JPA e Hibernate – continuação

Relacionamento um-para-muitos / muitos-para-um (unidirecional e bidirecional)

A primeira estratégia é muitos-para-um (Um funcionário pode ter um cargo e um cargo pode ser de muitos funcionários).

|  |  |
| --- | --- |
| @Entity  @Table(name = "funcionarios")  public class Funcionario 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**  **private Cargo cargo;**  ...  } | @Entity  @Table(name = "cargos")  public class Cargo implements Serializable{  private static final long serialVersionUID = 1L;    @Id  @GeneratedValue(strategy = GenerationType.IDENTITY)  private Integer id;    @Column(nullable = false, length = 50)  private String nome;  ...  } |

Vamos melhorar fazendo com que o relacionamento seja bidirecional:

|  |  |
| --- | --- |
| @Entity  @Table(name = "funcionarios")  public class Funcionario implements Serializable {  ...  **@ManyToOne**  **private Cargo cargo;**    **public Funcionario(String nome, Date dataNascimento, BigDecimal salario, Cargo cargo) {**  **this.nome = nome;**  **this.dataNascimento = dataNascimento;**  **this.salario = salario;**  **cargo.adicionarFuncionario(this);**  **}** | @Entity  @Table(name = "cargos")  public class Cargo implements Serializable{  private static final long serialVersionUID = 1L;    @Id  @GeneratedValue(strategy = GenerationType.IDENTITY)  private Integer id;    @Column(nullable = false, length = 50)  private String nome;    **@OneToMany (mappedBy = "cargo")**  **private Set<Funcionario>funcionarios = new**  **HashSet<Funcionario>();**  public Cargo(Integer id, String nome) {  super();  this.id = id;  this.nome = nome;  }  **public void adicionarFuncionario(Funcionario funcionario) {**  **funcionario.setCargo(this);**  **this.funcionarios.add(funcionario);**  **}** |

Vamos também construir **DaoFuncionario** e **DaoCargo**

package br.edu.unoesc.testejpa.dao;

import java.util.List;

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

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

import jakarta.persistence.EntityManager;

import jakarta.persistence.TypedQuery;

**public class DaoFuncionario {**

private EntityManager em;

public DaoFuncionario() {

em = JPAUtil.getEntityManager();

}

// métodos privados

private DaoFuncionario abrirTransacao() {

em.getTransaction().begin();

return this;

}

private DaoFuncionario fecharTransacao() {

em.getTransaction().commit();

return this;

}

private DaoFuncionario incluir(Funcionario f) {

em.persist(f);

return this;

}

private DaoFuncionario remover(Funcionario f) {

em.remove(f);

return this;

}

// métodos públicos

public DaoFuncionario salvar(Funcionario f) {

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

}

public DaoFuncionario excluir(Funcionario f) {

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

}

public List<Funcionario> obterTodos() {

String jpql = "SELECT f FROM Funcionario f";

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

}

public Funcionario buscarPorId(Integer id) {

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

}

public List<Funcionario> buscarPorNome(String nome) {

String jpql = "SELECT f FROM Funcionario f WHERE f.nome LIKE :nome";

TypedQuery<Funcionario> consulta = em.createQuery(jpql, Funcionario.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.Cargo;

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

import jakarta.persistence.EntityManager;

import jakarta.persistence.TypedQuery;

**public class DaoCargo {**

private EntityManager em;

public DaoCargo() {

em = JPAUtil.getEntityManager();

}

// métodos privados

private DaoCargo abrirTransacao() {

em.getTransaction().begin();

return this;

}

private DaoCargo fecharTransacao() {

em.getTransaction().commit();

return this;

}

private DaoCargo incluir(Cargo c) {

em.persist(c);

return this;

}

private DaoCargo remover(Cargo c) {

em.remove(c);

return this;

}

// métodos públicos

public DaoCargo salvar(Cargo c) {

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

}

public DaoCargo excluir(Cargo c) {

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

}

public List<Cargo> obterTodos() {

String jpql = "SELECT c FROM Cargo c";

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

}

public Cargo buscarPorId(Integer id) {

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

}

public List<Cargo> buscarPorNome(String nome) {

String jpql = "SELECT c FROM Cargo c WHERE c.nome LIKE :nome";

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

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

return consulta.getResultList();

}

public void fechar() {

em.close();

}

}

E o app para testar fica assim:

package br.edu.unoesc.testejpa.app;

...

public class App7 {

public static void main(String[] args) {

DaoCargo daoCargo = new DaoCargo();

Cargo cargo1 = new Cargo("Gerente");

Cargo cargo2 = new Cargo("Assistente");

daoCargo.salvar(cargo1).salvar(cargo2);

DaoFuncionario daoFuncionario = new DaoFuncionario();

Funcionario funcionario1 = new Funcionario("Paula", java.sql.Date.valueOf("2000-04-01"),

new BigDecimal("10000.0"), cargo1);

Funcionario funcionario2 = new Funcionario("Ana", java.sql.Date.valueOf("2001-04-01"),

new BigDecimal("20000.0"), cargo2);

Funcionario funcionario3 = new Funcionario("Eduarda", java.sql.Date.valueOf("2002-04-01"),

new BigDecimal("30000.0"), cargo2);

daoFuncionario.salvar(funcionario1).salvar(funcionario2).salvar(funcionario3);

System.out.println(daoFuncionario.buscarPorId(1));

System.out.println(daoFuncionario.buscarPorId(2));

List<Funcionario> funcionarios = daoFuncionario.obterTodos();

System.out.println("Funcionario\tCargo");

for (Funcionario f : funcionarios) {

System.out.println(f.getNome() + "\t\t" + f.getCargo().getNome());

}

List<Cargo> cargos = daoCargo.obterTodos();

System.out.println("Cargos e Funcionarios");

for (Cargo c : cargos) {

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

for (Funcionario f : c.getFuncionarios()) {

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

}

}

daoCargo.fechar();

daoFuncionario.fechar();

}

}

Relacionamento muitos-para-muitos (ManyToMany)

|  |  |  |  |
| --- | --- | --- | --- |
| @Entity  @Table(name = "projetos")  public class Projeto implements Serializable {  private static final long serialVersionUID = 1L;  @Id  @GeneratedValue(strategy = GenerationType.IDENTITY)  private Integer id;  @Column(length = 50, nullable = false)  private String nome;  **@ManyToMany(mappedBy = "projetosFuncionario",**  **//cascade = CascadeType.ALL, fetch = FetchType.EAGER**  **)**  **private List<Funcionario> funcionarios;**  public Projeto() {  }  ...  public void adicionarFuncionario(Funcionario funcionario) {  if (funcionario != null && !this.getFuncionarios().contains(funcionario)) {  this.funcionarios.add(funcionario);  if (!funcionario.getProjetos().contains(this)) {  funcionario.getProjetos().add(this);  }  }  }  public List<Funcionario> getFuncionarios() {  if (this.funcionarios == null) {  this.funcionarios = new ArrayList<>();  }  return funcionarios;  }  ...  } | @Entity  @Table(name = "funcionarios")  public class Funcionario implements Serializable {  private static final long serialVersionUID = 1L;  @Id  @GeneratedValue(strategy = GenerationType.IDENTITY)  private Integer id;  @Column(length = 50, nullable = false)  private String nome;  @Column(name = "data\_nascimento", nullable = false)  private LocalDate dataNascimento;  @Column(precision = 12, scale = 2, nullable = false)  private BigDecimal salario;  **@ManyToMany // (cascade = CascadeType.ALL,**  **fetch = FetchType.EAGER)**  **@JoinTable(name = "funcionarios\_projeto",**  **joinColumns = @JoinColumn(name = "id\_funcionario"),**  **inverseJoinColumns = @JoinColumn(name = "id\_projeto"))**  **private List<Projeto> projetosFuncionario;**  public Funcionario() {  }  ...  public void adicionarProjeto(Projeto projeto) {  if (projeto != null && !this.getProjetos().contains(projeto)) {  this.projetosFuncionario.add(projeto);  if (!projeto.getFuncionarios().contains(this)) {  projeto.getFuncionarios().add(this);  }  }  }  public List<Projeto> getProjetos() {  if (projetosFuncionario == null) {  projetosFuncionario = new ArrayList<>();  }  return projetosFuncionario;  }  ...  } |  |  |

Para testar usamos App8 dessa forma:

package br.edu.unoesc.testejpa.app;

...

public class App8 {

public static EntityManager em;

public static void main(String[] args) {

adicionarDados();

listarPorProjeto();

listarPorFuncionario();

}

public static void adicionarDados() {

em = JPAUtil.getEntityManager();

Projeto p1 = new Projeto("Projeto P1");

Projeto p2 = new Projeto("Projeto P2");

Funcionario f1 = new Funcionario("Ana", java.sql.Date.valueOf("2000-01-01"), new BigDecimal("1000.00"));

Funcionario f2 = new Funcionario("Bernardo", java.sql.Date.valueOf("2000-04-01"), new BigDecimal("2000.00"));

Funcionario f3 = new Funcionario("Carlos", java.sql.Date.valueOf("2000-01-01"), new BigDecimal("3000.00"));

// adiciona Ana e Bernardo ao Projeto P1

p1.adicionarFuncionario(f1);

p1.adicionarFuncionario(f2);

// adiciona o Projeto P2 aos funcionários Ana e Carlos

f1.adicionarProjeto(p2);

f3.adicionarProjeto(p2);

em.getTransaction().begin();

em.persist(p1);

em.persist(p2);

em.persist(f1);

em.persist(f2);

em.persist(f3);

em.getTransaction().commit();

em.close();

}

public static void listarPorProjeto() {

em = JPAUtil.getEntityManager();

String jpql = "SELECT p FROM Projeto p";

List<Projeto> projetos = em.createQuery(jpql, Projeto.class).getResultList();

System.out.println("================================");

System.out.println("Projetos e seus funcionários");

System.out.println("================================");

for (Projeto p : projetos) {

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

for (Funcionario f : p.getFuncionarios())

System.out.println("\t" + f.getId() + " - " + f.getNome());

}

}

public static void listarPorFuncionario() {

em = JPAUtil.getEntityManager();

String jpql = "SELECT f FROM Funcionario f";

List<Funcionario> funcionarios = em.createQuery(jpql, Funcionario.class).getResultList();

System.out.println("================================");

System.out.println("Funcionários e seus projetos");

System.out.println("================================");

for (Funcionario f : funcionarios) {

System.out.println(f.getId() + " - " + f.getNome());

for (Projeto p : f.getProjetos()) {

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

}

}

}

}

Podemos agora construir as classes DAO faltantes para concluir o projeto e reconstruir a classe de testes utilizando as classes DAO.