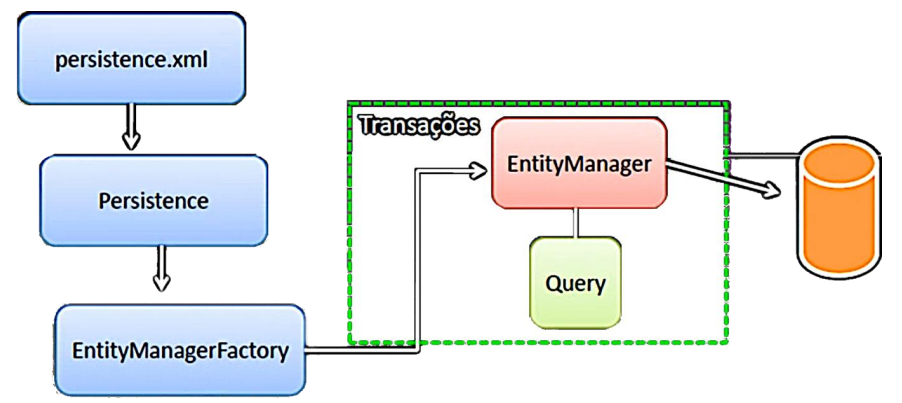
**JPA e Hibernate com Maven**

JPA é uma especificação (JSR-317) para mapeamento objeto-relacional enquanto que Hibernate é uma implementação desta especificação. *Java Persistence API* (ou simplesmente JPA) é uma API padrão da linguagem Java que descreve uma interface comum para frameworks de persistência de dados. A JPA define um meio de mapeamento objeto-relacional para objetos Java simples e comuns (Plain Old Java Objects - POJOs), denominados *beans de entidade*.

Hibernate surgiu primeiro e influenciou grandemente a forma final da JPA. O Hibernate é um framework para o mapeamento objeto-relacional escrito na linguagem Java. O objetivo do Hibernate é diminuir a complexidade entre os programas Java, baseado no modelo orientado a objeto, que precisam trabalhar com um banco de dados do modelo relacional. Sua principal característica é a transformação das classes em Java para tabelas de dados.

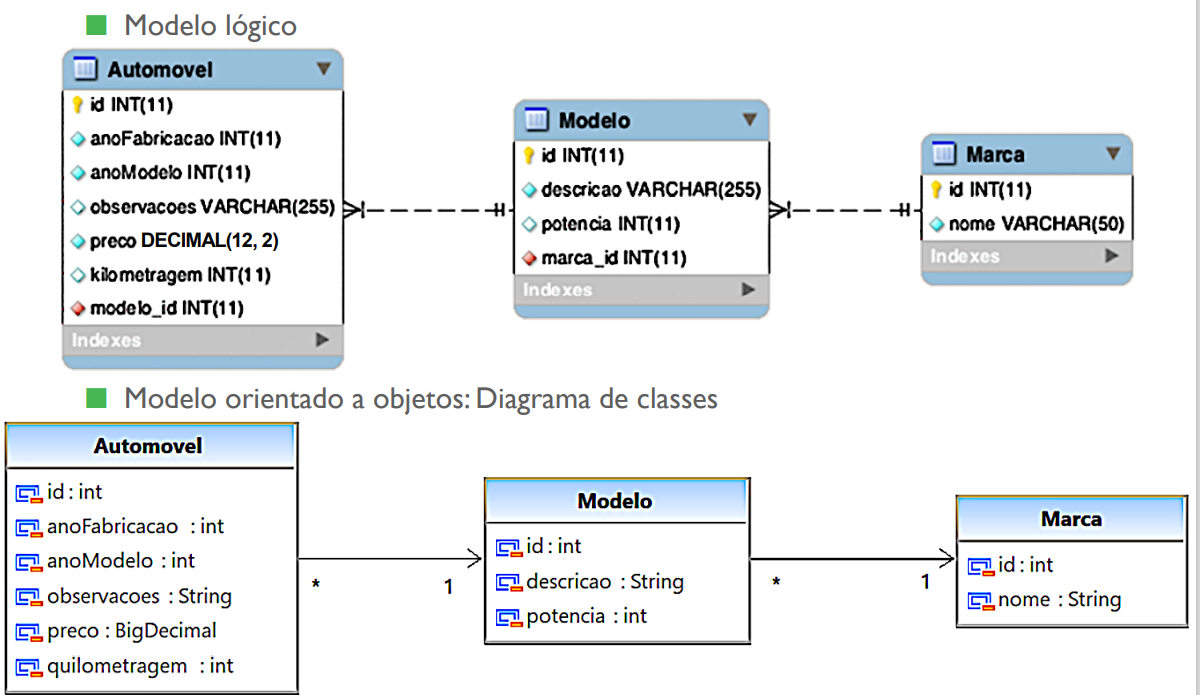
Arquitetura



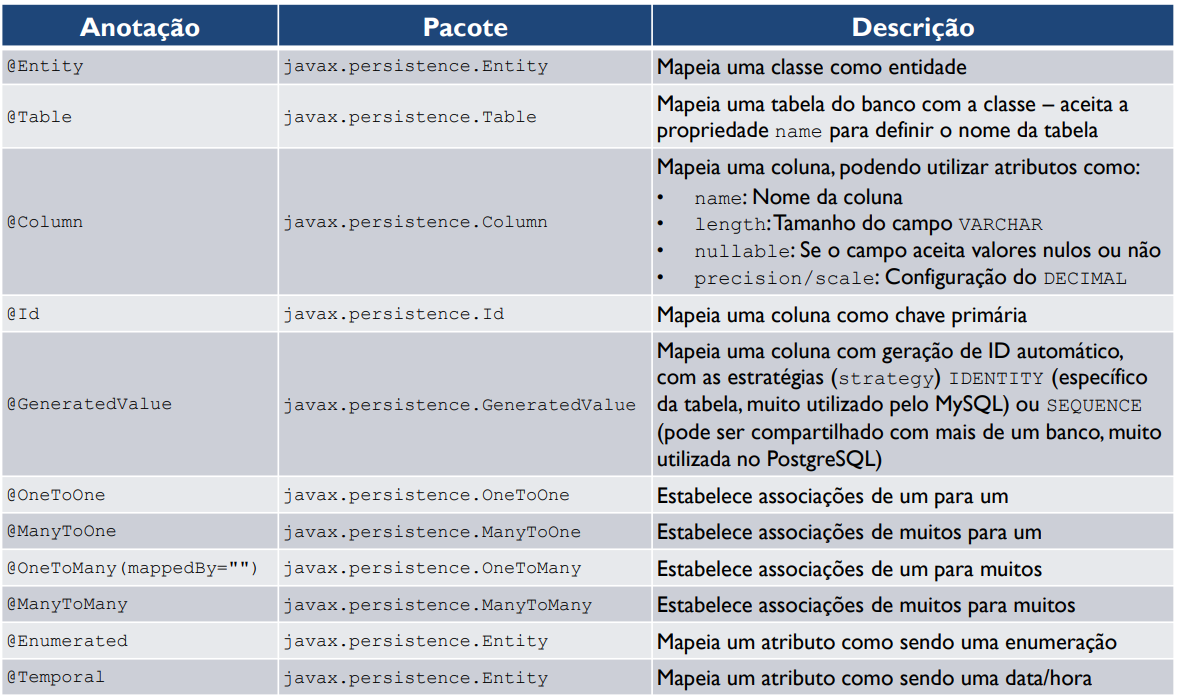
Classes importantes



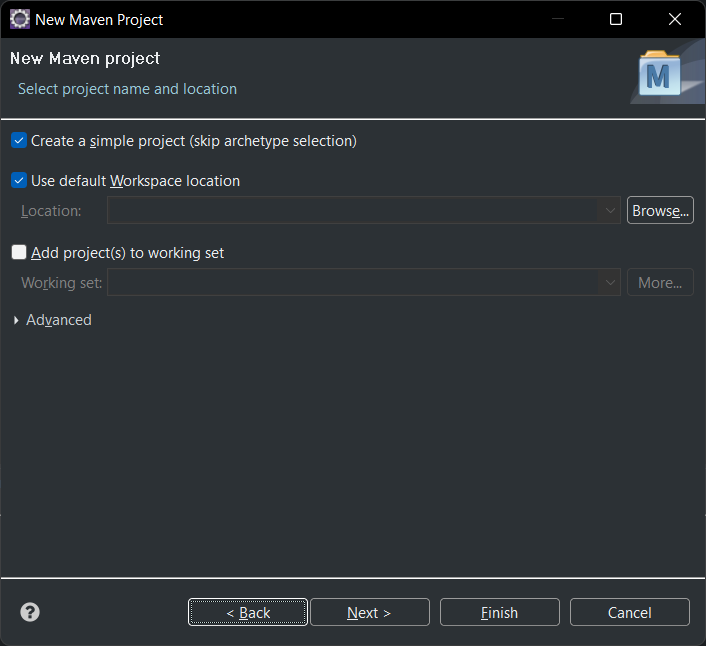
Modelo lógico x Modelo orientado a objetos

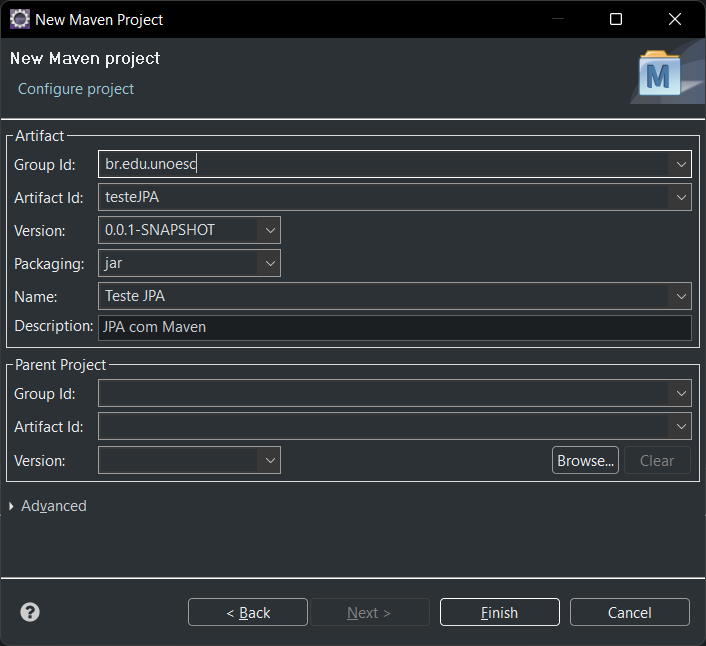


Anotações



Vamos começar criando um projeto Maven, conforme a sequência de passos descrita a seguir:





No arquivo pom.xml, incluir properties e dependencies para a versão 17 do Java, para o Hibernate e para o PostgreSQL.

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<java.version>17</java.version>

<maven.compiler.source>${java.version}</maven.compiler.source>

<maven.compiler.target>${java.version}</maven.compiler.target>

</properties>

<dependencies>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>6.1.2.Final</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.postgresql/postgresql -->

<dependency>

<groupId>org.postgresql</groupId>

<artifactId>postgresql</artifactId>

<version>42.5.0</version>

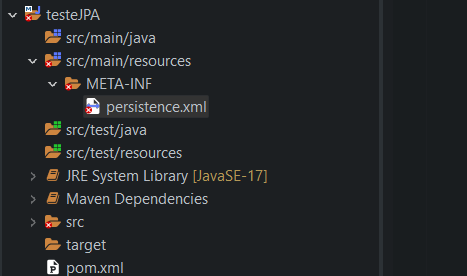
</dependency>

</dependencies>

Atualizar o projeto e conferir se a versão do Java passou a ser a versão 17 e se as demais dependências do Maven foram carregadas.

Criar uma nova **pasta (folder)** chamada META-INF no package src/main/resources.

Criar um **arquivo(file)** chamado persistence.xml dentro desta pasta.



Informar o seguinte conteúdo para o arquivo de persistência:

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<persistence version="3.0"

xmlns="https://jakarta.ee/xml/ns/persistence"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence https://jakarta.ee/xml/ns/persistence\_3\_0.xsd">

<!-- unidade de persistencia com o nome unoesc -->

<persistence-unit name="jpapersistencia" transaction-type="RESOURCE\_LOCAL">

<provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

<!-- precisa mudar aqui -->

<!--

<class>br.edu.unoesc.testejpa.modelo.Pessoa</class>

<class>br.edu.unoesc.testejpa.modelo.Departamento</class>

<class>br.edu.unoesc.testejpa.modelo.Cargo</class>

-->

<properties>

<property name="jakarta.persistence.jdbc.driver" value="org.postgresql.Driver" />

<property name="jakarta.persistence.jdbc.url" value="jdbc:postgresql://localhost:5432/jpapersistencia"/>

<property name="jakarta.persistence.jdbc.user" value="postgres"/>

<property name="jakarta.persistence.jdbc.password" value="postgres"/>

<!-- Configurações específicas do Hibernate -->

<property name="hibernate.dialect"

value="org.hibernate.dialect.PostgreSQLDialect" />

<property name="hibernate.hbm2ddl.auto" value="create" />

<property name="hibernate.show\_sql" value="true" />

<property name="hibernate.format\_sql" value="true" />

</properties>

</persistence-unit>

</persistence>

Criar a classe Pessoa no package br.edu.unoesc.testejpa.model.

- Usar a anotação **@Entity**

- Definir o atributo **id** como um campo de identificação (chave) e cujo valor é automaticamente gerado pelo banco de dados (anotação **@GeneratedValue**).

- Definir o atributo data\_nascimento como um campo Date (anotação **@Temporal**) - Projetos menos atuais costumam usar DATE ou Calendar, para projetos novos pode-se usar LocalDate ou Instant .

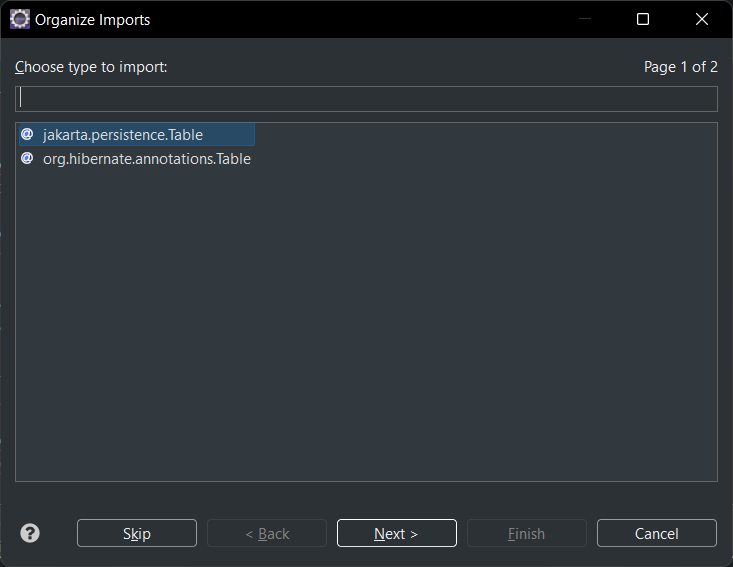
- Definir do tamanho total e precisão para o campo decimal do salário.

- Declarar da interface **Serializable** e criação do **UID**

- Criar os construtores, os getters e setters e o método toString()

Com o atalho Ctrl+Shift+O, atualizar os imports – conferir se as classes corretas foram importadas, como no exemplo a seguir:

Confira os imports (**Jakarta**)



A classe Pessoa deve ser construída assim

package br.edu.unoesc.testejpa.model;

import java.io.Serializable;

import java.math.BigDecimal;

import java.sql.Date;

import jakarta.persistence.Column;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

import jakarta.persistence.Temporal;

import jakarta.persistence.TemporalType;

@Entity

@Table(name = "pessoas")

public class Pessoa implements Serializable {

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

@Column(length = 50, nullable = false)

private String nome;

@Temporal(TemporalType.DATE)

@Column(name = "data\_nascimento", nullable = false)

private java.sql.Date dataNascimento;

@Column(precision = 12, scale = 2, nullable = false)

private BigDecimal salario;

// construtores

// getters / setters

// toString

}

Criar a classe JPAUtil no package br.edu.unoesc.testejpa.util assim:

package br.edu.unoesc.testejpa.util;

import jakarta.persistence.EntityManager;

import jakarta.persistence.EntityManagerFactory;

import jakarta.persistence.Persistence;

public class JPAUtil {

private static final EntityManagerFactory EMF = Persistence.createEntityManagerFactory("jpapersistencia");

private JPAUtil() {

}

public static EntityManager getEntityManager() {

return EMF.createEntityManager();

}

public static void closeEntityManagerFactory() {

EMF.close();

}

}

Construir uma classe para testar a conexão - classe App no package br.edu.unoesc.testejpa.app assim:

package br.edu.unoesc.testejpa.app;

import br.edu.unoesc.testejpa.util.JPAUtil;

import jakarta.persistence.EntityManager;

public class App {

private static EntityManager em;

public static void main(String[] args) {

App app = new App();

em = JPAUtil.getEntityManager();

em.close();

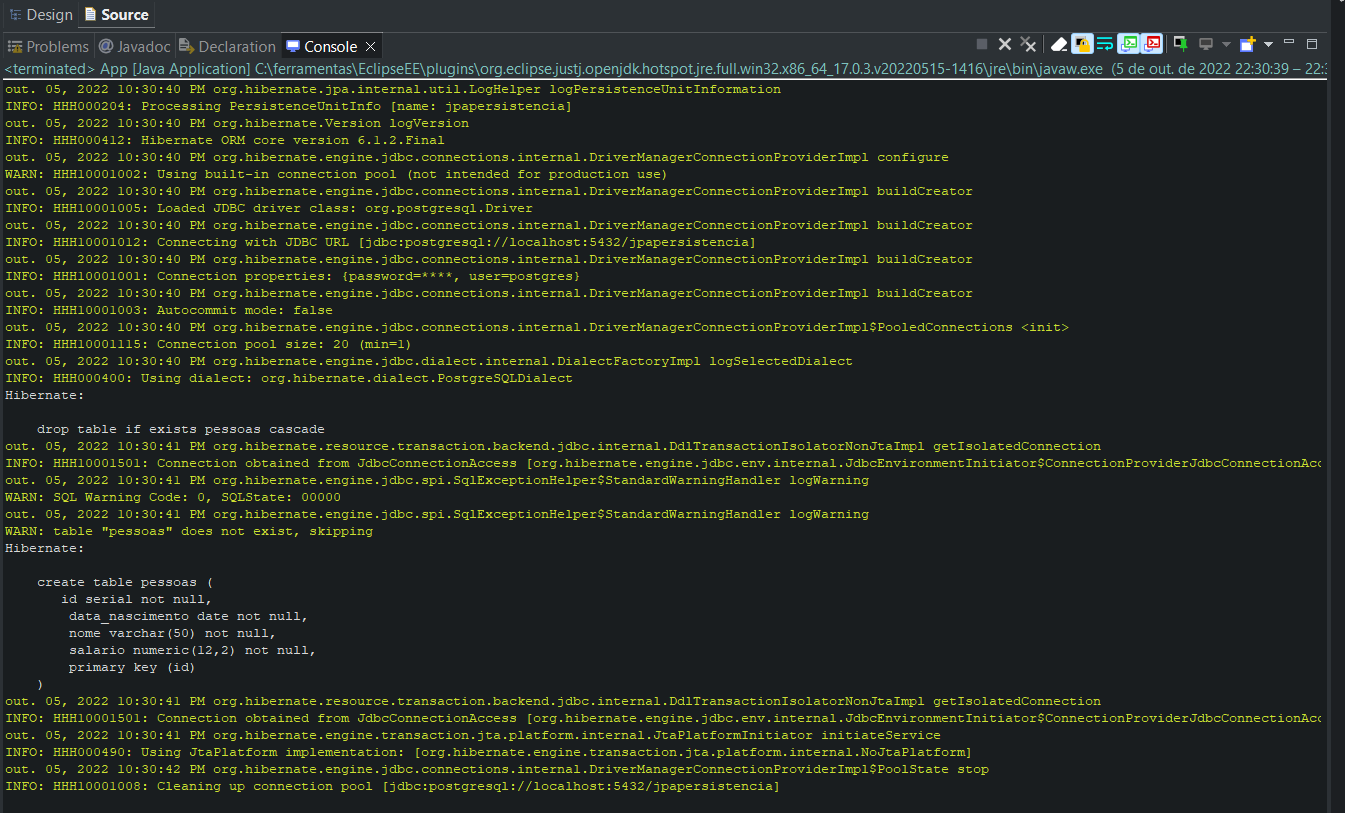
JPAUtil.closeEntityManagerFactory();

}

}

Criar o banco de dados com o PgAdmin chamado jpapersistencia

Rodar a aplicação que deve, nesse momento, mostrar algo assim:



Conferir no PgAdmin a tabela pessoas que foi criada.

Vamos agora construir uma nova classe App1, e o método adiciona para persistir dois registros de pessoa assim:

package br.edu.unoesc.testejpa.app;

import java.math.BigDecimal;

import br.edu.unoesc.testejpa.model.Pessoa;

import br.edu.unoesc.testejpa.util.JPAUtil;

import jakarta.persistence.EntityManager;

public class App1 {

private static EntityManager em;

private void adiciona() {

Pessoa p1 = new Pessoa("Otilia", java.sql.Date.valueOf("2000-04-01"), new BigDecimal("10000.0"));

Pessoa p2 = new Pessoa("Maria", java.sql.Date.valueOf("2001-05-02"), new BigDecimal("20000.0"));

// inicia a transação, persiste os objetos e faz commit

em.getTransaction().begin();

em.persist(p1);

em.persist(p2);

em.getTransaction().commit();

}

public static void main(String[] args) {

App1 app = new App1();

em = JPAUtil.getEntityManager();

app.adiciona();

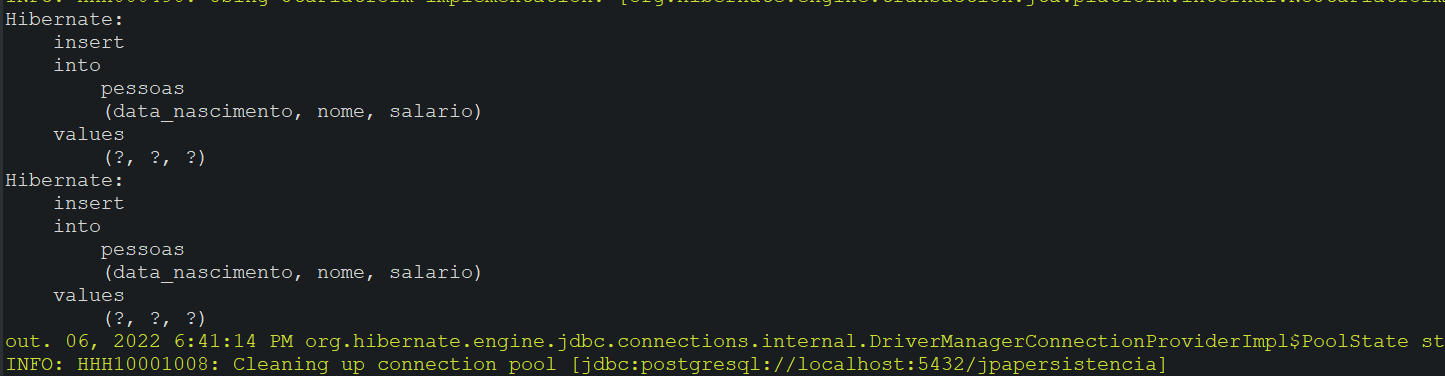
em.close();

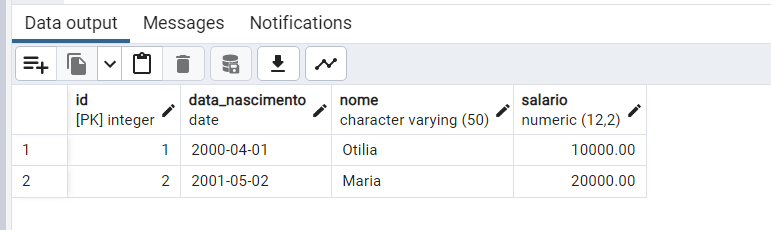
JPAUtil.closeEntityManagerFactory();

}

}

Ao executar deveremos ter 2 registros no banco de dados e um resultado assim:





Vamos construir outros métodos (classe App2 – incluir também o método adiciona) assim:

private void adiciona() { // igual ao descrito antes

...

}

private List<Pessoa> **buscarPorNome**(String nome) {

String jpql = "SELECT p FROM Pessoa p WHERE p.nome LIKE: nome";

return em.createQuery(jpql, Pessoa.class).setParameter("nome", "%" + nome + "%").getResultList();

}

private List<Pessoa> **buscarTodos**() {

String jpql = "SELECT p FROM Pessoa p";

return em.createQuery(jpql, Pessoa.class).getResultList();

}

private Pessoa **buscarPorId**(Integer id) {

return em.find(Pessoa.class, id);

}

private void **altera**() {

Pessoa pessoa = this.buscarPorId(1);

pessoa.setNome("Outra Otilia");

pessoa.setDataNascimento(java.sql.Date.valueOf("2002-05-05"));

pessoa.setSalario(new BigDecimal("12345.67"));

em.getTransaction().begin();

em.getTransaction().commit();

}

private void **exclui**() {

Pessoa pessoa = this.buscarPorId(2);

em.getTransaction().begin();

em.remove(pessoa);

em.getTransaction().commit();

}

Vamos agora construir o método main para testar esses métodos.

package br.edu.unoesc.testejpa.app;

import java.math.BigDecimal;

import java.util.List;

import br.edu.unoesc.testejpa.model.Pessoa;

import br.edu.unoesc.testejpa.util.JPAUtil;

import jakarta.persistence.EntityManager;

public class App2 {

private static EntityManager em;

public static void main(String[] args) {

App2 app = new App2();

em = JPAUtil.getEntityManager();

//Adicionar pessoas

app.adiciona();

// Alterar dados

app.altera();

//Dados da pessoa 1

System.out.println("Dados da pessoa 1");

System.out.println(app.buscarPorId(1).getNome());

//Dados da pessoa 10

System.out.println("Dados da pessoa 10");

Pessoa p = app.buscarPorId(10);

if(p==null) {

System.out.println("Pessoa inesistente!");

}else {

System.out.println(p.getNome());

}

//Todos os dados cadastrados

System.out.println("Todos os cadastrados");

List<Pessoa>lista = app.buscarTodos();

for(Pessoa pessoa: lista) {

System.out.println(pessoa.getNome());

}

//ou

//app.buscarTodos().forEach(System.out::println);

//Buscar por nome

System.out.println("Buscar por nome Maria");

lista = app.buscarPorNome("Maria");

for(Pessoa pessoa: lista) {

System.out.println(pessoa.getNome());

}

//Excluir

app.exclui();

em.close();

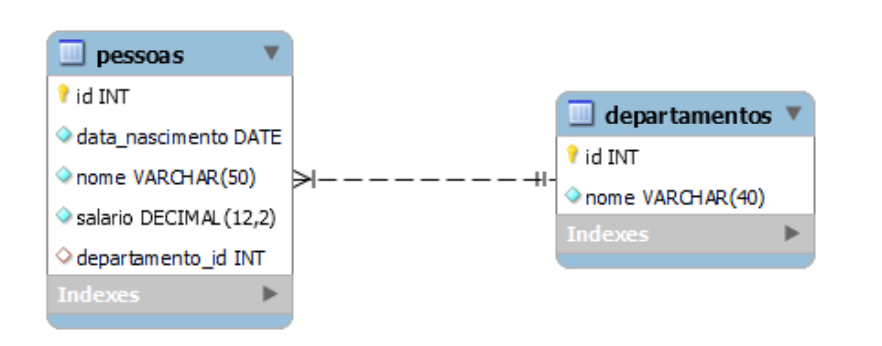
JPAUtil.closeEntityManagerFactory();

}

...

}

Vamos ampliar o projeto para incluir um Departamento ligado a cada Pessoa, assim:



Vemos que:

- Uma pessoa pertence a 1 e somente 1 departamento mas várias pessoas diferentes podem estar vinculadas a um mesmo departamento (𝑛: 1, ou muitos para um).

- Um departamento pode ter 0 ou mais pessoas vinculadas a ele (1: 𝑛, ou um para muitos).

Começaremos construindo a classe Departamento no package Model assim:

package br.edu.unoesc.testejpa.model;

import java.io.Serializable;

import jakarta.persistence.Column;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

@Entity

@Table(name = "departamentos")

public class Departamento implements Serializable{

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

@Column(length = 40, nullable = false)

private String nome;

// construtor completo e vazio

//getters/setters

//toString

}

Agora vamos fazer as ligações entre Pessoa e Departamento modificando a primeira classe assim:

package br.edu.unoesc.testejpa.model;

import java.io.Serializable;

import java.math.BigDecimal;

import java.sql.Date;

import jakarta.persistence.Column;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

**import jakarta.persistence.ManyToOne;**

import jakarta.persistence.Table;

import jakarta.persistence.Temporal;

import jakarta.persistence.TemporalType;

@Entity

@Table(name = "pessoas")

public class Pessoa implements Serializable {

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

@Column(length = 50, nullable = false)

private String nome;

@Temporal(TemporalType.DATE)

@Column(name = "data\_nascimento", nullable = false)

private java.sql.Date dataNascimento;

@Column(precision = 12, scale = 2, nullable = false)

private BigDecimal salario;

**@ManyToOne**

**Departamento departamento;**

**//criar um novo construtor completo – manter o existente**

**// atualizar gettes/setters**

**// atualizar toString**

}

Vamos construir um novo app para testar os ajustes:

package br.edu.unoesc.testejpa.app;

import java.math.BigDecimal;

import br.edu.unoesc.testejpa.model.Departamento;

import br.edu.unoesc.testejpa.model.Pessoa;

import br.edu.unoesc.testejpa.util.JPAUtil;

import jakarta.persistence.EntityManager;

public class App3 {

private static EntityManager em;

public static void main(String[] args) {

em = JPAUtil.getEntityManager();

Departamento d1 = new Departamento(null, "Desenvolvimento");

Departamento d2 = new Departamento(null, "Testes");

Pessoa p1 = new Pessoa("Maria", java.sql.Date.valueOf("2000-04-01"), new BigDecimal("10000.0"), d1);

Pessoa p2 = new Pessoa("Antônio", java.sql.Date.valueOf("2001-05-02"), new BigDecimal("20000.0"),d2);

Pessoa p3 = new Pessoa("Osvaldo", java.sql.Date.valueOf("2001-05-10"), new BigDecimal("10000.0"),d1);

em.getTransaction().begin();

em.persist(d1);

em.persist(d2);

em.persist(p1);

em.persist(p2);

em.persist(p3);

em.getTransaction().commit();

em.close();

JPAUtil.closeEntityManagerFactory();

}

}

Ao conferir já teremos as duas tabelas com seus dados.

Podemos agora consultar os dados utilizando uma query:

// Procurando uma pessoa pelo id

System.out.println("Primeira pessoa:");

System.out.println(em.find(Pessoa.class, 1));

//Mostrando todas pessoas e seus departamentos

String jpql1 = "SELECT p FROM Pessoa p JOIN Departamento d on p.departamento = d.id";

List<Pessoa> pessoas1 = em.createQuery(jpql1, Pessoa.class).getResultList();

System.out.println("Pessoas e seus departamentos")

System.out.println("Pessoa\t\tDepartamento");

for(Pessoa p: pessoas1) {

System.out.println(p.getNome()+"\t\t"+p.getDepartamento().getNome());

}

//Mostrando as pessoas de um determinado departamento

String jpql2 = "SELECT p FROM Pessoa p WHERE p.departamento.nome = :nome";

List<Pessoa> pessoas2 = em.createQuery(jpql2, Pessoa.class)

.setParameter("nome", "Desenvolvimento")

.getResultList();

System.out.println("Pessoas do departamento de Desenvolvimento");

System.out.println("Pessoa\t\tDepartamento");

for(Pessoa p:pessoas2) {

System.out.println(p.getNome()+"\t\t"+p.getDepartamento().getNome());

}

Vamos agora separar os métodos de DAO.

Para isso vamos antes atualizar a classe JPAUtil assim:

package br.edu.unoesc.testejpa.util;

import jakarta.persistence.EntityManager;

import jakarta.persistence.EntityManagerFactory;

import jakarta.persistence.Persistence;

public class JPAUtil {

**private static final EntityManagerFactory EMF;**

**static {**

**try {**

**EMF = Persistence.createEntityManagerFactory("jpapersistencia");**

**} catch(Exception e) {**

**throw new RuntimeException(e);**

**}**

**}**

private JPAUtil() {

}

public static EntityManager getEntityManager() {

return EMF.createEntityManager();

}

public static void closeEntityManagerFactory() {

EMF.close();

}

}

Vamos criar a classe **DaoPessoa** e **DaoDepartamento** no package **br.edu.unoesc.testejpa.dao** assim:

package br.edu.unoesc.testejpa.dao;

import java.util.List;

import br.edu.unoesc.testejpa.model.Pessoa;

import br.edu.unoesc.testejpa.util.JPAUtil;

import jakarta.persistence.EntityManager;

import jakarta.persistence.TypedQuery;

**public class DaoPessoa {**

private EntityManager em;

public DaoPessoa() {

em = JPAUtil.getEntityManager();

}

// métodos privados

private DaoPessoa abrirTransacao() {

em.getTransaction().begin();

return this;

}

private DaoPessoa fecharTransacao() {

em.getTransaction().commit();

return this;

}

private DaoPessoa incluir(Pessoa p) {

em.persist(p);

return this;

}

private DaoPessoa remover(Pessoa p) {

em.remove(p);

return this;

}

// métodos públicos

public DaoPessoa salvar(Pessoa p) {

return this.abrirTransacao().incluir(p).fecharTransacao();

}

public DaoPessoa excluir(Pessoa p) {

return this.abrirTransacao().remover(p).fecharTransacao();

}

public List<Pessoa> obterTodos() {

String jpql = "SELECT p FROM Pessoa p";

return em.createQuery(jpql, Pessoa.class).getResultList();

}

public Pessoa buscarPorId(Integer id) {

return em.find(Pessoa.class, id);

}

public List<Pessoa> buscarPorNome(String nome) {

String jpql = "SELECT p FROM Pessoa p WHERE p.nome LIKE :nome";

TypedQuery<Pessoa> consulta = em.createQuery(jpql, Pessoa.class);

consulta.setParameter("nome", "%" + nome + "%");

return consulta.getResultList();

}

public void fechar() {

em.close();

}

}

package br.edu.unoesc.testejpa.dao;

import java.util.List;

import br.edu.unoesc.testejpa.model.Departamento;

import br.edu.unoesc.testejpa.util.JPAUtil;

import jakarta.persistence.EntityManager;

import jakarta.persistence.TypedQuery;

**public class DaoDepartamento** {

private EntityManager em;

public DaoDepartamento() {

em = JPAUtil.getEntityManager();

}

// métodos privados

private DaoDepartamento abrirTransacao() {

em.getTransaction().begin();

return this;

}

private DaoDepartamento fecharTransacao() {

em.getTransaction().commit();

return this;

}

private DaoDepartamento incluir(Departamento d) {

em.persist(d);

return this;

}

private DaoDepartamento remover(Departamento d) {

em.remove(d);

return this;

}

// métodos públicos

public DaoDepartamento salvar(Departamento d) {

return this.abrirTransacao().incluir(d).fecharTransacao();

}

public DaoDepartamento excluir(Departamento d) {

return this.abrirTransacao().remover(d).fecharTransacao();

}

public List<Departamento> obterTodos() {

String jpql = "SELECT d FROM Departamento d";

return em.createQuery(jpql, Departamento.class).getResultList();

}

public Departamento buscarPorId(Integer id) {

return em.find(Departamento.class, id);

}

public List<Departamento> buscarPorNome(String nome) {

String jpql = "SELECT d FROM Departamento d WHERE d.nome LIKE :nome";

TypedQuery<Departamento> consulta = em.createQuery(jpql, Departamento.class);

consulta.setParameter("nome", "%" + nome + "%");

return consulta.getResultList();

}

public void fechar() {

em.close();

}

}

E podemos testar as duas classes Dao criadas em um novo app:

package br.edu.unoesc.testejpa.app;

import java.math.BigDecimal;

import java.time.LocalDate;

import java.util.List;

import br.edu.unoesc.testejpa.dao.DaoDepartamento;

import br.edu.unoesc.testejpa.dao.DaoPessoa;

import br.edu.unoesc.testejpa.model.Departamento;

import br.edu.unoesc.testejpa.model.Pessoa;

public class App4 {

public static void main(String[] args) {

DaoDepartamento daoDepartamento = new DaoDepartamento();

DaoPessoa daoPessoa = new DaoPessoa();

Departamento departamento = new Departamento(null, "Marketing");

Pessoa pessoa = new Pessoa("Elisa", java.sql.Date.valueOf(LocalDate.now()), new BigDecimal("12000"),

departamento);

System.out.println(daoDepartamento.salvar(departamento).obterTodos());

daoDepartamento.fechar();

daoPessoa.salvar(pessoa);

List<Pessoa> pessoas = daoPessoa.obterTodos();

for (Pessoa p : pessoas) {

System.out.println(p.getNome() + " - " + p.getDepartamento().getNome());

}

daoPessoa.fechar();

}

}

Para relacionar Pessoa com Departamento utilizamos a anotação @ManyToOne, ou seja, muitas Pessoas podem se relacionar com um Departamento. Mas essa não é a única anotação para relacionamentos. Temos os seguintes relacionamentos por associação:

- Um-para-um (*@OneToOne*)

- Um-para-muitos / Muitos-para-um *(@OneToMany / @ManyToOne*)

- Muitos-para-muitos (*@ManyToMany*)

Temos também as anotações para mapeamento:

- @*mappedBy*

- @*JoinTable*

- @*JoinColumn*

E as estratégias

*- lazy*

*- eager*

Vamos ampliar o projeto acrescentando uma classe **Gerente**, sendo que haverá um gerente para cada departamento, utilizando um relacionamento **um-para-um unidirecional**. A classe Departamento continua como já estava antes.

package br.edu.unoesc.testejpa.model;

import java.io.Serializable;

import jakarta.persistence.CascadeType;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.JoinColumn;

import jakarta.persistence.OneToOne;

import jakarta.persistence.Table;

@Entity

@Table(name = "gerentes")

public class Gerente implements Serializable {

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

private String nome;

**@OneToOne(cascade = CascadeType.ALL)**

**@JoinColumn(unique = true)**

**private Departamento departamento;**

//construtores

//getters/setters

//toString

}

Podemos testar esse relacionamento assim:

package br.edu.unoesc.testejpa.app;

import java.util.List;

import br.edu.unoesc.testejpa.model.Departamento;

import br.edu.unoesc.testejpa.model.Gerente;

import br.edu.unoesc.testejpa.util.JPAUtil;

import jakarta.persistence.EntityManager;

public class App5 {

private static EntityManager em;

public static void main(String[] args) {

em = JPAUtil.getEntityManager();

Departamento departamento1 = new Departamento(null, "Marketing");

Departamento departamento2 = new Departamento(null, "RH");

Gerente gerente1 = new Gerente("Marisa", departamento1);

Gerente gerente2 = new Gerente("Bernardo", departamento2);

em.getTransaction().begin();

em.persist(gerente1);

em.persist(gerente2);

em.getTransaction().commit();

String jpql = "SELECT g FROM Gerente g";

List<Gerente> gerentes = em.createQuery(jpql, Gerente.class).getResultList();

for (Gerente g : gerentes) {

System.out.println(g.getNome() + " - " + g.getDepartamento().getNome());

}

em.close();

JPAUtil.closeEntityManagerFactory();

}

}

Podemos também construir um relacionamento **um-para-um bidirecional** ao relacionar Gerente com Departamento. Agora, as duas classes têm a indicação do relacionamento.

package br.edu.unoesc.testejpa.model;

import java.io.Serializable;

import jakarta.persistence.Column;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.OneToOne;

import jakarta.persistence.Table;

@Entity

@Table(name = "departamentos")

public class Departamento implements Serializable{

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

@Column(length = 40, nullable = false)

private String nome;

**@OneToOne(mappedBy = "departamento")**

**private Gerente gerente;**

//gerar um novo construtor

//getters/setters

//toString

}

E para testar construímos uma nova classe assim:

package br.edu.unoesc.testejpa.app;

import java.util.List;

import br.edu.unoesc.testejpa.model.Departamento;

import br.edu.unoesc.testejpa.model.Gerente;

import br.edu.unoesc.testejpa.util.JPAUtil;

import jakarta.persistence.EntityManager;

public class App6 {

private static EntityManager em;

public static void main(String[] args) {

em = JPAUtil.getEntityManager();

Departamento departamento1 = new Departamento(null, "Marketing", null);

Departamento departamento2 = new Departamento(null, "RH", null);

Gerente gerente1 = new Gerente("Marisa", departamento1);

Gerente gerente2 = new Gerente("Bernardo", departamento2);

departamento1.setGerente(gerente1);

departamento2.setGerente(gerente2);

em.getTransaction().begin();

em.persist(gerente1);

em.persist(gerente2);

em.getTransaction().commit();

String jpql = "SELECT g FROM Gerente g";

List<Gerente> gerentes = em.createQuery(jpql, Gerente.class).getResultList();

for (Gerente g : gerentes) {

System.out.println(g.getNome() + " - " + g.getDepartamento().getNome());

}

jpql = "SELECT d FROM Departamento d";

List<Departamento> departamentos = em.createQuery(jpql, Departamento.class).getResultList();

for (Departamento d : departamentos) {

System.out.println(d.getNome() + " - " + d.getGerente().getNome());

}

em.close();

JPAUtil.closeEntityManagerFactory();

}

}

Em seguida podemos construir DaoDepartamento, a exemplo do que já foi feito antes.