

EAPLI  
Teórico-Prática

# **Mapeamento Objeto/Relacional**

## **JPA – Parte 2**

## Object-Relational Mapping (ORM)

# Associações entre classes

# Associações

- Em Java existem associações entre classes
- Em bases de dados há referências entre tabelas.
- As referências entre tabelas podem ser modeladas de dois modos:
  - usando uma **Join Column**
  - usando uma **Join Table**

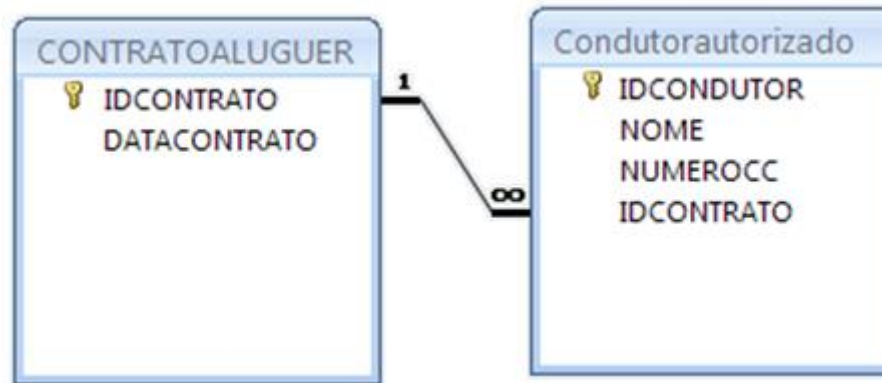
# Associações

- O mapeamento efetuado pelo JPA por omissão (na ausência de anotações e de ficheiro de mapeamento xml) é o seguinte:

Associação		SGDB-R
OneToOne	→	Join Column
ManyToOne	→	Join Column
OneToMany	→	Join Table
ManyToMany	→	Join Table

# Associação One-to-many

- Contrato Aluguer (**one**) contém vários Condutores Autorizados (**many**).



# Associação One-to-many

- ContratoAluguer → Lista de CondutorAutorizado (bidireccional)

@Entity

```
public class ContratoAluguer {
```

@Id

@GeneratedValue

```
private int idContrato;
```

@Temporal(TemporalType.DATE)

```
private Date dataContrato;
```

*mappedBy* indica relação bidireccional

@OneToMany(mappedBy="contrato", cascade = CascadeType.ALL)

```
private List<CondutorAutorizado> condutoresAutorizados=  
    new ArrayList<CondutorAutorizado>();
```

(...)

```
}
```

*CondutorAutorizado* é parte integrante de um Contrato

# Cascading

- As associações entre entidades dependem, geralmente, da existência de uma das entidades.
- Por exemplo:
  - Relação: ContratoAluguer-ConductorAutorizado
  - Sem ContratoAluguer a entidade ConductorAutorizado não tem significado próprio
  - Logo remover ContratoAluguer deve implicar remover ConductorAutorizado
- Cascading é o processo que permite **executar a mesma ação numa entidade (pai) e nas suas associadas (filhos)**

# Cascading

- **JPA Cascade Types:**
  - **none** – nothing happens. Default value
  - **ALL** - propagates all operations
  - **PERSIST** - propagates the persist operation
  - **MERGE** - propagates the merge operation
  - **REMOVE** - propagates the remove operation
  - **REFRESH** - reloads the parent and the child entities from the database
  - **DETACH** - removes the parent and the child entities from the persistent context.



# Fetching

- **Associações podem ter como FetchType:**
  - **LAZY** – obtido quando solicitado (aka “lazy loaded”)
    - One-to-many
    - Many-to-many
  - **EAGER** – obtido quando a entidade é carregada
    - One-to-One
    - Many-to-One
- Propriedades também podem ser carregadas com **lazy load**
  - Por omissão o modo de carregamento utilizado é *eager*

# Fetching - Lazy

```
@Entity
public class CondutorAutorizado {
    @Id
    @GeneratedValue
    private int idCondutor;

    private String nome;
    private String numeroCartaConducao;

    @ManyToOne(fetch=FetchType.LAZY)
    @JoinColumn(name="contrato_id")
    private ContratoAluguer contrato; // ligação bidireccional

    (...)
}
```

Por omissão seria utilizado  
*eager load*

Contrato só será carregado  
quando for usado

# Fetching - Lazy

```
@Entity
public class Automovel {
    @Id
    private int pk;

    @Column(unique=true)
    String matricula;

    @Lob
    @Basic(fetch=FetchType.LAZY)
    @Column(name="Foto")
    private byte[] fotografia;

    (...)
}
```

@Lob – Large Object

Por omissão seria utilizado  
*eager load*

# One-to-many + Cascading

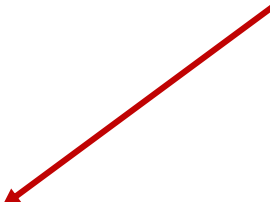
## a) Inserir

```
EntityManager manager=PersistenceInit.getEntityManager();  
(...)  
// Cria contrato  
ContratoAluguer contrato=new ContratoAluguer();  
// Cria dois objectos CondutorAutorizado  
CondutorAutorizado condutor1= new CondutorAutorizado("Manuel");  
CondutorAutorizado condutor2= new CondutorAutorizado("Carlos");  
// Adiciona os condutores ao set da classe contrato  
contrato.addCondutorAutorizado(condutor1);  
contrato.addCondutorAutorizado(condutor2);  
// Grava o objecto contrato e condutores autorizados e  
// estabelece a relação entre eles na BD  
manager.getTransaction().begin();  
manager.persist(contrato);  
manager.getTransaction().commit();
```

# One-to-many + Cascading

## b) Consultar

O JPA faz o mapeamento para o objecto **contrato**, com todos os seus atributos, incluindo a lista de condutores autorizados



```
id=(...)
```

```
// Utilização do método find por id
```

```
contrato=manager.find(ContratoAluguer.class, id);
```

```
// Obtem e exhibe a lista de condutores do contrato
```

```
List<ConductorAutorizado> condutores= contrato.getCondutores();
```

```
for (int i=0;i<condutores.size();i++){
```

```
    System.out.println(condutores.get(i).nome());
```

```
}
```

```
manager.close();
```

```
(...)
```

# Associação One-to-many

## ■ ***One-to-many*** unidirecional com *join column*

@Entity

```
public class ContratoAluguer{
```

```
    @Id
```

```
    @GeneratedValue
```

```
    private int idcontrato;
```

```
    @Temporal(javax.persistence.TemporalType.DATE)
```

```
    private Date datacontrato;
```

```
    @OneToMany(cascade = CascadeType.ALL)
```

```
    @JoinColumn(name="ContratoID")
```

```
    private List<CondutorAutorizado> condutoresAutorizados=  
        new ArrayList<CondutorAutorizado>();
```

```
    (...)
```


```
}
```

# Associação One-to-many

## ■ *One-to-many* unidirecional com *join column*

@Entity

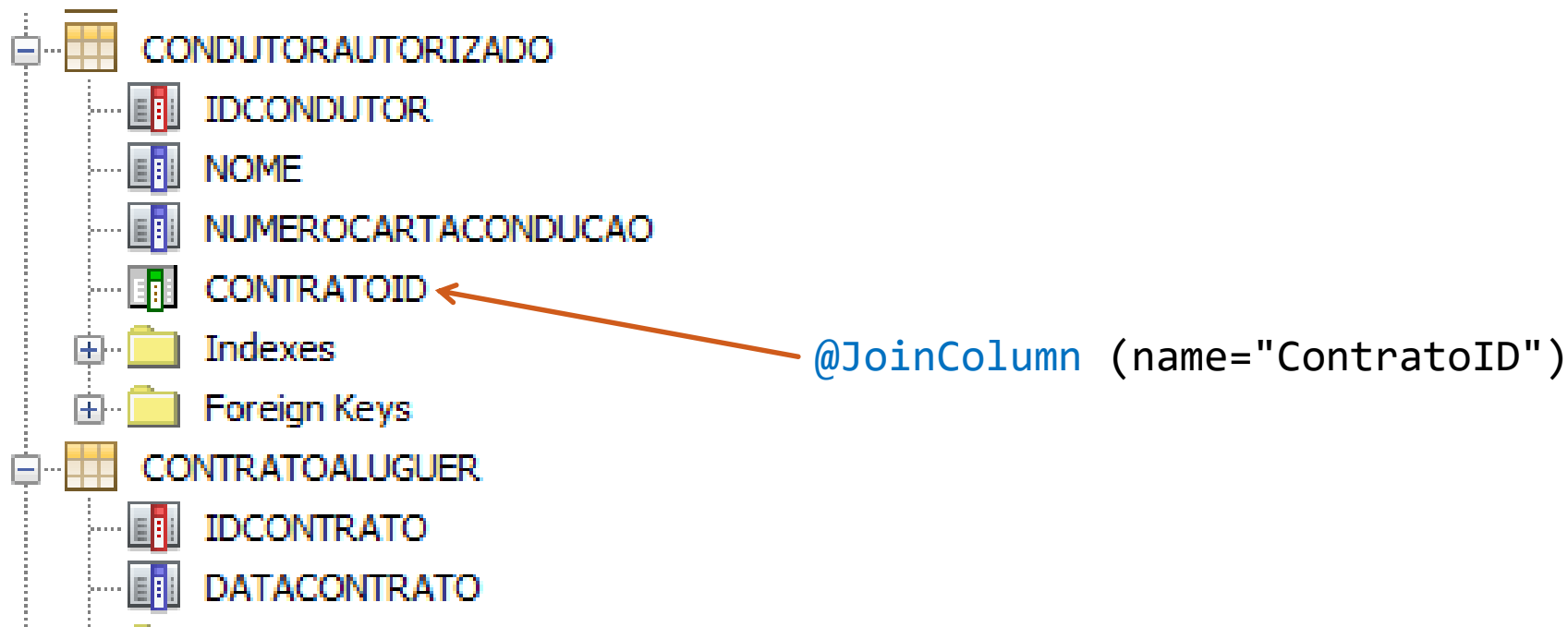
```
public class CondutorAutorizado {  
    @Id  
    @GeneratedValue  
    private int idcondutor;  
    private String nome;  
    private String numeroCartaConducao;  
  
    (...)  
}
```



A classe CondutorAutorizado não tem uma referência para o contrato. A associação é unidirecional.

# Associação One-to-many

## ■ Tabelas criadas

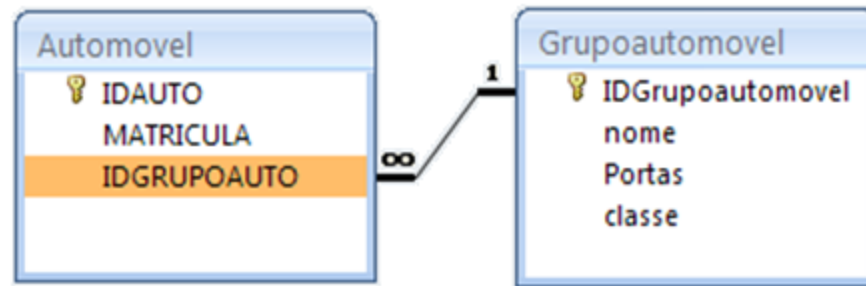


- A coluna de junção fica na tabela CondutorAutorizado, sendo uma **chave estrangeira** para ContratoAluguer



# Associação many-to-one

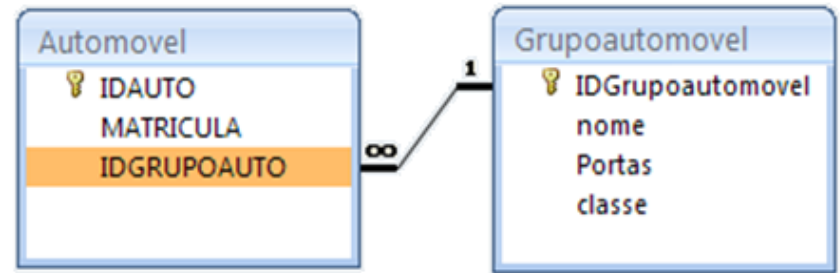
- **Automovel** pertence a um **GrupoAutomovel**



```
public class Automovel {  
    private int idAuto;  
    private String matricula;  
    private Grupoautomovel grupo;  
  
    (...)  
}
```

```
public class GrupoAutomovel {  
    private int idGrupoAutomovel;  
    private String nome;  
    private Integer portas;  
    private String classe;  
  
    (...)  
}
```

# Associação One-to-many



@Entity

```
public class Automovel {
```

@Id

```
private int idauto;
```

```
String matricula;
```

```
@ManyToOne (/* by default CascadeType.NONE */)
```

```
private GrupoAutomovel grupo;
```

```
public Automovel() {  
}
```

```
(...)
```

```
}
```

O conceito GrupoAutomovel existe independente da existencia, ou não, de Automoveis

# Associação many-to-one

## ■ Gravar objeto Automovel

(...)

```
EntityManager manager= PersistenceInit.getEntityManager();
```

```
GrupoAutomovel grupo= new GrupoAutomovel("C",5,"utilitario");
```

```
Automovel auto= new Automovel("15-BM-17",grupo);
```

```
manager.getTransaction().begin();
```

```
manager.persist(grupo);
```

```
manager.persist(auto);
```

```
manager.getTransaction().commit();
```



O Grupo tem de ser persistido  
antes do Automovel

(...)

# Persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_2_1.xsd">
    <persistence-unit name="Demo_ORMPU" transaction-type="RESOURCE_LOCAL">

        <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>

        <class>demo_orm.dominio.Automovel</class>
        <class>demo_orm.dominio.GrupoAutomovel</class>

        <properties>
            <property name="javax.persistence.jdbc.driver" value="org.h2.Driver"/>
            <property name="javax.persistence.jdbc.url" value="jdbc:h2:./bd/rentacar" />
            <property name="javax.persistence.jdbc.user" value="" />
            <property name="javax.persistence.jdbc.password" value="" />
            <property name="javax.persistence.schema-generation.database.action"
                value="create"/>
        </properties>
    </persistence-unit>
</persistence>
```

# Mapeamento por omissão

- O mapeamento efetuado pelo JPA por omissão (na ausência de anotações e de ficheiro xml de mapeamento) é o seguinte:

Associação	SGBD-R	Fetching	Cascade
OneToOne	Join Column	Eager	None
ManyToOne			
OneToMany	Join Table	Lazy	
ManyToMany			

# Querying - considerações

- JPQL pode ser utilizado para expressões de pesquisa complexas que levam em linha de conta os objetos associados;
- Ter em atenção que estaremos sempre a referir o **nome dos atributos java** de cada classe e não os nomes das colunas na BD;
- Ao utilizar parametros, o valor deve ser da **classe correta**.

# Querying

- **Obter todos os contratos de aluguer que tenham sido realizados no mês de fevereiro de 2020 para automoveis do grupo “B”.**

```
SELECT ca FROM ContratoAutomovel WHERE  
        ca.automovel.grupoAutomovel = :ga AND  
        ca.dataContrato BETWEEN :d1 AND :d2
```

```
// obter o grupo automovel desejado
```

```
GrupoAutomovel g = em.find(2, GrupoAutomovel.class);
```

```
Query q = em.createQuery(...);
```

```
q.setParameter("ga", g);
```

```
q.setParameter("d1", new Date(2020, 02, 01));
```

```
q.setParameter("d2", new Date(2020, 02, 29));
```

# Querying

## ■ Em alternativa:

```
SELECT ca FROM ContratoAutomovel WHERE  
    ca.automovel.grupoAutomovel.nome = :ga AND  
    ca.dataContrato BETWEEN :d1 AND :d2
```

```
Query q = em.CreateQuery(...);  
q.setParameter("ga", "B");  
q.setParameter("d1", new Date(2020, 02, 01));  
q.setParameter("d2", new Date(2020, 02, 29));
```