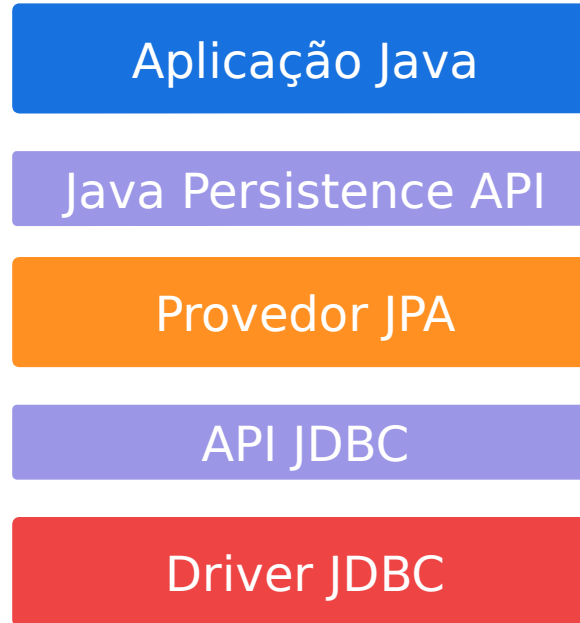
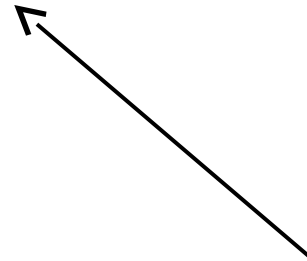


JPA - Arquitetura

Hibernate
TopLink
Kodo
OpenJPA
etc...

MySQL
Postgres
JavaDB
Oracle
Etc...

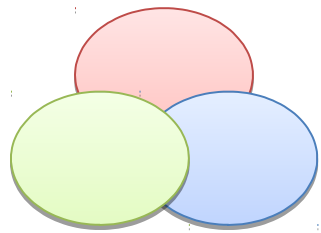


Banco de Dados



Arquitetura

- XML
(deprecated)
- Anotações



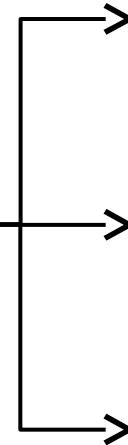
Objetos



Mapeamento Objeto
Relacional



Tabelas
Banco de
dados Relacional



MySQL

PostgreSQL

ORACLE
DATABASE

persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
  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_0.xsd">
  <persistence-unit name="Banco-ServicePU">

    <provider>org.hibernate.ejb.HibernatePersistence</provider>

    <class>threeway.projeto.modelo.Agencia</class>
    <class>threeway.projeto.modelo.Banco</class>
    <class>threeway.projeto.modelo.Cliente</class>
    <class>threeway.projeto.modelo.Conta</class>
    <class>threeway.projeto.modelo.Pessoa</class>
    <class>threeway.projeto.modelo.Transacao</class>

    <properties>
      <property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver" />
      <property name="javax.persistence.jdbc.url" value="jdbc:postgresql://localhost:5432/threeway" />
      <property name="javax.persistence.jdbc.user" value="postgres" />
      <property name="javax.persistence.jdbc.password" value="123456" />
      <property name="hibernate.hbm2ddl.auto" value="update" />
      <property name="hibernate.show_sql" value="true" />
      <property name="hibernate.format_sql" value="true" />
      <property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect" />
    </properties>
  </persistence-unit>
</persistence>
```

Implementação da JPA a ser utilizada

Classes persistidas

Conexão com banco e configurações do Hibernate

Persistence.xml

Conexão com o Banco de Dados

```
<property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver" />
<property name="javax.persistence.jdbc.url" value="jdbc:postgresql://localhost:5432/threeway" />
<property name="javax.persistence.jdbc.user" value="postgres" />
<property name="javax.persistence.jdbc.password" value="123456" />
```

driver

url

senha

usuário

Exibe a SQL bem
formatada no console.

Propriedades do Hibernate

```
<property name="hibernate.hbm2ddl.auto" value="update" />
<property name="hibernate.show_sql" value="true" />
<property name="hibernate.format_sql" value="true" />
<property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect" />
```

habilita a exibição
no console do SQL
gerado.

Valida ou exporta automaticamente o
schema ddl para o bando de dados
quando o SessionFactory é criado.

Values:

- validate
- update
- create
- create-drop

Dialeto que o
hibernate irá utilizar.

Mapeamento por Anotações

Física

Determinam o
Relacionamento
entre as classes e
o Banco de dados.

`@Entity`

`public class Carro {`

`@Id`

`private long id;`

`private String modelo;`

`private String placa;`

`@ManyToOne`

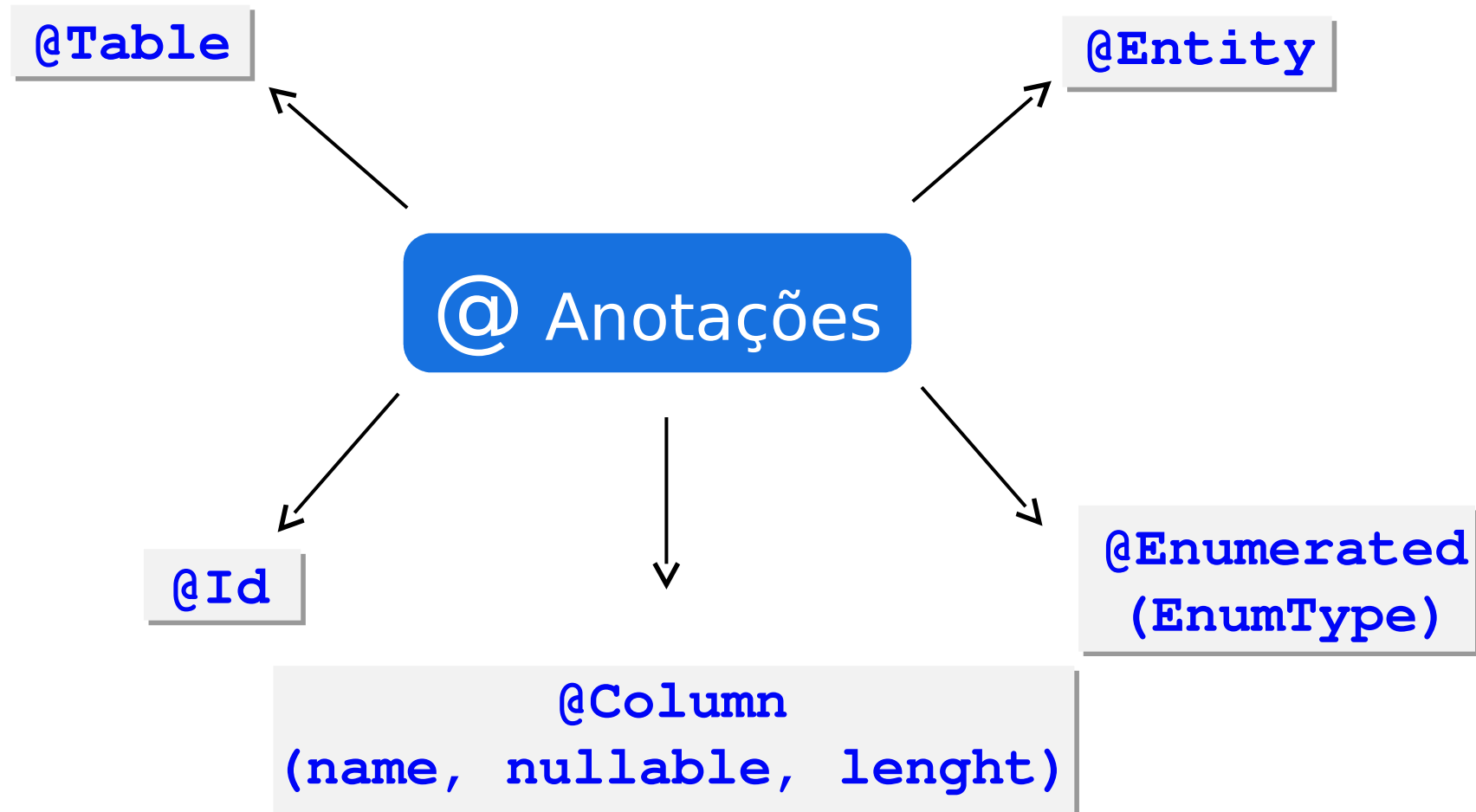
`@JoinColumn(name = "id_proprietario")`

`private Pessoa proprietario;`

Lógica

Definem a modelagem
da classe com relação
ao sistema.

Anotações



Anotações

```
public class Pessoa {  
    private long id;  
    private String nome;  
    private String registroGeral;  
    private Endereco endereco;
```

```
@Entity  
public class Pessoa {
```

```
    @Id  
    private long id;  
    private String nome;  
    @Column (name = "RG")  
    private String registroGeral;  
    @OneToOne  
    private Endereco endereco;
```

Indica que a classe é uma tabela no banco.

Indica que o campo será chave primária.

Anotação de relacionamento.

Indica que a coluna deve receber o nome "RG"

TABELA CORRESPONDENTE

Pessoa		
ID	NOME	RG

Geração Automática de Chaves Primárias IDENTITY

```
@Entity
public class Pessoa {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
```

O próprio banco de dados decide qual será o próximo ID.

SQLite, MySQL (AUTO_INCREMENT), SQL Server

Geração Automática de Chaves Primárias SEQUENCE

Indica que haverá uma
sequência no banco de dados

```
@Entity
@SequenceGenerator(name="seq_gen", initialValue=1, allocationSize=100)
public class Pessoa {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator="seq_gen")
    private long id;
```

Oracle, SQL Server, PostgreSQL
Padrão do PostgreSQL e Oracle

Indica que uma sequência salva
no banco de dados será usada.

Mantém salvo o próximo
valor a ser usado.

Geração Automática de Chaves Primárias TABLE

```
@Entity
@TableGenerator(name="tab_gen", initialValue=0, allocationSize=100)
public class Pessoa {

    @Id
    @GeneratedValue(strategy = GenerationType.TABLE, generator="tab_gen")
    private long id;
```

Utiliza uma tabela para armazenar o id atual de cada entidade. Mantém salvo o último valor que foi usado.

Única estratégia suportada em qualquer BD
Portabilidade

Geração Automática de Chaves Primárias AUTO

```
@Entity
public class Pessoa {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;
```

IDENTITY

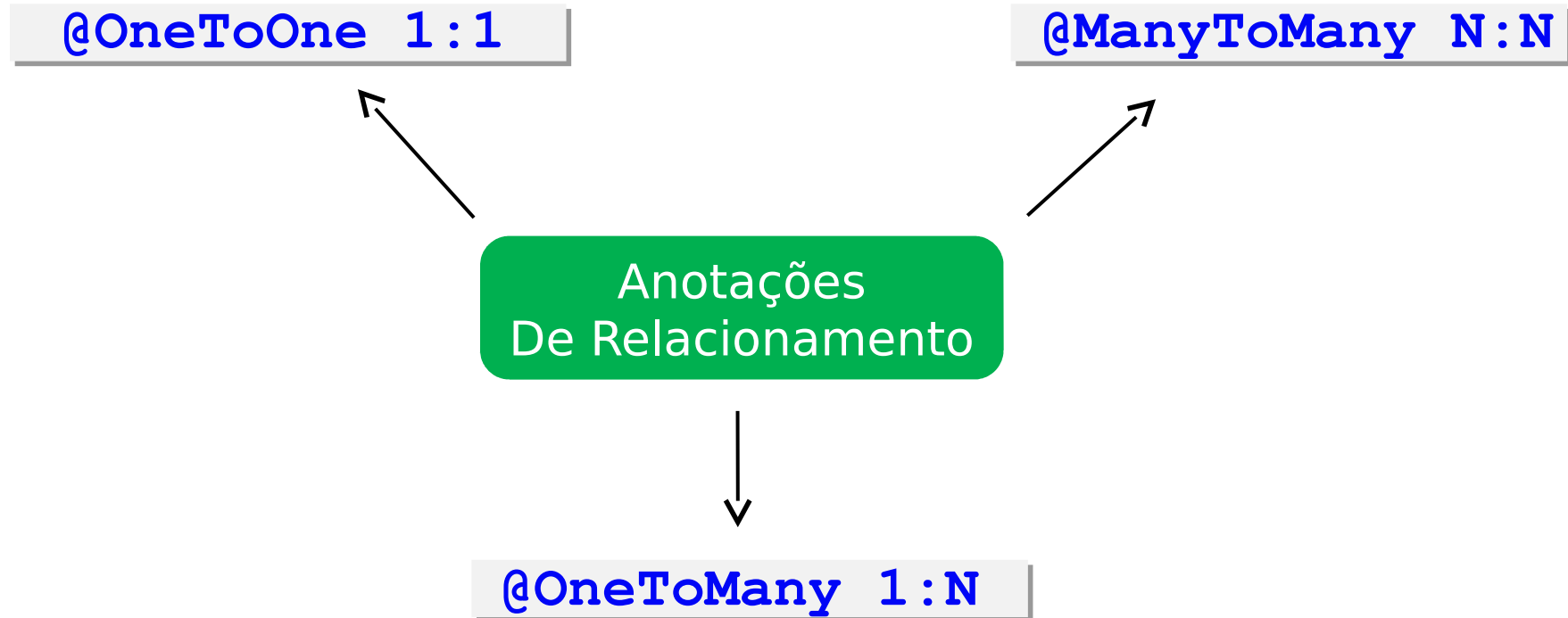
- MySQL, SQLite e MsSQL

Indica que a JPA deve selecionar uma estratégia de geração apropriada de acordo com o banco.

SEQUENCE

- Oracle e PostgreSQL

Anotações de Relacionamento



One To One Unidirecional

```
@Entity
public class Pessoa {

    @Id
    private long id;

    private String nome;

    @Column (name = "RG")
    private String registroGeral;

    @OneToOne
    private Endereco endereco;
```

```
@Entity
public class Endereco {

    @Id
    private long id;

    private String rua;

    private String cep;
```

TABELA CORRESPONDENTE



Ono To One Bidirecional

```
@Entity
public class Pessoa {

    @Id
    private long id;

    private String nome;

    @Column (name = "RG")
    private String registroGeral;

    @OneToOne
    private Endereco endereco;
```

```
@Entity
public class Endereco {

    @Id
    private long id;

    private String rua;

    private String cep;

    @OneToOne(mappedBy = "endereco")
    private Pessoa pessoa;
```

TABELA CORRESPONDENTE

Pessoa			
ID	NOME	RG	ID_ENDERECO

Endereco		
ID	RUA	CEP

One To Many Unidirecional

```
@Entity
public class Pessoa {

    @Id
    private long id;

    private String nome;

    @Column (name = "RG")
    private String registroGeral;

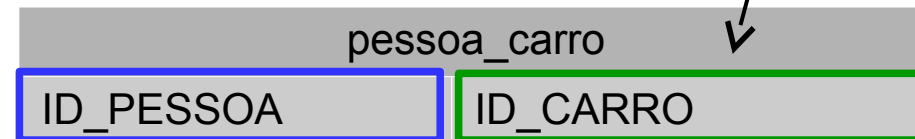
    @OneToMany
    private List<Carro> carros;
```

```
@Entity
public class Carro {

    @Id
    private long id;

    private String modelo;

    private String placa;
```



*Tabela adicional para armazenar as
chaves do relacionamento*

Pessoa		
ID	NOME	RG

Carro		
ID	MODELO	PLACA

One To Many ndo Join Column

```
@Entity
public class Pessoa {

    @Id
    private long id;

    private String nome;

    @Column (name = "RG")
    private String registroGeral;

    @OneToMany
    @JoinColumn(name = "proprietario")
    private List<Carro> carros;
```

```
@Entity
public class Carro {

    @Id
    private long id;

    private String modelo;

    private String placa;
```

Carro			
ID	MODELO	PLACA	PROPRIETARIO

Chave Estrangeira

Pessoa		
ID	NOME	RG

Chave Primária

Elimina a necessidade
de uma tabela adicional

One To Many Bidirecional

```
@Entity
public class Pessoa {

    @Id
    private long id;

    private String nome;

    @Column (name = "RG")
    private String registroGeral;
```

```
@OneToMany(mappedBy = "proprietario")
private List<Carro> carros;
```

```
@Entity
public class Carro {

    @Id
    private long id;

    private String modelo;

    private String placa;
```

```
@ManyToOne
@JoinColumn(name = "id_proprietario")
private Pessoa proprietario;
```

Pessoa		
ID	NOME	RG

Carro			
ID	MODELO	PLACA	ID_PROPRIETARIO

Many To Many

```
@Entity
public class Autor {

    @Id
    private long id;

    private String nome;

    @ManyToMany
    @JoinTable(name="autor_livro")
    private List<Livro> livros;
```

```
@Entity
public class Livro {

    @Id
    private long id;

    private String nome;

    @Column(name="DATA_PUBLICACAO")
    @Temporal(TemporalType.DATE)
    private Date ano;

    @ManyToMany(mappedBy="livros")
    private List<Autor> autores;
```

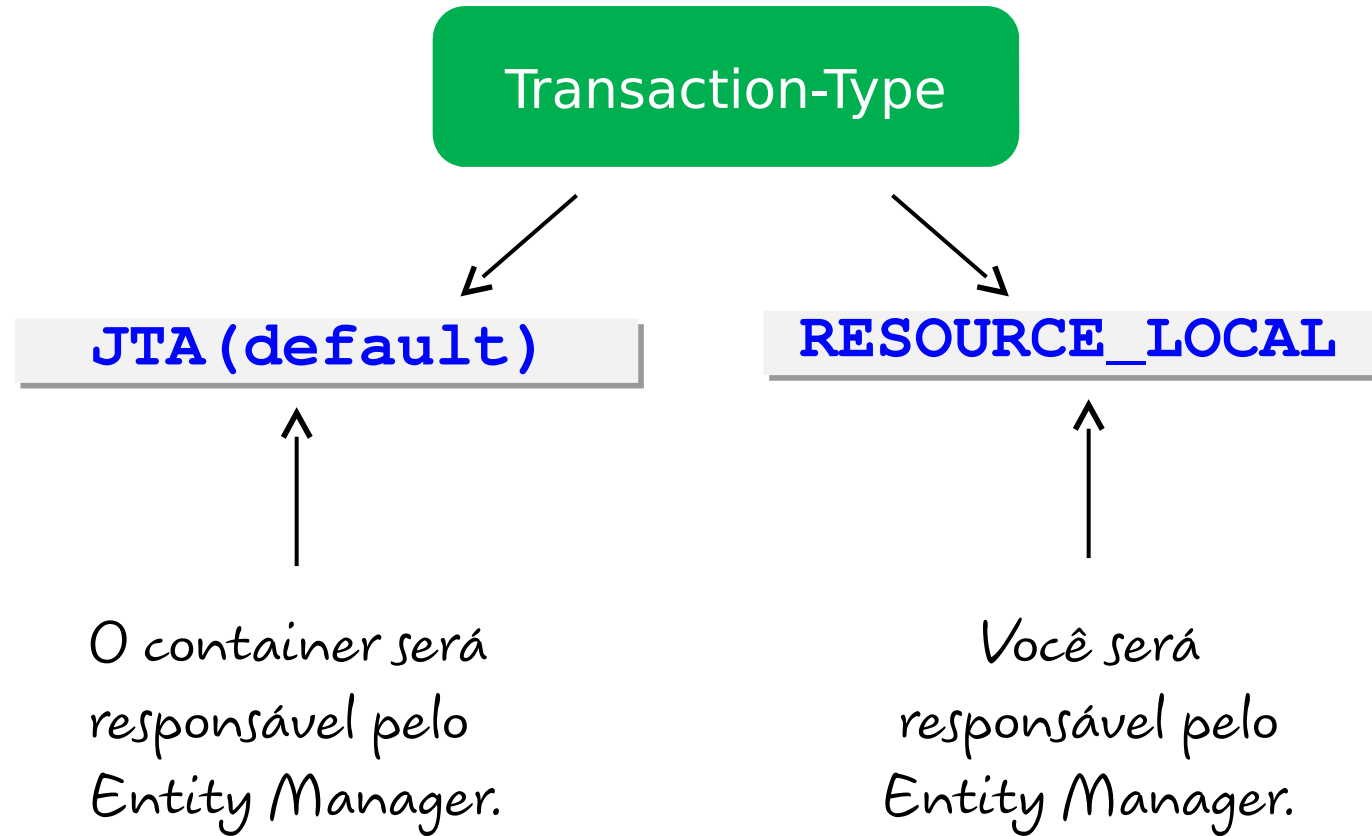
Autor	
ID	NOME

Livro		
ID	NOME	DATA_PUBLICACAO

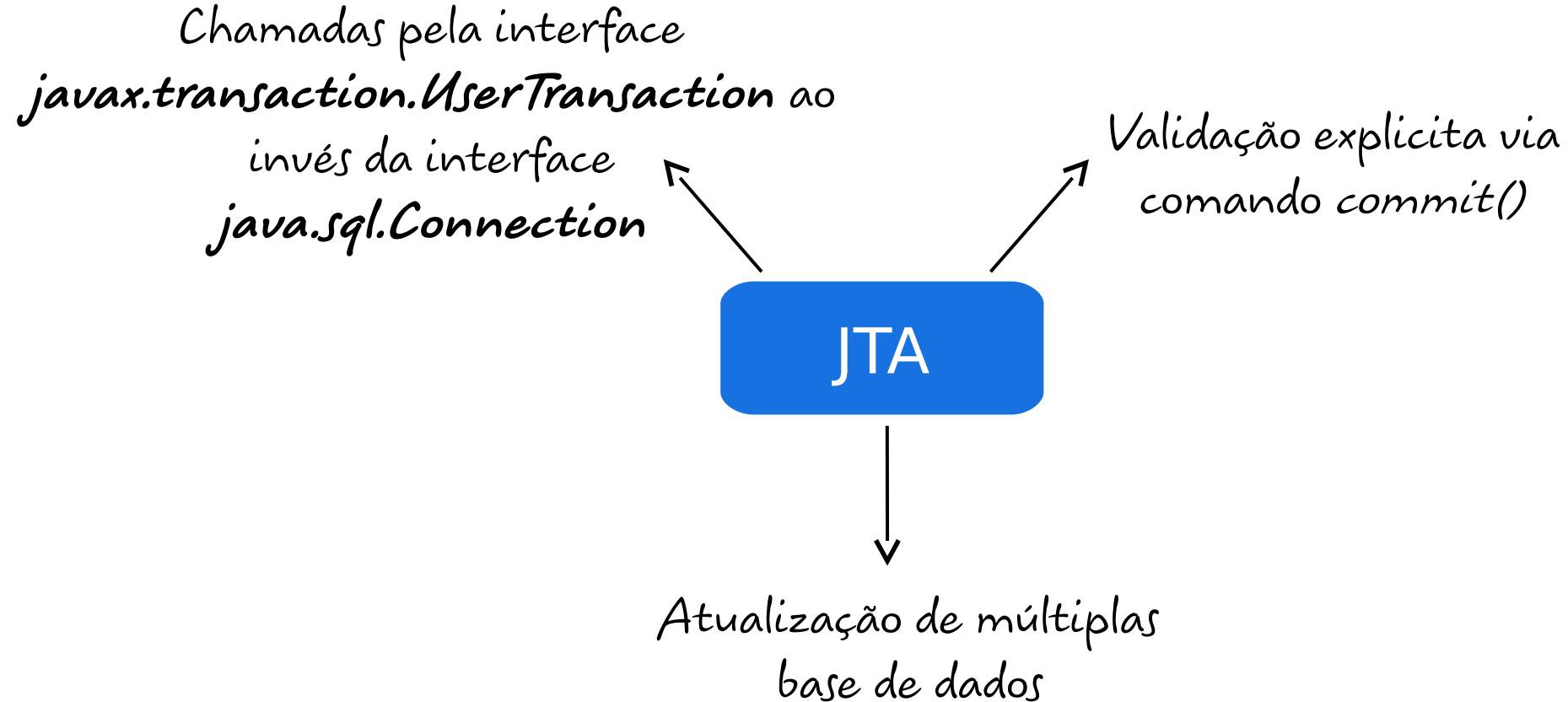
autor_livro	
ID_AUTOR	ID_LIVRO

União das tabelas com os IDs

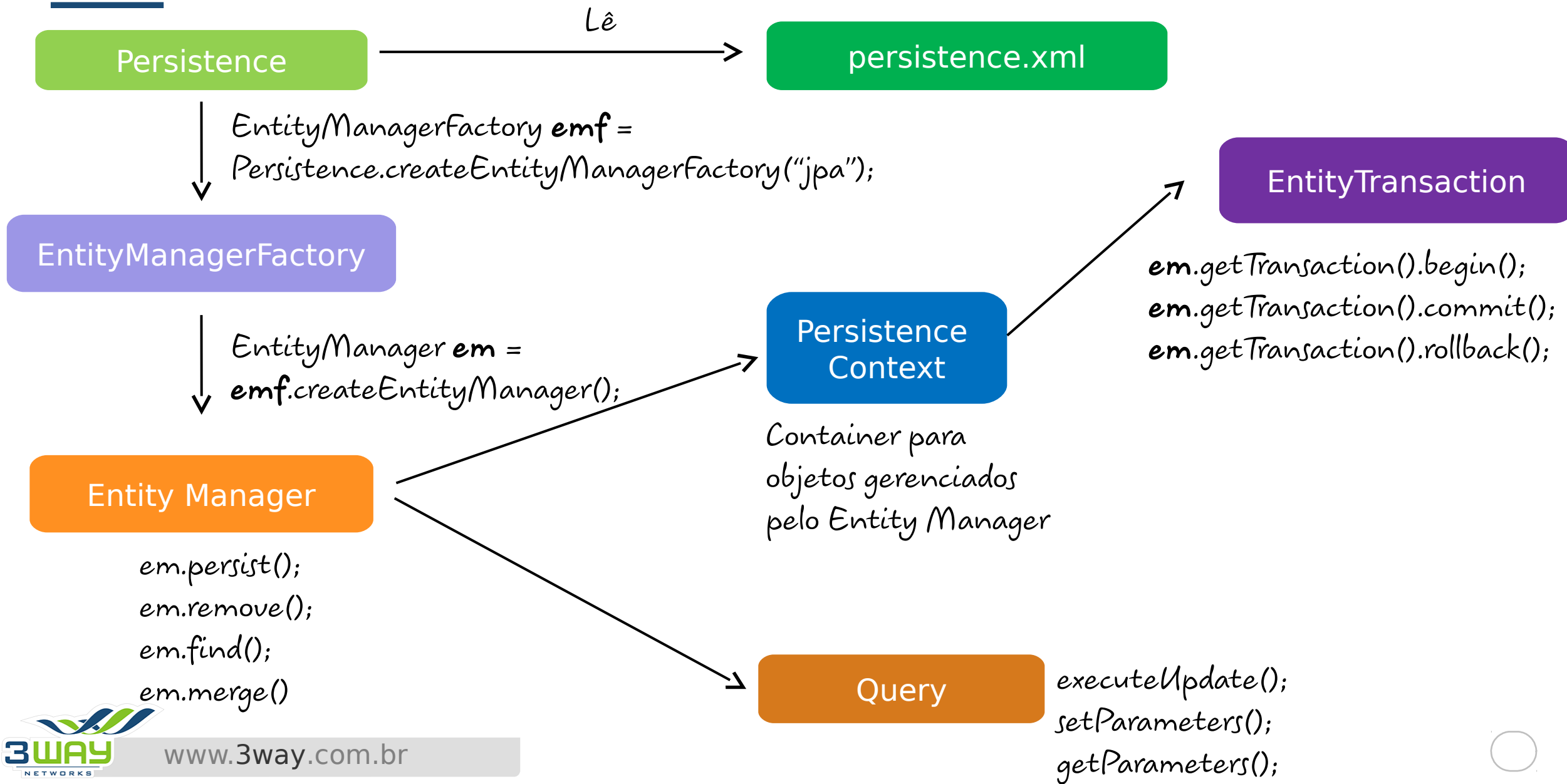
Tipos de Transação



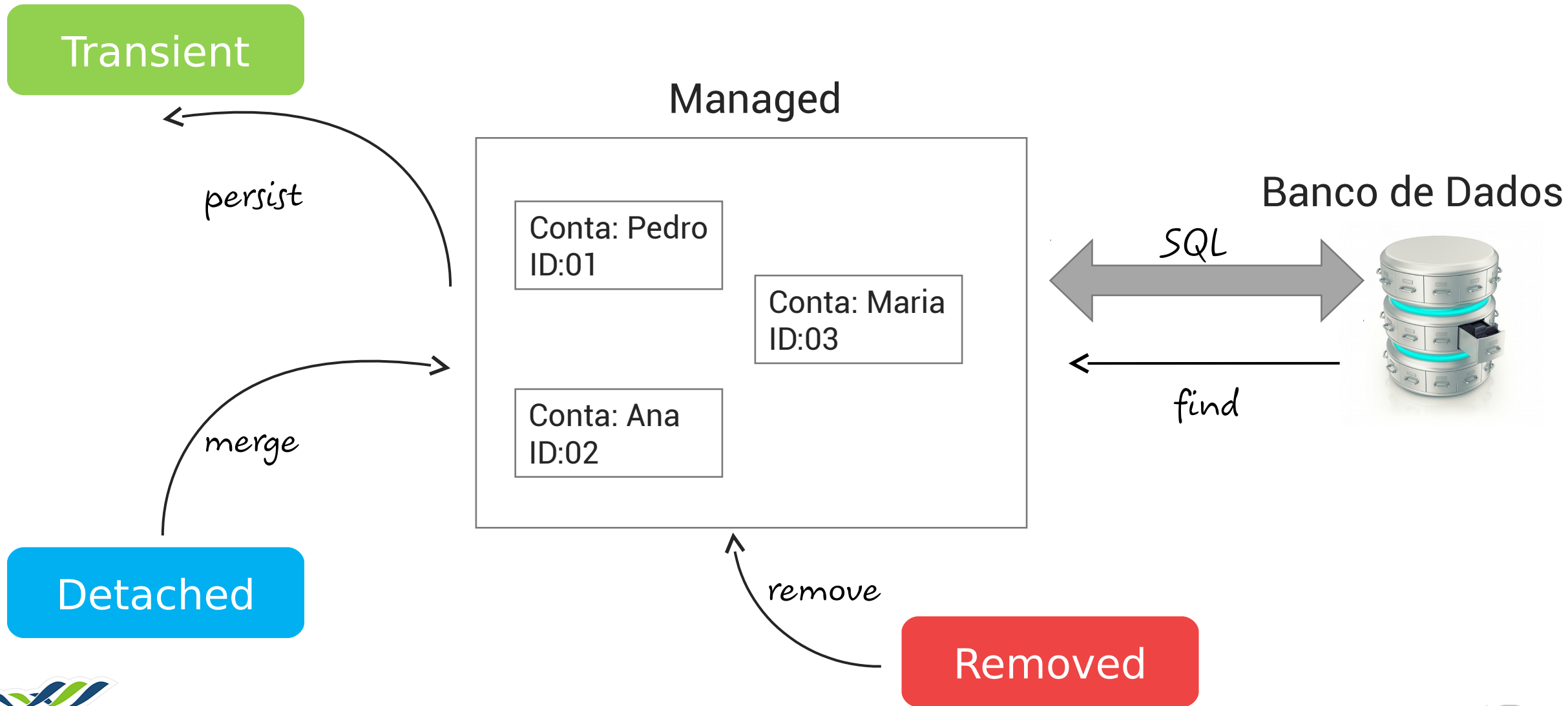
JTA



Entity Manager



Estados de uma Entidade



EntityManager Criar

Criação do Entity Manager

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpa");  
EntityManager em = emf.createEntityManager();
```

```
Pessoa pessoa = new Pessoa();  
pessoa.setNome("Joao");  
pessoa.setCPF("123456");
```

*Instanciação de um novo objeto.
Neste momento, o objeto está
estado Transient.*

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

*Persiste o novo objeto no banco de dados e ele passa
para o estado de Managed (gerenciado).*

EntityManager Recuperar

```
Pessoa pessoa = em.find(Pessoa.class, 1);
```

Tipo da entidade
a ser buscada

Busca feita pelo valor
da Primary Key

EntityManager Update

Objeto deixa o estado de Gerenciado e passa para o estado *Detached* (destacado)

Busca o objeto, deixando-a em estado *Managed*

```
Pessoa pessoa = em.find(Pessoa.class, 1);  
em.getTransaction().begin();  
em.detach(pessoa);  
  
pessoa.setNome("Maria Tereza");  
  
em.merge(pessoa);  
em.getTransaction().commit;
```

Alteração do nome

Objeto volta para o estado de Gerenciado e as alterações podem ser salvas no banco

EntityManager Deletar

Busca o objeto

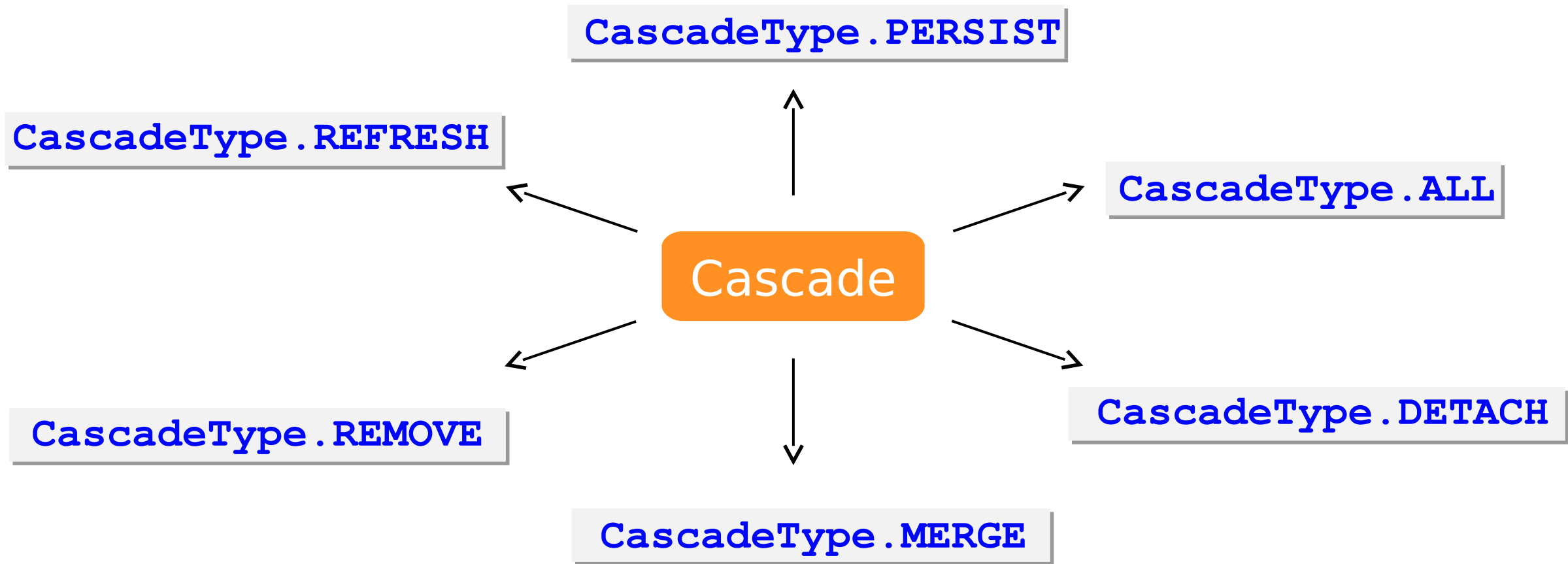


```
Pessoa pessoa = em.find(Pessoa.class, 1);  
  
em.getTransaction().begin();  
em.remove(pessoa);  
em.getTransaction().commit();
```



Remove a entidade. Entidade agora esta no estado *Removed*

Cascade



Cascade

```
@Entity  
public class Pessoa {
```

```
    @Id  
    private long id;
```

```
    private String nome;
```

```
    @Column (name = "RG")  
    private String registroGeral;
```

```
    @OneToOne  
    private Endereco endereco;
```

```
@Entity  
public class Pessoa {
```

```
    @Id  
    private long id;
```

```
    private String nome;
```

```
    @Column (name = "RG")  
    private String registroGeral;
```

```
    @OneToOne(cascade = CascadeType.PERSIST)  
    private Endereco endereco;
```

CascadeType deve ser passado dentro na
anotação de relacionamento

CascadeType.Persist faz com que o
endereço também seja persistido

```
Pessoa pessoa = new Pessoa();  
pessoa.setCPF("123456");  
pessoa.setNome("Maria");
```

```
Endereco endereco = new Endereco();  
endereco.setRua("Rua 2015");  
endereco.setCep("12345678");
```

```
pessoa.setEndereco(endereco);
```

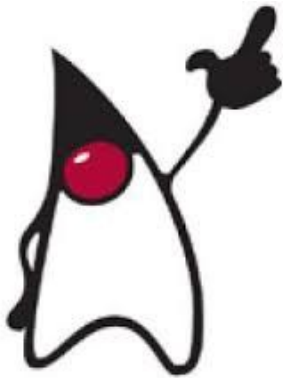
```
em.getTransaction().begin();  
em.persist(pessoa);  
em.getTransaction().commit();
```

OneToOne, OneToMany e ManyToMany

Orphan Removal

(orphanRemoval = true)

A entidade do lado inverso do lado dominante é deletada do banco de dados quando não há relacionamento com outra entidade dominante



FetchType

FetchType.LAZY

- As informações do atributo marcado não serão carregadas.

```
@Entity
public class Pessoa {

    @Id
    private long id;

    private String nome;

    @Column (name = "RG")
    private String registroGeral;

    @OneToMany(mappedBy = "proprietario", fetch = FetchType.LAZY)
    private List<Carro> carros;
```

FetchType.EAGER

- As informações do atributo marcado serão carregadas.

```
@Entity
public class Pessoa {

    @Id
    private long id;

    private String nome;

    @Column (name = "RG")
    private String registroGeral;

    @OneToOne(cascade = CascadeType.PERSIST, fetch = FetchType.EAGER)
    private Endereco endereco;
```

JPQL Java Persistence Query Language

Recupera classes e objetos

Portabilidade entre bancos

JPQL
Java Persistence Query Language

JPQL não suporta o método INSERT!

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpa");
EntityManager em = emf.createEntityManager();

Pessoa pessoa = new Pessoa();
pessoa.setNome("Joao");
pessoa.setCPF("123456");

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


JPQL Recuperar

Criação do Entity Manager

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpa");  
EntityManager em = emf.createEntityManager();  
  
Query query = em.createQuery("SELECT p FROM Pessoa p", Pessoa.class);  
List<Pessoa> resultList = query.getResultList();
```

Criação da Query

Retorna o resultado em uma
lista de objetos do tipo indicado

Tipo do objeto a
ser retornado

JPQL Update

Criação do Entity Manager

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpa");
EntityManager em = emf.createEntityManager();

Query query = em.createQuery(
    "UPDATE Pessoa SET nome = \"Pedro\" " +
    "WHERE nome = :nome ");

query.setParameter("nome", "Joao");
query.executeUpdate();
```

Criação da Query

Parâmetro nome

JPQL Deletar

Criação do Entity Manager

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpa");
EntityManager em = emf.createEntityManager();

Query query = em.createQuery("DELETE FROM Pessoa p WHERE p.id = :id ");
query.setParameter("id", "1");
query.executeUpdate();
```

Parâmetro id

Criação da Query

Named Queries

Anotacao @NamedQuery na entidade

```
@Entity
@NamedQuery (name = "Cliente.BUSCA_POR_CPF",
            query = "SELECT Cliente WHERE cpf = :cpf")
public class Cliente {
    public String final BUSCA_POR_CPF = "Cliente.BUSCA_POR_CPF";
    ...
}
```

Nome da query

Query parametrizada

Boa prática
Constante com
o nome da query

Nome da query

```
TypedQuery<Cliente> query = em.createNamedQuery(Cliente.BUSCA_POR_CPF, Cliente.class);
query.setParameter("cpf", "123456");

Cliente cliente = query.getSingleResult();
```

Tipo do objeto a
ser retornado

Native Query

Criação do entity manager

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpa");
EntityManager em = emf.createEntityManager();

String consulta = "SELECT * FROM Pessoa WHERE id = :id";

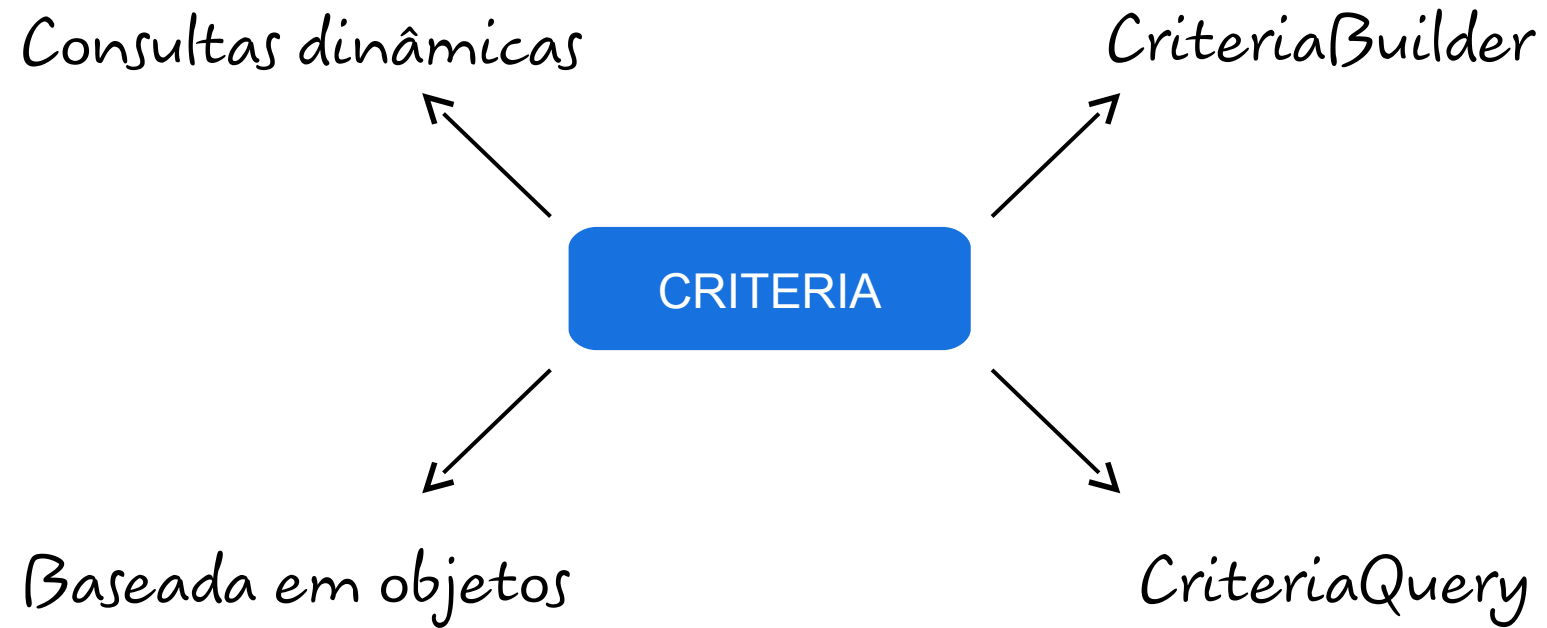
Query query = em.createNativeQuery(consulta, Pessoa.class);
query.setParameter("id", 1);

Pessoa pessoa = query.getSingleResult();
```

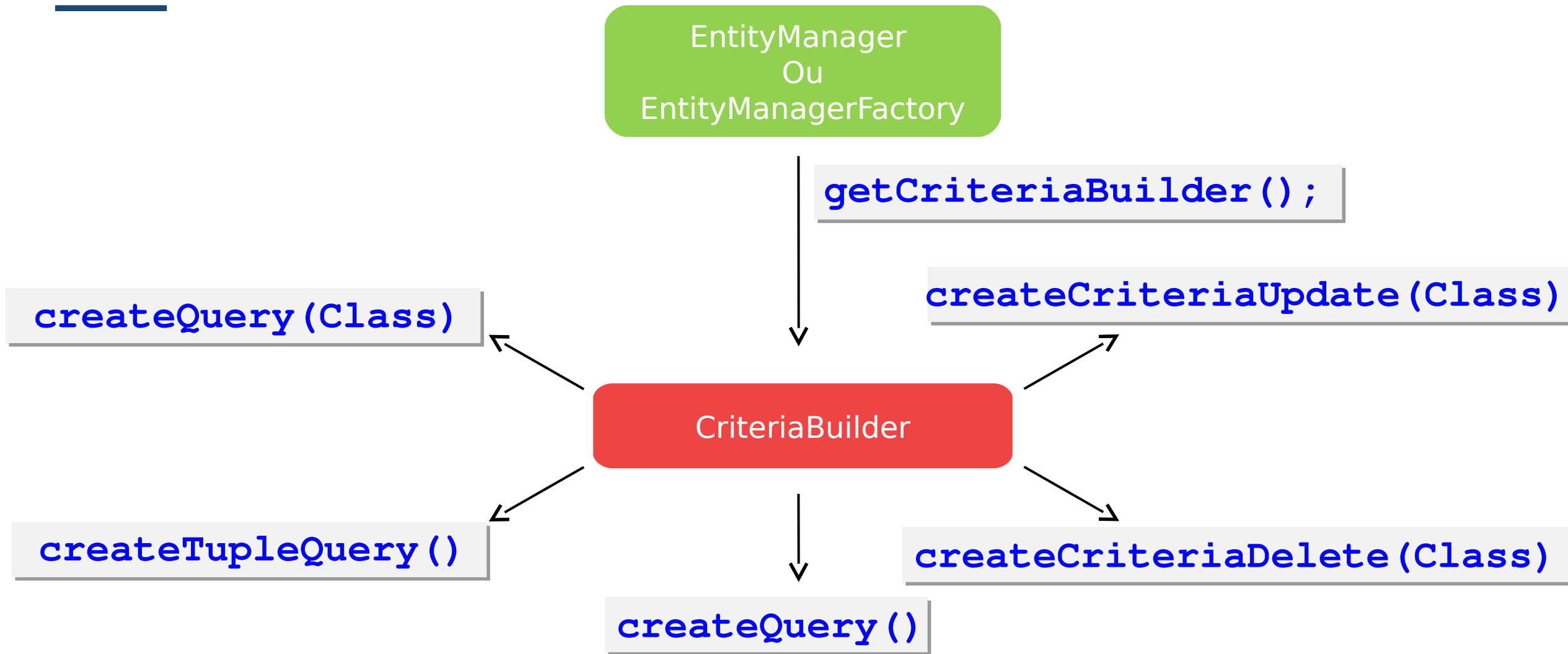
Criação da Query
Nativa

A query deve estar na linguagem
nativa do banco que está sendo
utilizado

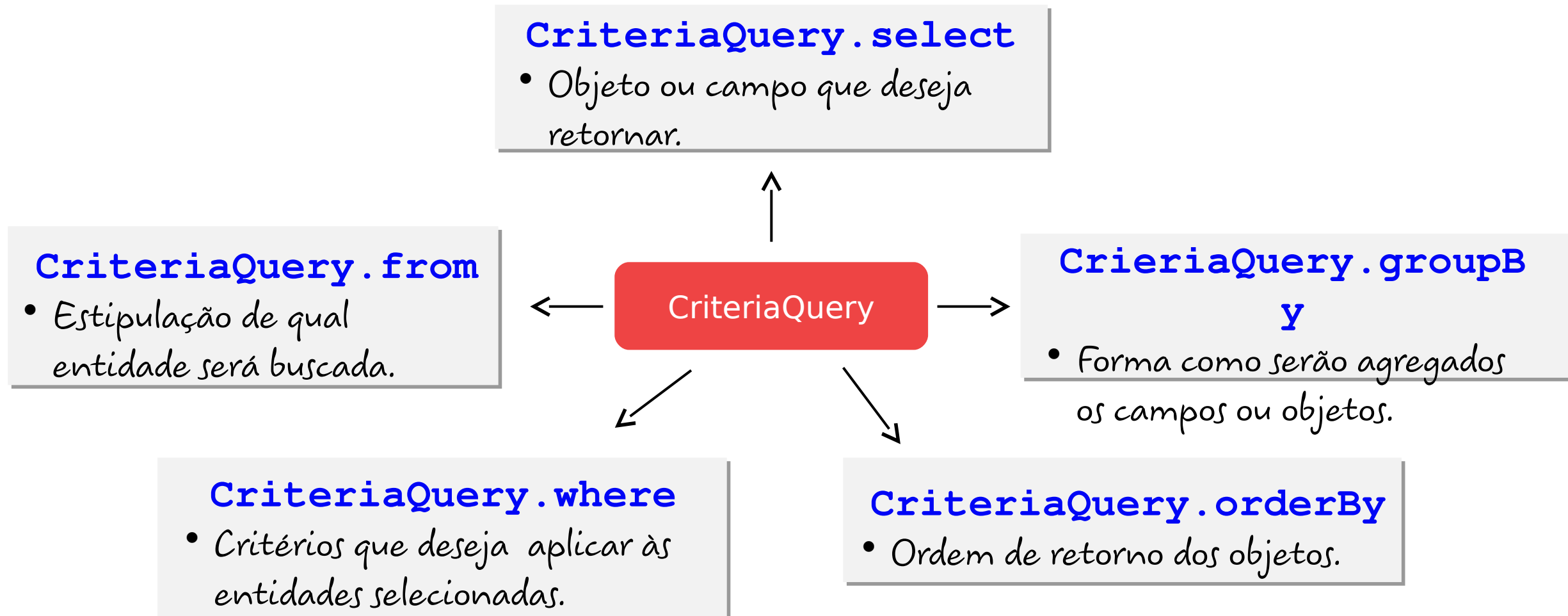
CRITERIA



Criteria



CriteriaQuery



CriteriaQuery

Entidade raiz da busca

Deverá retornar uma entidade desta classe

```
CriteriaQuery criteriaQuery = criteriaBuilder.createQuery();  
Root pessoa = criteriaQuery.from(Pessoa.class);  
criteriaQuery.where(criteriaBuilder.equal(pessoa.get("nome"), "Joao"));  
Query query = entityManager.createQuery(criteriaQuery);  
List<Pessoa> result = query.getResultList();
```

Cláusula **WHERE**
da consulta.

Resultado será uma
lista do tipo *Pessoa*

CriteriaQuery

Entidade raiz da busca

Deverá retornar uma entidade desta classe

```
CriteriaQuery criteriaQuery = criteriaBuilder.createQuery();
Root pessoa = criteriaQuery.from(Pessoa.class);
criteriaQuery.where(criteriaBuilder.equal(pessoa.get("id"),
                                         criteriaBuilder.parameter(Long.class, "id")));
Query query = entityManager.createQuery(criteriaQuery);
query.setParameter("id", 5);
Pessoa result2 = (Pessoa)query.getSingleResult();
```

Cláusula WHERE da consulta
com parâmetro id

Único resultado do tipo
Pessoa

Tipo do parâmetro
a ser recebido

Novidades na JPA 2.1

Alvos	Função
Attribute converter	Conversores de código customizados entre o banco de dados e objetos.
Criteria UPDATE/DELETE	updates e deletes em massa através da Criteria API.
Schema generation	Geração automática de tabelas, indexes e schemas.
Named Stored Procedures Query	Permite definir queries para procedimentos armazenados no banco de dados.
Constructor Result Mapping	Suporte para SQLResultSetMapping.
Dynamic Named Queries	Criação de queries em tempo de execução.
Melhorias na JPQL	Subqueries aritméticas, funções genéricas do banco de dados, cláusula JOIN ON e opção TREAT.
Entity Graph	Permite fetching ou merging parcial ou específico de objetos..
Suporte de CDI no Entity Listener	Permite utilizar CDI para injeção de beans em <i>EntityListeners</i> .