações (4)

- Buscar uma categoria **(1)**

```
Categoria cat = em.find(Categoria.class, 3);    // (1)

System.out.println("Categoria: "+cat.getNome());

// Imprimindo os produtos desta categoria
System.out.println(“Produtos: ");
for(Produto prod : cat.getProdutos()){          // (2)
    System.out.println("Cód: "+prod.getId()+", Descrição"+prod.getDescricao());
}
```

- **Os produtos são recuperados junto com a categoria (2)**
  - será impresso:

```
Categoria: Brinquedos
Produtos
Cód: 1, Descrição: Bicicleta 18 marchas
```

Relacionamento **Um para Muitos**

Manipulações (5)

- Buscar uma categoria que possui produtos **(1)**
- Remover a categoria **(2)**

```
Categoria cat = em.find(Categoria.class, 1);  // (1)

em.getTransaction().begin();
em.remove(cat);   // (2)

em.getTransaction().commit(); // (3)
```

- **Se Cascade = ALL ou REMOVE, os produtos serão removidos (3)**

Relacionamento **Um para Muitos**

Consultas (1)

- Buscar todos os produtos de uma categoria (informada por parâmetro)
- Situação no banco de dados:

produto

ID	Descrição	ID Categoria
1	Bicicleta 18 marchas	1
2	Mochila 10 litros	2
3	Esteira de caminhada	1

categoria

ID	Nome
1	Esporte Lazer
2	Malas
3	Brinquedos

- Consulta:

```
Query cons = em.createQuery("select c from Categoria c where c.id = :idCat");
cons.setParameter("idCat", 1);

Categoria cat = (Categoria)cons.getSingleResult();
for (Produto prod : cat.getProdutos()) { // (1)
    System.out.println("Cód: " + prod.getId() + " Descr: " + prod.getDescricao());
}
```

- **Os produtos são recuperados automaticamente (*join* pelo mapeamento) (1)**
- Resultado:

ID	Descrição	ID Categoria
1	Bicicleta 18 marchas	1
3	Esteira de caminhada	1

Relacionamento **Um para Muitos**

Consultas (2)

- Buscar quantidade de produtos por categoria: **join pela coleção**
- Situação no banco de dados:

produto

ID	Descrição	ID Categoria
1	Bicicleta 18 marchas	1
2	Mochila 10 litros	2
3	Esteira de caminhada	1

categoria

ID	Nome
1	Esporte Lazer
2	Malas
3	Brinquedos

- Consulta:

```
Query cons = em.createQuery("select c,
                                    count(prods)
                                from Categoria c join c.produtos prods
                            group by c");
List<Object[]> resultados = cons.getResultList();
for (Object[] result : resultados) {
    Categoria cat = (Categoria)result[0];   // posicao [0] é a categoria
    Long qtd = (Long)result[1];             // posicao [1] é a quantidade
    System.out.println("Categoria: "+cat.getNome()+ ", Qtd: "+qtd);
}
```

o join é feito pela lista de produtos na entidade categoria

- Resultado:

```
Categoria: Esporte Lazer, Qtd: 2
Categoria: Malas,         Qtd: 1
```

E as categorias que não possuem produtos?

Relacionamento **Um para Muitos**

Consultas (3)

- Buscar quantidade de produtos por categoria (**TODAS**)
- Situação no banco de dados:

produto

ID	Descrição	ID Categoria
1	Bicicleta 18 marchas	1
2	Mochila 10 litros	2
3	Esteira de caminhada	1

categoria

ID	Nome
1	Esporte Lazer
2	Malas
3	Brinquedos

- Consulta:

```
Query cons = em.createQuery("select c,
                                    count(prods)
                                from Categoria c left join c.produtos prods
                            group by c");
List<Object[]> resultados = cons.getResultList();
for (Object[] result : resultados) {
    Categoria cat = (Categoria)result[0];   // posicao [0] é a categoria
    Long qtd = (Long)result[1];             // posicao [1] é a quantidade
    System.out.println("Categoria: "+cat.getNome()+ ", Qtd: "+qtd);
}
```

basta fazer um left join

- Resultado:

```
Categoria: Esporte Lazer, Qtd: 2
Categoria: Malas,         Qtd: 1
Categoria: Brinquedos,    Qtd: 0
```

Relacionamento **Um para Muitos**

Consultas (4)

- Buscar categorias que não possuem produtos (subconsulta)
- Situação no banco de dados:

produto

ID	Descrição	ID Categoria
1	Bicicleta 18 marchas	1
2	Mochila 10 litros	2
3	Esteira de caminhada	1

categoria

ID	Nome
1	Esporte Lazer
2	Malas
3	Brinquedos

- Consulta:

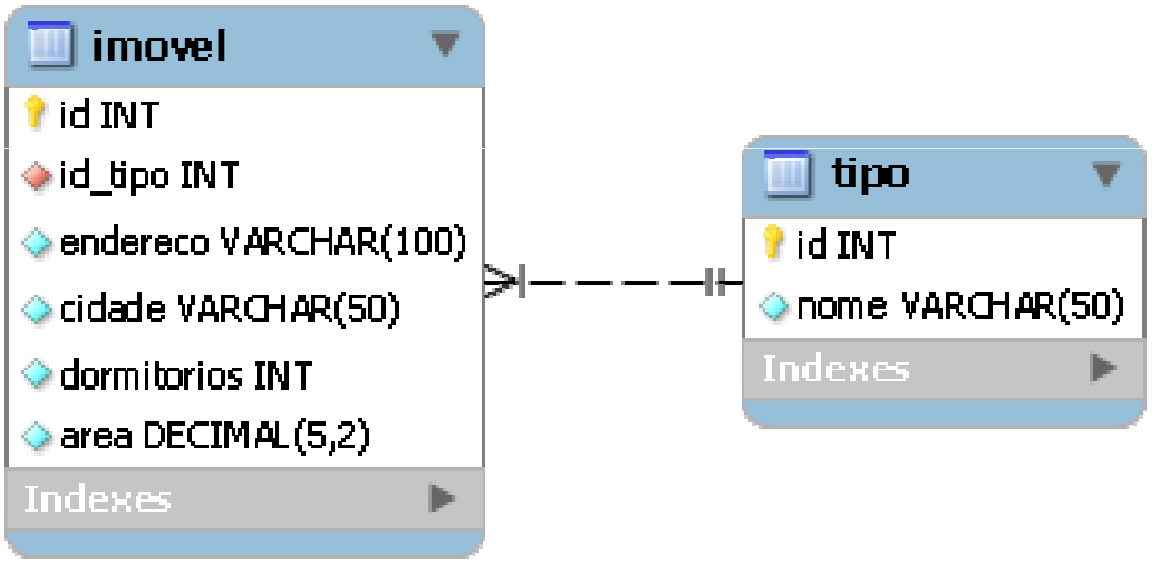
```
Query cons = em.createQuery("select c from Categoria c where c.produtos is empty");

List<Categoria> categorias = cons.getResultList();
for (Categoria cat : categorias) {
    System.out.println("ID: "+cat.getId()+", Nome: "+cat.getNome());
}
```

- Resultado:

ID	Nome
3	Brinquedos

Exercício

- Criar um relacionamento One-to-Many entre tipo e imóvel



- Altere seu sistema para refletir a inclusão deste relacionamento;
- Adicione novas opções de consultas, que devem ser implementadas utilizando o relacionamento One-to-Many:
  - todos os imóveis de uma determinado tipo (informado por parâmetro)
  - tipos que não possuem imóveis
  - todos os tipos, e para cada tipo, mostrar todos os imóveis do tipo

Bibliografia

- BAUER, Christian; KING, Gavin. **Java Persistence com Hibernate.** Rio de Janeiro: Ciência Moderna, 2007. 844 p.

- BURKE, Bill; MONSON-HAEFEL, Richard. **Enterprise JavaBeans 3.0**. 5.ed. São Paulo: Prentice Hall, 2007. 538 p.

- **The Java EE 6 Tutorial**, parte VI (Persistence)
  - http://download.oracle.com/javaee/6/tutorial/doc/