

# JPA: Mapeamento de Relações

sexta-feira, 30 de julho de 2021 10:29

## Classe Embutível ou Incorporável

As classes embutíveis não são classes persistentes independentes, são classes comuns da orientação objeto porém só existe para ser embutida em classes de persistências.

Normalmente é usada para definir o estado de algum atributo de uma classe de persistência, dessa maneira não possui um ID próprio

*Exemplo:*

A classe A\_VO e B\_VO são entidades que possuem o nome de suas tabelas como a e b respectivamente.

```
@Entity  
@Table(name = "a")  
public class A_VO implements Serializable {  
    @Id  
    @GeneratedValue(strategy = GenerationType.SEQUENCE)  
    private int codigo;  
    @Column(length = 40, nullable = false)  
    private String nome;  
    @Embedded  
    private C_VO c_vo;  
}
```

```
@Entity  
@Table(name = "b")  
public class B_VO implements Serializable {  
    @Id  
    @GeneratedValue(strategy = GenerationType.SEQUENCE)  
    private int codigo;  
    @Column(length = 40, nullable = false)  
    private String nome;  
    @Embedded  
    private C_VO c_vo;  
}
```

Observe a anotação @Embedded nas duas entidades, isso significa que C\_VO será a classe embutível.

```
@Embeddable  
public class C_VO implements Serializable {  
  
    @Column(length = 40, nullable = false)  
    private String valor1;  
  
    @Column(nullable = false)  
    private int valor2;  
  
    @Column(nullable = false)  
    private int valor3;  
}
```

Ao embutirmos a classe C\_VO é adicionado os atributos valor1, valor2 e valor3 nas classes A\_VO e

B\_VO.

Observe como a JPA irá criar a tabela de A\_VO e B\_VO

```
CREATE TABLE a
(
    codigo integer NOT NULL,
    nome character varying(40) NOT NULL,
    valor1 character varying(40) NOT NULL,
    valor2 integer NOT NULL,
    valor3 integer NOT NULL,
    CONSTRAINT a_pkey PRIMARY KEY (codigo)
)

CREATE TABLE b
(
    codigo integer NOT NULL,
    nome character varying(40) NOT NULL,
    valor1 character varying(40) NOT NULL,
    valor2 integer NOT NULL,
    valor3 integer NOT NULL,
    CONSTRAINT b_pkey PRIMARY KEY (codigo)
)
```

#### CASCATA DE ATRIBUTOS EMBUTIDOS

Observe agora a classe A\_VO, ela está embutindo a classe B\_VO que também embutiu a classe C\_VO, esse tipo de situação é conhecido como cascada de atributos embutidos.

```
@Entity
@Table(name = "a")
public class A_VO implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private int codigo;
    @Column(length = 40, nullable = false)
    private String nome;
    @Embedded
    private B_VO b_vo;
}
```

```
@Embeddable
public class B_VO implements Serializable {

    @Column(length = 40, nullable = false)
    private String valor0;

    @Embedded
    private C_VO c_vo;
}
```

```

@Embeddable
public class C_VO implements Serializable {

    @Column(length = 40, nullable = false)
    private String valor1;
    @Column(nullable = false)
    private int valor2;
    @Column(nullable = false)
    private int valor3;
}

```

A criação do banco da tabela a realizada pela JPA dessa vez vai possuir todos atributos de b\_vo e c\_vo.

```

CREATE TABLE a
(
    codigo integer NOT NULL,
    nome character varying(40) NOT NULL,
    valor0 character varying(40) NOT NULL,
    valor1 character varying(40) NOT NULL,
    valor2 integer NOT NULL,
    valor3 integer NOT NULL,
    CONSTRAINT a_pkey PRIMARY KEY (codigo)
)

```

Exemplo:

```

@Entity
@Table(name = "pessoa")
public class PessoaVO implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private int codigo;
    @Column(length = 40, nullable = false)
    private String nome;
    @Embedded
    private TelefoneVO fone;
}

```

```

@Embeddable
public class TelefoneVO implements Serializable {

    @Column(length = 3, nullable = false)
    private String fone_ddi;
    @Column(length = 3, nullable = false)
    private String fone_ddd;
    @Column(length = 9, nullable = false)
    private String fone_numero;
}

```

```

CREATE TABLE pessoa
(
    codigo integer NOT NULL,
    nome character varying(40) NOT NULL,
    fone_ddd character varying(3) NOT NULL,
    fone_ddi character varying(3) NOT NULL,
    fone_numero character varying(9) NOT NULL,
    CONSTRAINT pessoa_pkey PRIMARY KEY (codigo)
)

```

Agora imagina-se a seguinte situação, se for necessário inserir dois telefones para a mesma entidade, isso não é possível utilizando apenas um atributo embutido, desse modo é necessário criar dois atributos Embeddable.

```

@Entity
@Table(name = "pessoa")
public class PessoaVO implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private int codigo;
    @Column(length = 40, nullable = false)
    private String nome;
    @Embedded
    private TelefoneVO foneC; ←
    @Embedded
    private TelefoneVO foneR; ←
}

```

```

@Embeddable
public class TelefoneVO implements Serializable {

    @Column(length = 3, nullable = false)
    private String fone_ddi;
    @Column(length = 3, nullable = false)
    private String fone_ddd;
    @Column(length = 9, nullable = false)
    private String fone_numero;
}

```

Mas ainda sim existe um problema, observe a construção da tabela pessoa

```

CREATE TABLE pessoa
(
    codigo integer NOT NULL,
    nome character varying(40) NOT NULL,
    fone_ddd character varying(3) NOT NULL,
    fone_ddi character varying(3) NOT NULL,
    fone_numero character varying(9) NOT NULL,
    CONSTRAINT pessoa_pkey PRIMARY KEY (codigo)
)

```

Não existe diferença entre os atributos da classe telefoneVO, dessa maneira só é possível adicionar/embutir um telefone. Para solucionar este problema é realizadoa sobre escrita de atributos

#### **SOBRE ESCRITA DE ATRIBUTOS**

Para solucionar o problema apresentado anteriormente é utilizado outro tipo de anotação além da embedded, a @AttributeOverrides. Esse tipo de anotação permite sobreescrever os atributos.

Dentro a anotação @AttributeOverrides definimos através de name qual sera o o atributo do da classe Embeddable sera utilizado e depois qual nome da coluna que sera dada.

```

@Table(name = "pessoa")
public class PessoaVO implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private int codigo;
    @Column(length = 40, nullable = false)
    private String nome;

    @AttributeOverrides({
        @AttributeOverride(name = "fone_ddi", column = @Column(name = "fonec_ddi")),
        @AttributeOverride(name = "fone_ddd", column = @Column(name = "fonec_ddd")),
        @AttributeOverride(name = "fone_numero", column = @Column(name = "fonec_numero"))
    })
    @Embedded
    private TelefoneVO foneC;

    @AttributeOverrides({
        @AttributeOverride(name = "fone_ddi", column = @Column(name = "foner_ddi")),
        @AttributeOverride(name = "fone_ddd", column = @Column(name = "foner_ddd")),
        @AttributeOverride(name = "fone_numero", column = @Column(name = "foner_numero"))
    })
    @Embedded
    private TelefoneVO foneR;

```

Sobrescrita de nome de atributos

9

Dessa maneira sera criado colunas com nomes diferentes para cada atributo da classe embutida

```

CREATE TABLE pessoa
(
    codigo integer NOT NULL,
    nome character varying(40) NOT NULL,
    fonec_ddd character varying(255),
    fonec_ddi character varying(255),
    fonec_numero character varying(255),
    foner_ddd character varying(255),
    foner_ddi character varying(255),
    foner_numero character varying(255),
    CONSTRAINT pessoa_pkey PRIMARY KEY (codigo)
)

```

## ATRIBUTOS ENUMERADOS

Podemos criar classes de enum e atribuir ela como um tipo de atributo, o JPA utiliza a anotação @Enumerated(EnumType.ORDINAL) para enumerações, sendo "ORDINAL" o tipo de dado do enum que será armazenado no banco de dados, nesse caso números.

```

@Embeddable
public class TelefoneVO implements Serializable {

    @Enumerated(EnumType.ORDINAL)
    private TipoFoneEnum tipoFone;

    @Column(length = 3, nullable = false)
    private String fone_ddi;

    @Column(length = 3, nullable = false)
    private String fone_ddd;

    @Column(length = 9, nullable = false)
    private String fone_numero;
}

```

```

public enum TipoFoneEnum {
    TRABALHO,
    RESIDENCIAL,
    CELULAR
}

```

Nesse caso, TRABALHO, RESIDENCIAL, CELULAR serão armazenados no banco de dados no forma de numeros, 0,1,2.

Como já visto, para adicionar 2 telefones temos de criar duas sobreposições de atributos, mas se quisermos vários telefones vinculados a uma única entidade não seria conveniente fazer varios atributos com sobreposição, para resolver esse problema podemos utilizar coleção de tipos.

#### COLEÇÃO DE TIPOS BASE OU EMBUTIDOS

Ainda para os exemplos de telefone e pessoa, caso queiramos vincular varios telefones a uma pessoa precisamos de criar um @ElementCollection, que é uma coleção de telefones. Essa anotação será realizada em um set<TelefoneVO> (poderia trocar set por array ou arraylist), conjunto responsável por armazenar os telefones. Além dessa anotação, é usado o @CollectionTable, esta anotação irá criar uma nova tabela com os atributos do @Embeddable em questão (TelefoneVO) e dentro dela especificamos o nome da nova tabela, chave estrangeira da pessoa (em joinColumns) e, como essa nova tabela não possui chave (JPA 2.0) é necessário criar uma unicidade para que não se repita o telefone, isto é feita utilizando a o uniqueConstraint. Nesta situação utilizamos como unicidade o tipofone e pessoa\_fone (chave estrangeira da tabela pessoa, que referencia o código da pessoa).

```

@Entity
@Table(name = "pessoa")
public class PessoaVO implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private int codigo;

    @Column(length = 40, nullable = false)
    private String nome;

    @ElementCollection
    @CollectionTable(
        name="telefone",
        joinColumns=@JoinColumn(name="pessoa_fone"),
        uniqueConstraints= @UniqueConstraint(columnNames={"pessoa_fone", "tipofone"})
    )
    private Set<TelefoneVO> listaFone;
}

```

JPA 2.0 Não define Chave Primária para CollectionTable

A criação do banco de dados criado será:

```

CREATE TABLE pessoa
(
    codigo integer NOT NULL,
    nome character varying(40) NOT NULL,
    CONSTRAINT pessoa_pkey PRIMARY KEY (codigo)
)

CREATE TABLE telefone
(
    fone_ddd character varying(3) NOT NULL,
    fone_ddi character varying(3) NOT NULL,
    fone_numero character varying(9) NOT NULL,
    tipofone integer,
    pessoa_fone integer,
    CONSTRAINT fk_telefone_pessoa_fone FOREIGN KEY (pessoa_fone)
        REFERENCES pessoa (codigo) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE NO ACTION,
    CONSTRAINT unq_telefone_0 UNIQUE (pessoa_fone, tipofone)
)

```

*Exemplo de aplicação:*

- Realizando a inserção de telefones em uma lista de telefones e vinculando a uma pessoa

```
EntityManager entityManager = FabricaEntityManager.getEntityManager();

Set<TelefoneVO> listaFone = new HashSet();
listaFone.add(new TelefoneVO(TipoFoneEnum.TRABALHO, "+55", "065", "7070-7070"));
listaFone.add(new TelefoneVO(TipoFoneEnum.CELULAR, "+55", "065", "8040-4080"));

PessoaVO pessoaVO = new PessoaVO();
pessoaVO.setNome("Fulano");
pessoaVO.setListaFone(listaFone);

entityManager.getTransaction().begin();
entityManager.persist(pessoaVO);
entityManager.getTransaction().commit();
```

- Exibindo o(os) telefones vinculados a uma pessoa

```
EntityManager entityManager = FabricaEntityManager.getEntityManager();
entityManager.getTransaction().begin();
```

```
Query query = entityManager.createQuery("SELECT p FROM PessoaVO p ");
List<PessoaVO> listaPessoa = query.getResultList();
```

```
for (PessoaVO p : listaPessoa) {
    System.out.println("Nome: " + p.getNome());
    for(TeloneVO foneVO : p.getListaFone()){
        System.out.println("Fone: "+foneVO.getFone_ddi()
                           +"-"+foneVO.getFone_ddd()
                           +"-"+foneVO.getFone_numero());
    }
}
entityManager.getTransaction().commit();
```

#### CHAVE PRIMÁRIA COMPOSTA

Mesmo que atualmente o uso de chave compostas seja uma infração das normativas de bancos de dados, é possível construir uma chave composta utilizando classes embutíveis. Para isto, basta criar uma classe embutível e utilizar em uma entidade na chave primária.

```
@Entity  
@Table(name = "a")  
public class A_VO implements Serializable {  
  
    @EmbeddedId  
    private APK_VO id;  
  
    @Column(length = 40, nullable = false)  
    private String nome;  
}
```

```
@Embeddable  
public class APK_VO implements Serializable {  
  
    private int id1;  
    private float id2;  
}
```

```
CREATE TABLE a  
(  
    nome character varying(40) NOT NULL,  
    id2 double precision NOT NULL,  
    id1 integer NOT NULL,  
    CONSTRAINT a_pkey PRIMARY KEY (id2, id1)  
)
```

*Exemplo de aplicação*

- Inserir dados em chave composta

```
try {  
    EntityManager entityManager = FabricaEntityManager.getEntityManager();  
  
    A_VO a = new A_VO();  
    a.setId(new APK_VO(10,20));  
    a.setNome("a");  
  
    entityManager.getTransaction().begin();  
    entityManager.persist(a);  
    entityManager.getTransaction().commit();  
  
} catch (PersistenciaException ex) {  
    System.out.println(ex.toString());  
}  
-
```

- Recuperação de dados

```

try {
    EntityManager entityManager = FabricaEntityManager.getEntityManager();

    entityManager.getTransaction().begin();

    Query query = entityManager.createQuery("SELECT a FROM A_VO a WHERE a.id.id1 = 10");
    List<A_VO> listaAVO = query.getResultList();

    for(A_VO a : listaAVO){
        System.out.println("ID1: "+a.getId0.getId1());
        System.out.println("ID2: "+a.getId0.getId2());
        System.out.println("nome: "+a.getNome());
    }

    entityManager.getTransaction().commit();

} catch (PersistenciaException ex) {
    System.out.println(ex.toString());
}

```

Observe em WHERE a.id.id1=10, esse trecho refere-se ao APK\_VO id, logo para acessar o id1 que está contido na classe embutida deve percorrer a.id.id1.

#### PROPRIEDADE DE CASCATA

A anotação cascata é um tipo de anotação que permite que se replique em forma de cascata as ações realizadas na entidade. Deve-se utilizar essa propriedade com consciência das ações devido possuir replicação das suas ações definidas. Em projetos, uma má documentação pode interferir no controle da aplicação, pois a manutenção da aplicação pode ser feita por outro programador que não tenha consciência plena do uso de cascata.

As ações que são permitidos é:

- All: replica qualquer ação

```

@Entity
@Table(name = "a")
public class A_VO implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private int codigo;
    @Column(length = 40, nullable = false)
    private String nome;

```

```

    @OneToOne(cascade= CascadeType.ALL)
    private B_VO b_VO;

```

- Persist: replica a persistência
- Merge: replica as alterações
- Remove: replica a remoção
- Refresh: replica a atualização

```

try {
    EntityManager entityManager = FabricaEntityManager.getEntityManager();
    entityManager.getTransaction().begin();

    Query query = entityManager.createQuery("SELECT a FROM A_VO a WHERE a.nome = 'a1'");
    A_VO a = (A_VO) query.getSingleResult();

    entityManager.refresh(a); ←

    entityManager.getTransaction().commit();
} catch (PersistenciaException ex) {
    System.out.println(ex.toString());
}

```

## Cascade = REFRESH

- Detach: replica a desvinculação da entidade do contexto de persistencia

*Exemplo:*

Inclusão utilizando all

```

EntityManager entityManager = FabricaEntityManager.getEntityManager();
entityManager.getTransaction().begin();

```

```

B_VO b = new B_VO();
b.setNome("b1");
A_VO a = new A_VO();
a.setNome("a1");
a.setB_VO(b);

```

```
entityManager.persist(a);
```

```
entityManager.getTransaction().commit();
```

	codigo [PK] integer	nome character vari	b_vo_codigo integer
<b>1</b>	3551	a1	3552
*			

	codigo [PK] integer	nome character vari
<b>1</b>	3552	b1
*		

Observe que nas situações anteriores onde tínhamos uma classe encapsulada tínhamos de instanciar o objeto e inserir o seu valor e retornar(get) o resultado na entidade, agora nesta situação temos de apenas setar o resultado e inserir não o retorno (get) do objeto e sim todo o objeto, isso facilita na programação. Ao persistir apenas o objeto 'a' o objeto 'b' também é persistido devido ao efeito cascata all.

Removendo utilizando o all

```

try {
    EntityManager entityManager = FabricaEntityManager.getEntityManager();
    entityManager.getTransaction().begin();

    Query query = entityManager.createQuery("SELECT a FROM A_VO a WHERE a.nome = 'a1'");
    A_VO a = (A_VO) query.getSingleResult();

    entityManager.remove(a);

    entityManager.getTransaction().commit();
} catch (PersistenciaException ex) {
    System.out.println(ex.toString());
}

```

	<b>codigo</b> [PK] integer	<b>nome</b> character var	<b>b_vo_codigo</b> integer
*			

	<b>codigo</b> [PK] integer	<b>nome</b> character var
*		

Podemos configurar o cascade para realizar varias ações específicas, não apenas uma única

```

@Entity
@Table(name = "a")
public class A_VO implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private int codigo;
    @Column(length = 40, nullable = false)
    private String nome;

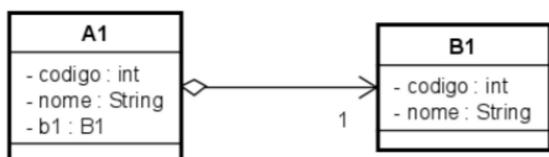
    @OneToOne(cascade = {CascadeType.MERGE, CascadeType.REFRESH})
    private B_VO b_VO;
}

```

## Configurando Cascade específico

### RELACIONAMENTO 1X1(ONE TO ONE)

O relacionamento entre entidades na JPA é o mapeamento entre relações. O relacionamento one to one, na prática acontece pouco. Quando encontramos uma relação one to one, temos de dizer a JPA que é esse tipo de relação, observe o exemplo abaixo.



Para que A1 tenha ocorrência de B1, temos de ter o atributo de b1 em A1

#### Anotação @OneToOne

Propriedades:

- Fetch: FetchType.EAGER recupera a instância de A1 e recupera a instância de B1 também.

FecthTyp.Lazy recupera a instancia de A1 mas não recupera a instancia de B1, apenas quando solicitado utilizando getB1()

#### *Implementação da classe B1*

```
package vo;

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "tabela_b1")
public class B1 implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private int codigo;

    @Column(length = 40, nullable = false)
    private String nome;

    public B1() {
    }

    public B1(String nome) {
        this.nome = nome;
    }

    public int getCodigo() {
```

#### *Implementação da classe A1*

```
package vo;
import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.Table;

@Entity
@Table(name = "tabela_a1")
public class A1 implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private int codigo;

    @Column(length = 40, nullable = false)
    private String nome;

    @OneToOne(fetch= FetchType.EAGER)
    private B1 b1;

    public A1() {
    }
```

Nessa caso da implementação da A1, ira recuperar a instancia de B1 vinculada ao atributo b1.

#### *Implementação main B1*

```

package apps;
import javax.persistence.EntityManager;
import util.EntityManagerUtil;
import vo.B1;
public class AppB1i {
    public static void main(String args[]) {
        try {
            EntityManager em = EntityManagerUtil.getEntityManager();
            em.getTransaction().begin();

            B1 b = new B1("b1");
            em.persist(b);

            b = new B1("b2");
            em.persist(b);

            b = new B1("b3");
            em.persist(b);

            em.getTransaction().commit();
        } catch (Exception ex) {
            System.out.println(ex.toString());
        }
    }
}

```

Data Output			Explain	Messages	Notifications
	codigo [PK] integer	nome character varying (40)			
1	951	b1			
2	952	b2			
3	953	b3			

Agora que já instanciado 3 objetos de B1, podemos instanciar objetos de A1. Nesse caso, vamos buscar os dados de B1, e instanciar em um novo objeto o primeiro resultado através do singleResult(). Em seguida instanciar A1, setando no atributo b1 o objeto b, após isso realizar a persistencia.

```

package apps;
import javax.persistence.EntityManager;
import javax.persistence.Query;
import util.EntityManagerUtil;
import vo.A1;
import vo.B1;
public class AppA1i {
    public static void main(String args[]) {
        try {
            EntityManager entityManager = EntityManagerUtil.getEntityManager();
            entityManager.getTransaction().begin();

            Query query = entityManager.createQuery("SELECT b FROM B1 b WHERE b.nome = 'b1'");
            B1 b = (B1)query.getSingleResult();

            A1 a = new A1("a1");
            a.setB1(b); //A1 agrega uma instancia de B1

            entityManager.persist(a);

            query = entityManager.createQuery("SELECT b FROM B1 b WHERE b.nome = 'b2'");
            b = (B1)query.getSingleResult();

            a = new A1("a2");
            a.setB1(b); //A1 agrega uma instancia de B1

            entityManager.persist(a);

            entityManager.getTransaction().commit();
        } catch (Exception ex) {
            System.out.println(ex.toString());
        }
    }
}

```

No banco de dados será salvo a chave primaria da tabela B1 no campo b1\_codigo (criado pela JPA)

Data Output			
	codigo [PK] integer	nome character varying (40)	b1_codigo integer
1	1101	a1	951
2	1102	a2	952

Recuperando dados de select

```

package apps;

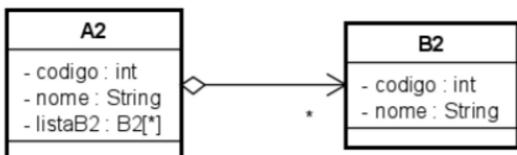
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.Query;
import util.EntityManagerUtil;
import vo.A1;

public class AppA1c {

    public static void main(String args[]) {
        try {
            EntityManager entityManager = EntityManagerUtil.getEntityManager();
            entityManager.getTransaction().begin();
            Query query = entityManager.createQuery("SELECT a FROM A1 a");
            List<A1> listaA1 = query.getResultList();
            for (A1 a : listaA1) {
                System.out.println("Codigo A: " + a.getCodigo());
                System.out.println("Nome A: " + a.getNome());
                if (a.getB1() != null) {
                    System.out.println("Codigo B: " + a.getB1().getCodigo());
                    System.out.println("Nome B: " + a.getB1().getNome());
                }
                System.out.println("-----");
            }
            entityManager.getTransaction().commit();
        } catch (Exception ex) {
            System.out.println(ex.toString());
        }
    }
}

```

## RELACIONAMENTO 1XN (ONE TO MANY)



Neste caso, A2 referencia uma lista de B2 devido possuir relação one to Many (um para muitos). Cada instancia de A2 pode estar relacionado a varias estancias de B2. Pode ser representado por list, ArrayList, set, etc.

### Anotação @OneToMany

Fecth = FecthTyp.EAGER irá recuperar uma lista de entidades relacionada a entidade de B2.  
Fecth = FecthTyp.Lazy irá recuperar a lista apenas quando solicitado atraves de getListB2()

Quando vamos obter atraves do select os dados de A2 e B2 temos de realizar dois SELECT, e em alguns casos o custo de fazer isso é maior que de fazer um JOIN com as duas tabelas.  
Logo, deve-se analisar com cautela quando se usar.

Implementação B2

```

package vo;

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "tabela_b2")
public class B2 implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private int codigo;

    @Column(length = 40, nullable = false)
    private String nome;

    public B2() {
    }

    public B2(String nome) {
        this.nome = nome;
    }

    public int getCodigo() {
        return codigo;
    }
}

```

*Implementação A2*

```

package vo;
import java.io.Serializable;
import java.util.List;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@Entity
@Table(name = "tabela_a2")
public class A2 implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private int codigo;

    @Column(length = 40, nullable = false)
    private String nome;

    @OneToMany(fetch= FetchType.LAZY)
    private List<B2> listaB2;

    public A2() {
    }

    public A2(String nome) {

```

*Implementação de main para inserção de dados em B2*

```

package apps;
import javax.persistence.EntityManager;
import util.EntityManagerUtil;
import vo.B2;
public class AppB2i {
    public static void main(String args[]) {
        try {
            EntityManager em = EntityManagerUtil.getEntityManager();
            em.getTransaction().begin();

            B2 b = new B2("b1");
            em.persist(b);

            b = new B2("b2");
            em.persist(b);

            b = new B2("b3");
            em.persist(b);

            em.getTransaction().commit();
        } catch (Exception ex) {
            System.out.println(ex.toString());
        }
    }
}

```

Data Output			Explain	Messages	Notifications
	codigo [PK] integer	nome character varying (40)			
1	1252	b2			
2	1253	b3			
3	1251	b1			

#### Implementação de dados para A1

```

package apps;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.Query;
import util.EntityManagerUtil;
import vo.A2;
import vo.B2;
public class AppA2i {
    public static void main(String args[]) {
        try {
            EntityManager entityManager = EntityManagerUtil.getEntityManager();
            entityManager.getTransaction().begin();

            Query query = entityManager.createQuery("SELECT b FROM B2 b");
            List<B2> listaB = query.getResultList();

            A2 a = new A2("a1");
            a.setListaB2(listaB.subList(0, 2)); //A1 agrega uma lista de instancias de B2

            entityManager.persist(a);

            a = new A2("a2");
            a.setListaB2(listaB.subList(2, 3)); //A1 agrega uma lista de instancias de B2

            entityManager.persist(a);
        }
    }
}

```

Nesse caso irá setar uma sublista de B2. subList(0,2) pega duas posições, posição 0 e 1, subList(2,3) pega a posição 2.

A forma nativa da JPA tratar essa relação oneToMany é criando uma nova tabela que contenha a relação entre as duas tabelas, chave primária de A2 e dados de B2 setados como solicitado na inserção.

Data Output			Explain	Messages	Notifications
	codigo [PK] integer	nome character varying (40)			
1	1352	a2			
2	1351	a1			

Data Output			Explain	Messages	Notifications
	a2_codigo [PK] integer	listab2_codigo [PK] integer			
1	1352	1251			
2	1351	1252			
3	1351	1253			

Observe a chave primaria 1351 esta relacionada com duas chaves de itens devido a inserção de dois itens da sublist.

#### Implementação do busca

Nesse caso, ira fazer dois for, um para apresentar o resultado do select principal e outro para a potencial lista de B2

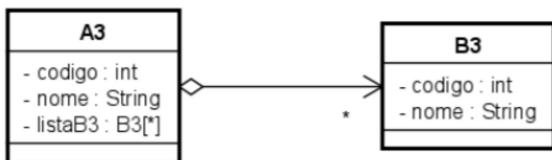
```
package apps;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.Query;
import util.EntityManagerUtil;
import vo.A2;
import vo.B2;

public class AppA2c {

    public static void main(String args[]) {
        try {
            EntityManager entityManager = EntityManagerUtil.getEntityManager();
            entityManager.getTransaction().begin();
            Query query = entityManager.createQuery("SELECT a FROM A2 a");
            List<A2> listaA2 = query.getResultList();
            for (A2 a : listaA2) {
                System.out.println("Codigo A: " + a.getCodigo());
                System.out.println("Nome A: " + a.getNome());
                for (B2 b : a.getListab2()) {
                    System.out.println("\tCodigo B: " + b.getCodigo());
                    System.out.println("\tNome B: " + b.getNome());
                }
                System.out.println("-----");
            }
            entityManager.getTransaction().commit();
        } catch (Exception ex) {
            System.out.println(ex.toString());
        }
    }
}
```

#### RELACIONAMENTO 1XN (oneToMany) COM COLUNA DE JUNÇÃO



Funciona de maneira similar ao caso anterior, entretanto agora ira criar uma coluna com a relação em vez de uma tabela pegando as chaves primarias das duas.

#### Anotação @JoinColumn(name="codigo\_a3")

A notação informa que a relação deve ser resolvida com uma chave estrangeira no lado N da relação. Name define o nome da coluna criada a partir da relação N. Sempre sera

armazenado o lado N nesta tabela.

*Implementação de B3*

```
package vo;

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "tabela_b3")
public class B3 implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private int codigo;

    @Column(length = 40, nullable = false)
    private String nome;

    public B3() {
    }

    public B3(String nome) {
        this.nome = nome;
    }

    public int getCodigo() {
        return codigo;
    }
}
```

*Implementação de A3*

```
package vo;
import java.io.Serializable;
import java.util.List;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@Entity
@Table(name = "tabela_a3")
public class A3 implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private int codigo;

    @Column(length = 40, nullable = false)
    private String nome;

    @OneToMany(fetch= FetchType.LAZY)
    @JoinColumn(name = "codigo_a3")
    private List<B3> listaB3;

    public A3() {
    }
}
```

Observe a utilização do @JoinColumn

*Implementação de main para inserção de dados em B3*

```

package apps;
import javax.persistence.EntityManager;
import util.EntityManagerUtil;
import vo.B3;
public class AppB3i {
    public static void main(String args[]) {
        try {
            EntityManager em = EntityManagerUtil.getEntityManager();
            em.getTransaction().begin();

            B3 b = new B3("b1");
            em.persist(b);

            b = new B3("b2");
            em.persist(b);

            b = new B3("b3");
            em.persist(b);

            em.getTransaction().commit();
        } catch (Exception ex) {
            System.out.println(ex.toString());
        }
    }
}

```

*Implementação de main para inserção de dados em A3*

```

package apps;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.Query;
import util.EntityManagerUtil;
import vo.A3;
import vo.B3;
public class AppA3i {
    public static void main(String args[]) {
        try {
            EntityManager entityManager = EntityManagerUtil.getEntityManager();
            entityManager.getTransaction().begin();

            Query query = entityManager.createQuery("SELECT b FROM B3 b");
            List<B3> listaB = query.getResultList();

            A3 a = new A3("a1");
            a.setListaB3(listaB.subList(0, 2)); //A3 adiciona uma lista de instâncias de B3

            entityManager.persist(a);

            a = new A3("a2");
            a.setListaB3(listaB.subList(2, 3)); //A3 adiciona uma lista de instâncias de B3

            entityManager.persist(a);

            entityManager.getTransaction().commit();
        } catch (Exception ex) {
            System.out.println(ex.toString());
        }
    }
}

```

Data Output			Explain	Messages	Notifications
	código [PK] integer	nome character varying (40)			
1	1552	a2			
2	1551	a1			

*Banco de dados de B3*

Data Output				Explain	Messages	Notifications
	codigo [PK] integer	nome character varying (40)	codigo_a3 integer			
1	1451	b1	1552			
2	1452	b2	1551			
3	1453	b3	1551			

Observe que a coluna fica na tabela B3 inserindo nela a chave estrangeira de A3.

#### *Implementação para recuperação de informações*

```
package apps;

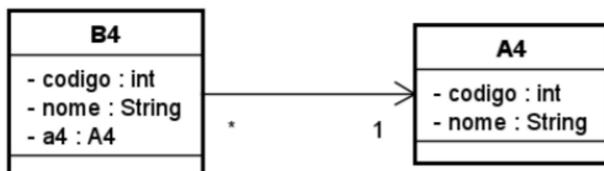
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.Query;
import util.EntityManagerUtil;
import vo.A3;
import vo.B3;

public class AppA3c {

    public static void main(String args[]) {
        try {
            EntityManager entityManager = EntityManagerUtil.getEntityManager();
            entityManager.getTransaction().begin();
            Query query = entityManager.createQuery("SELECT a FROM A3 a");
            List<A3> listaA3 = query.getResultList();
            for (A3 a : listaA3) {
                System.out.println("Codigo A: " + a.getCodigo());
                System.out.println("Nome A: " + a.getNome());
                for (B3 b : a.getListB3()) {
                    System.out.println("\tCodigo B: " + b.getCodigo());
                    System.out.println("\tNome B: " + b.getNome());
                }
                System.out.println("-----");
            }
            entityManager.getTransaction().commit();
        } catch (Exception ex) {
            System.out.println(ex.toString());
        }
    }
}
```

## RELACIONAMENTO NX1

B4 referencia uma instância de A4, B4 agrega uma instância de A4.



#### Anotação @ManyToOne

A chave estrangeira fica do lado N.

#### *Implementação B4*

```

package vo;

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

@Entity
@Table(name = "tabela_b4")
public class B4 implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private int codigo;

    @Column(length = 40, nullable = false)
    private String nome;

    @ManyToOne(fetch = FetchType.EAGER)
    private A4 a4;

    public B4() {
    }

    public B4(String nome) {
        ...
    }
}

```

#### Implementação para inserir 3 objetos do tipo A4 no banco de dados

```

package apps;
import javax.persistence.EntityManager;
import util.EntityManagerUtil;
import vo.A4;
public class AppA4i {
    public static void main(String args[]) {
        try {
            EntityManager em = EntityManagerUtil.getEntityManager();
            em.getTransaction().begin();

            A4 a = new A4("a1");
            em.persist(a);

            a = new A4("a2");
            em.persist(a);

            a = new A4("a3");
            em.persist(a);

            em.getTransaction().commit();
        } catch (Exception ex) {
            System.out.println(ex.toString());
        }
    }
}

```

Data Output			Explain	Messages	Notifications
	codigo [PK] integer	nome character varying (40)			
1	1652	a2			
2	1653	a3			
3	1651	a1			

#### Implementação de inserção de 3 objetos do tipo B4

```

package apps;
import javax.persistence.EntityManager;
import javax.persistence.Query;
import util.EntityManagerUtil;
import vo.A4;
import vo.B4;
public class AppB4i {
    public static void main(String args[]) {
        try {
            EntityManager entityManager = EntityManagerUtil.getEntityManager();
            entityManager.getTransaction().begin();

            Query query = entityManager.createQuery("SELECT a FROM A4 a WHERE a.nome = 'a1'");
            A4 a = (A4)query.getSingleResult();

            B4 b = new B4("b1");
            b.setA4(a); //agrega a1
            entityManager.persist(b);

            b = new B4("b2");
            b.setA4(a); //agrega a1
            entityManager.persist(b);

            query = entityManager.createQuery("SELECT a FROM A4 a WHERE a.nome = 'a2'");
            a = (A4)query.getSingleResult();

            b = new B4("b3");

            b.setA4(a); //agrega a2
            entityManager.persist(b);

            b = new B4("b4");
            b.setA4(a); //agrega a2
            entityManager.persist(b);

            entityManager.getTransaction().commit();
        } catch (Exception ex) {
            System.out.println(ex.toString());
        }
    }
}

```

Data Output				Explain	Messages	Notifications
	codigo [PK] integer	nome character varying (40)	a4_codigo integer			
1	1751	b1	1651			
2	1752	b2	1651			
3	1754	b4	1652			
4	1753	b3	1652			

*Implementação de busca de dados*

```

package apps;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.Query;
import util.EntityManagerUtil;
import vo.B4;

public class AppB4c {

    public static void main(String args[]) {
        try {
            EntityManager entityManager = EntityManagerUtil.getEntityManager();
            entityManager.getTransaction().begin();
            Query query = entityManager.createQuery("SELECT b FROM B4 b");
            List<B4> listaB4 = query.getResultList();
            for (B4 b : listaB4) {
                System.out.println("Codigo B: " + b.getCodigo());
                System.out.println("Nome B: " + b.getNome());
                if (b.getA4() != null) {
                    System.out.println("\tCodigo A: " + b.getA4().getCodigo());
                    System.out.println("\tNome A: " + b.getA4().getNome());
                }
                System.out.println("-----");
            }
            entityManager.getTransaction().commit();
        } catch (Exception ex) {
            System.out.println(ex.toString());
        }
    }
}

```

## RELACIONAMENTO NXN (ManyToMany)



A5 referencia muitas instâncias de B5

B5 referencia muitas instâncias de A5

### Anotação @ManyToMany

Resolvido com duas tabelas de relacionamento (da mesma maneira que 1XN), criando uma tabela para cada relação

*Implementação de A5*

```

package vo;
import java.io.Serializable;
import java.util.List;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToMany;
import javax.persistence.Table;

@Entity
@Table(name = "tabela_a5")
public class A5 implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private int codigo;

    @Column(length = 40, nullable = false)
    private String nome;

    @ManyToMany(fetch= FetchType.EAGER)
    private List<B5> listaB5;

    public A5() {
    }

    public A5(String nome) {
}

```

Como A5 esta relacionado com varios N, então crio uma lista de B5.

#### *Implementação de B5*

```

package vo;
import java.io.Serializable;
import java.util.List;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToMany;
import javax.persistence.Table;

@Entity
@Table(name = "tabela_b5")
public class B5 implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private int codigo;

    @Column(length = 40, nullable = false)
    private String nome;

    @ManyToMany(fetch= FetchType.EAGER)
    private List<A5> listaA5;

    public B5() {
    }

    public B5(String nome) {
}

```

B5 esta ligado a varios N, logo possui uma lista de A5.

#### *Implementação para inserção de dados em A5*

```

package apps;
import javax.persistence.EntityManager;
import util.EntityManagerUtil;
import vo.A5;
public class AppA5i {
    public static void main(String args[]) {
        try {
            EntityManager em = EntityManagerUtil.getEntityManager();
            em.getTransaction().begin();

            A5 a = new A5("a1");
            em.persist(a);

            a = new A5("a2");
            em.persist(a);

            a = new A5("a3");
            em.persist(a);

            a = new A5("a4");
            em.persist(a);

            em.getTransaction().commit();
        } catch (Exception ex) {
            System.out.println(ex.toString());
        }
    }
}

```

Data Output			Explain	Messages	Notifications
	codigo [PK] integer	nome character varying (40)			
1	1951	a1			
2	1954	a4			
3	1953	a3			
4	1952	a2			

Realização da persistencia de A's

#### Implementação de B5

```

package apps;
import javax.persistence.EntityManager;
import util.EntityManagerUtil;
import vo.B5;
public class AppB5i {
    public static void main(String args[]) {
        try {
            EntityManager em = EntityManagerUtil.getEntityManager();
            em.getTransaction().begin();

            B5 b = new B5("b1");
            em.persist(b);

            b = new B5("b2");
            em.persist(b);

            b = new B5("b3");
            em.persist(b);

            b = new B5("b4");
            em.persist(b);

            b = new B5("b5");
            em.persist(b);

            em.getTransaction().commit();
        } catch (Exception ex) {
            System.out.println(ex.toString());
        }
    }
}

```

Data Output			Explain	Messages	Notifications
	codigo [PK] integer	nome character varying (40)			
1	1851	b1			
2	1855	b5			
3	1852	b2			
4	1854	b4			
5	1853	b3			

Implementação de aplicação para relacionamento entre entidades

#### INSERINDO DADOS EM A5

```
package apps;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.Query;
import util.EntityManagerUtil;
import vo.A5;
import vo.B5;

public class AppAB5i {

    public static void main(String args[]) {
        try {
            EntityManager em = EntityManagerUtil.getEntityManager();
            em.getTransaction().begin();

            Query query = em.createQuery("SELECT b FROM B5 b");
            List<B5> listaB = query.getResultList();

            query = em.createQuery("SELECT a FROM A5 a");
            List<A5> listaA = query.getResultList();

            A5 aTemp = listaA.get(0);
            aTemp.setListaB5(listaB.subList(0, 3));
            em.persist(aTemp);

            aTemp = listaA.get(1);
            aTemp.setListaB5(listaB.subList(3, 5));
            em.persist(aTemp);
        }
    }
}
```

Observe que é criado dois select para obter duas listas, neste caso estamos inserindo valores em listaA, para isso pegamos o primeiro valor get(0) e inserimos os tres primeiros valores da listaB, e para o segundo valor de A5. Já para o segundo valor get(1) inserimos mais dois valores.

Data Output			Explain	Messages	Notifications
	a5_codigo [PK] integer	listab5_codigo [PK] integer			
1	1951	1855			
2	1951	1851			
3	1951	1852			
4	1954	1853			
5	1954	1854			

#### INSERINDO DADOS EM B5

```

package apps;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.Query;
import util.EntityManagerUtil;
import vo.A5;
import vo.B5;

public class AppBA5i {

    public static void main(String args[]) {
        try {
            EntityManager em = EntityManagerUtil.getEntityManager();
            em.getTransaction().begin();

            Query query = em.createQuery("SELECT a FROM A5 a");
            List<A5> listaA = query.getResultList();

            query = em.createQuery("SELECT b FROM B5 b");
            List<B5> listaB = query.getResultList();

            B5 bTemp = listaB.get(0);
            bTemp.setListaA5(listaA.subList(0, 2));
            em.persist(bTemp);

            bTemp = listaB.get(1);
            bTemp.setListaA5(listaA.subList(1, 3));
            em.persist(bTemp);

            bTemp = listaB.get(2);
            bTemp.setListaA5(listaA.subList(2, 4));
            em.persist(bTemp);

            em.getTransaction().commit();
        } catch (Exception ex) {
            System.out.println(ex.toString());
        }
    }
}

```

Data Output			Explain	Messages	Notifications
	b5_codigo [PK] integer	listaa5_codigo [PK] integer			
1	1851	1951			
2	1851	1954			
3	1852	1953			
4	1852	1952			
5	1855	1954			
6	1855	1953			

IMPLEMENTAÇÃO DE BUSCA DE DADOS

```

package apps;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.Query;
import util.EntityManagerUtil;
import vo.A5;
import vo.B5;

public class AppAB5c {

    public static void main(String args[]) {
        try {
            EntityManager entityManager = EntityManagerUtil.getEntityManager();
            entityManager.getTransaction().begin();

            Query query = entityManager.createQuery("SELECT a FROM A5 a");
            List<A5> listaA5 = query.getResultList();
            for (A5 a : listaA5) {
                System.out.println("Codigo A: " + a.getCodigo());
                System.out.println("Nome A: " + a.getNome());
                System.out.println("-----");
                for (B5 b : a.getListB5()) {
                    System.out.println("\tCodigo B: " + b.getCodigo());
                    System.out.println("\tNome B: " + b.getNome());
                    System.out.println("\t-----");
                    for (A5 ab : b.getListA5()) {
                        System.out.println("\t\tCodigo A: " + ab.getCodigo());
                        System.out.println("\t\tNome A: " + ab.getNome());
                    }
                }
            }
        }
    }
}

```

## RELACIONAMENTO N X N COM UMA TABELA DE JUNÇÃO



Neste momento, assim como nas outras relações , quando não queremos que a JPA crie tabela de relações podemos utilizar o join.

Utiliza a notação (além de manytomany) utiliza-se a anotação

`@JoinTable(name, joinsColumns=@JoinColumn(name) = inverseJoinColumns=@JoinColumn(name = ))`

### Implementando A6

```

package vo;
import java.io.Serializable;
import java.util.List;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.Table;

@Entity
@Table(name = "tabela_a6")
public class A6 implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private int codigo;

    @Column(length = 40, nullable = false)
    private String nome;

    @ManyToMany(fetch= FetchType.EAGER)
    @JoinTable(name = "tabela_a6b6",
               joinColumns = {@JoinColumn(name = "chavep_a6")},
               inverseJoinColumns = {@JoinColumn(name = "chavep_b6")})
    private List<B6> listaB6;
}

```

Observe que cria-se uma tabela para as relações de a6b6, definimos o nome das colunas, uma para o sentido direto a6b6 e outra para inverso b6a6.

#### *Implementando B6*

```
package vo;
import java.io.Serializable;
import java.util.List;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

@Entity
@Table(name = "tabela_b6")
public class B6 implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private int codigo;

    @Column(length = 40, nullable = false)
    private String nome;

    @ManyToOne(mappedBy = "listaB6", fetch= FetchType.EAGER)
    private List<A6> listaA6;

    public B6() {
    }

    public B6(String nome) {
```

Como já foi especificado que sera criado uma joinTable em A6 não é necessário repetir o processor em B6, basta indicar que esta sendo mapeada por B6.

#### *Inserindo valore em A6*

```
package apps;
import javax.persistence.EntityManager;
import util.EntityManagerUtil;
import vo.A6;
public class AppA6i {
    public static void main(String args[]) {
        try {
            EntityManager em = EntityManagerUtil.getEntityManager();
            em.getTransaction().begin();

            A6 a = new A6("a1");
            em.persist(a);

            a = new A6("a2");
            em.persist(a);

            a = new A6("a3");
            em.persist(a);

            a = new A6("a4");
            em.persist(a);

            em.getTransaction().commit();
        } catch (Exception ex) {
            System.out.println(ex.toString());
        }
    }
}
```

	codigo [PK] integer	nome character varying(40)
1	151	a1
2	152	a2
3	153	a3
4	154	a4
*		

#### *Inserindo valores em B6*

```

package apps;
import javax.persistence.EntityManager;
import util.EntityManagerUtil;
import vo.B6;
public class AppB6i {
    public static void main(String args[]) {
        try {
            EntityManager em = EntityManagerUtil.getEntityManager();
            em.getTransaction().begin();

            B6 b = new B6("b1");
            em.persist(b);

            b = new B6("b2");
            em.persist(b);

            b = new B6("b3");
            em.persist(b);

            b = new B6("b4");
            em.persist(b);

            b = new B6("b5");
            em.persist(b);

            em.getTransaction().commit();
        } catch (Exception ex) {
            System.out.println(ex.toString());
        }
    }
}

```

	<b>codigo [PK] integer</b>	<b>nome character varying(40)</b>
<b>1</b>	251	b1
<b>2</b>	252	b2
<b>3</b>	253	b3
<b>4</b>	254	b4
<b>5</b>	255	b5
*		

*Inserindo B6 em A6*

```

package apps;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.Query;
import util.EntityManagerUtil;
import vo.A6;
import vo.B6;

public class AppAB6i {

    public static void main(String args[]) {
        try {
            EntityManager em = EntityManagerUtil.getEntityManager();
            em.getTransaction().begin();

            Query query = em.createQuery("SELECT b FROM B6 b");
            List<B6> listaB = query.getResultList();

            query = em.createQuery("SELECT a FROM A6 a");
            List<A6> listaA = query.getResultList();

            A6 aTemp = listaA.get(0);
            aTemp.setListaB6(listaB.subList(0, 3));
            em.persist(aTemp);

            aTemp = listaA.get(1);
            aTemp.setListaB6(listaB.subList(3, 5));
            em.persist(aTemp);
        }
    }
}

```

Tabela de relações criada

	chavep_a6 [PK] integer	chavep_b6 [PK] integer
1	152	253
2	152	255
3	153	251
4	153	252
5	153	254
*		

Inserindo A6 em B6

```
public static void main(String args[]) {
    try {
        EntityManager em = EntityManagerUtil.getEntityManager();
        em.getTransaction().begin();

        Query query = em.createQuery("SELECT a FROM A6 a");
        List<A6> listaA = query.getResultList();

        query = em.createQuery("SELECT b FROM B6 b");
        List<B6> listaB = query.getResultList();

        B6 bTemp = listaB.get(0);
        bTemp.setListaA6(listaA.subList(0, 2));
        em.persist(bTemp);

        bTemp = listaB.get(1);
        bTemp.setListaA6(listaA.subList(1, 3));
        em.persist(bTemp);

        bTemp = listaB.get(2);
        bTemp.setListaA6(listaA.subList(2, 4));
        em.persist(bTemp);

        em.getTransaction().commit();
    } catch (Exception ex) {
        System.out.println(ex.toString());
    }
}
```

	chavep_a6 [PK] integer	chavep_b6 [PK] integer
1	152	253
2	152	255
3	153	251
4	153	252
5	153	254
*		

Deve-se entender que, esta tabela criada com a relação entre A6 e B6 vale para a inserção de ambos os lados e em caso se relação repetida independente do lado, não sera repetido o valor na tabela.