
Residência em Software
Residência em Tecnologia da Informação e Comunicação
JPQL - Introdução
Professor:
Alvaro Degas Coelho
Java
INSTITUIÇÃO EXECUTORA
cepedi
UESC
COORDENADORA
MCTI FUTURO
Softex
APOIO
MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO
GOVERNO FEDERAL
BRASIL
UNIÃO E RECONSTRUÇÃO

JPQL – Java Persistence Query Language

- É uma linguagem de consulta
  - Também faz DML
- Baseada em SQL
- Usada em Java
- Ideia geral: operar em classes ao invés de tabelas

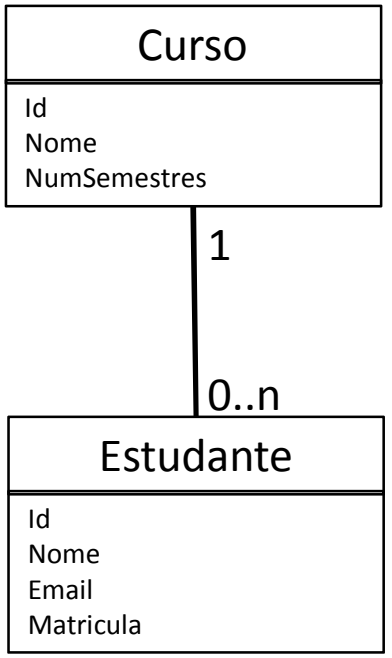
Residência em Software

Classes-Entidade

- JPQL utiliza classes entidade
  - Annotation @Entity
- E liga com classes do programa
- Vamos criar um exemplo para os acompanhar

Classes-Entidade

- JPQL utiliza classes entidade
  - Annotation @Entity
- E liga com classes do programa
- Vamos criar um exemplo para os acompanhar



Criando esta estrutura no BD

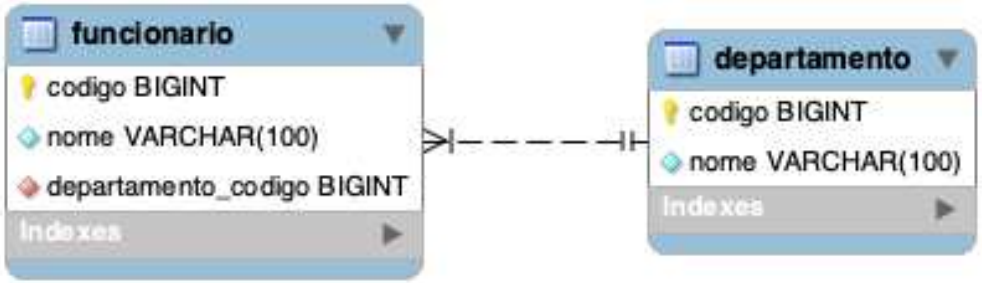
```
CREATE TABLE Curso (
        Id INT NOT NULL AUTO_INCREMENT,
        Nome VARCHAR(30) NOT NULL,
        NumSemestres INT NOT NULL,
        PRIMARY KEY (Id),
        UNIQUE(Nome)
);


CREATE TABLE Estudante (
        Id INT NOT NULL AUTO_INCREMENT ,
        IdCurso INT NOT NULL ,
        Nome VARCHAR(50) NOT NULL ,
        Email VARCHAR(30) NOT NULL ,
        Matricula VARCHAR(12) NOT NULL ,
        PRIMARY KEY (Id),
        UNIQUE (Email),
        UNIQUE (Matricula)
);
ALTER TABLE Estudante
        ADD FOREIGN KEY (IdCurso)
        REFERENCES Curso(Id)
        ON DELETE RESTRICT ON UPDATE RESTRICT;
```

Deixando esta tarefa para o JPA

- Classes precisam ser Entity
- Classes precisam registrar suas chaves primárias
- **Classes precisam registrar suas chaves estrangeiras**
  - **Annotations @OneToMany e ManyToOne**

Relacionamentos 1..N em JPA

- Observe este exemplo (depois implementaremos o nosso)



Mapeando as entidadres

```java
@Entity
@Table(name = "departamento")
public class Departamento {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Integer codigo;

  private String nome;

  // getters e setters omitidos
}
```

```java
@Entity
@Table(name = "funcionario")
public class Funcionario {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Integer codigo;

  private String nome;

  // getters e setters omitidos

}
```

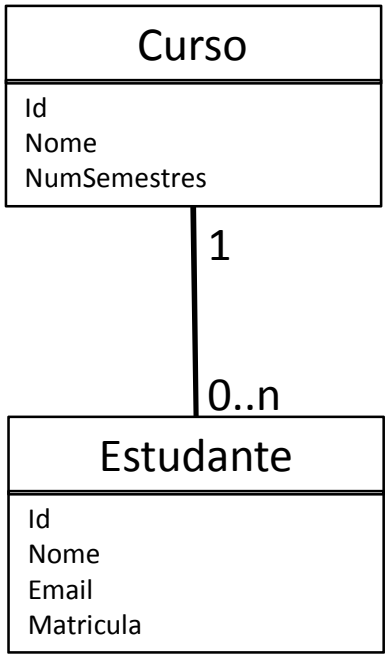
Usando a anotação ManyToOne

```java
@Entity
@Table(name = "departamento")
public class Departamento {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Integer codigo;

  private String nome;

  // getters e setters omitidos

}
```

```java
@Entity
@Table(name = "funcionario")
public class Funcionario {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Integer codigo;

  private String nome;

  // adicionamos aqui o mapeamento muitos-para-um
  @ManyToOne
  private Departamento departamento;

  // getters e setters omitidos

}
```

Voltando ao nosso exemplo

```
@Entity
public class Curso {
    @Id
    @GeneratedValue (strategy = GenerationType.IDENTITY)
    private Integer Id;
    private String Nome;
    private Integer NumSemestres;
      ...
  public class Estudante {
      @Id
      @GeneratedValue (strategy=GenerationType.IDENTITY)
      private Integer Id;
      @ManyToOne
      private Curso Curso;
      private String Nome;
      private String Email;
      private String Matricula;
      ...
```



Podemos testar isso

- Continuando o projeto da aula anterior
  - Certifique-se que a classe Estudante esteja correta
  - Crie a classe Curso na mesma package
    - Certifique-se que os atributos estejam corretos
- Não esquecer dos getters e setters, do construtor geral e do construtor com atributos!

Vamos “forçar” o BD a criar nossa estrutura

- No arquivo persistence.xml
- Modifique o parâmetro
  - *<property name="hibernate.hbm2ddl.auto“*
  - Ponha *value=“create”*
    - Isto forçará o Hibernate a recriar as tabelas a partir de nossas entidades
  - Bom para desenvolver. Perigoso em qualquer outro contexto

Criando (e populando) a estrutura do BD

- Podemos fazer isso via SQL
- Ideia geral do método preparaBD()
  - Instanciar objetos do tipo Curso
  - Instanciar objetos do tipo Estudante (ligados aos cursos)
  - Persistir os cursos (senão vai falhar a chave estrangeira)
  - Persistir os Estudantes
  - Commit

Classe TesteDAO

- Crie dentro de uma package “dao”
- 1ª parte do Método preparaBD()
  - Recebe um EntityManager

```
public static void preparaBD(EntityManager em) {

    Curso c1 = new Curso(null, "Matemática", 8);
    Curso c2 = new Curso(null, "Computação", 10);
    Curso c3 = new Curso(null, "Geografia", 8);
    Estudante e1 = new Estudante(null, c1, "Tõe", "toe@tutu", "111111");
    Estudante e2 = new Estudante(null, c1, "Lia", "lia@tutu", "222222");
    Estudante e3 = new Estudante(null, c1, "Tuca", "tuca@tutu", "333333");
    Estudante e4 = new Estudante(null, c2, "Peu", "peu@tutu", "4444");
    Estudante e5 = new Estudante(null, c2, "Leo", "leo@tutu", "55555");
    Estudante e6 = new Estudante(null, c3, "Val", "val@tutu", "66666");
```

Classe TesteDAO

- 2ª Parte

```
        em.getTransaction().begin();
        em.persist(c1);
        em.persist(c2);
        em.persist(c3);
        em.persist(e1);
        em.persist(e2);
        em.persist(e3);
        em.persist(e4);
        em.persist(e5);
        em.persist(e6);
        em.getTransaction().commit();

    }
```

Executando preparaBD()

- Passos
  - Instanciar um EntityManager Factory
  - Instanciar um EntityManager
  - Invocar preparaBD() passando o EM

```
public static void main(String[] args) {
    EntityManagerFactory emf =
            Persistence.createEntityManagerFactory("exemplo-jpa");
    EntityManager em = emf.createEntityManager();
    TesteDAO.preparaBD(em);
    System.out.println("BD Recriado");
```

Programa aí!

Observação

- É possível se definir o nome da coluna que vai ser criada como chave estrangeira
  - Força a criação de um List na entidade que exporta a chave
  - Anotação OneToMany

Residência em Software

Fica

```java
@Entity
public class Curso {
    @Id
    @GeneratedValue (strategy = GenerationType.IDENTITY)
    private Integer Id;
    private String Nome;
    private Integer NumSemestres;
    @OneToMany (mappedBy="Curso")
    List<Estudante> listaEstudantes;
```

```java
@Entity
public class Estudante {

    @Id
    @GeneratedValue (strategy=GenerationType.IDENTITY)
    private Integer Id;
    @ManyToOne
    @JoinColumn(name="CodCurso")
    private Curso Curso;
    private String Nome;
    private String Email;
    private String Matricula;
```

Residência em Software

Voltando ao JPQL

- Um exemplo padrão
  - Select * from Estudante
- Lembrando: não usamos tabelas, apenas classes
  - Select e from estudante e
- TypedQuery: implementa o “executador” de sentenças JPQL

Residência em Software

Usando o JPQL

- Uma String informa a sentença
- O TypedQuery é associado ao tipo de dado retornado na consulta
  - TypedQuery<Classe>
- Como fica
  - String jpql = "select e from Estudante e";
  - TypedQuery<Estudante> typedQuery = em.createQuery(jpql, Estudante.**class**);

Residência em Software

Executando

- Método GetResult() do TypedQuery
  - Retorna uma lista com os objetos da classe
- Fica
  - List<Estudante> lista = typedQuery.getResultList();
- Agora podemos usar a lista normalmente no programa

Um exemplo

- Para ilustrar o uso e o retorno
- Um método para listar todos os estudantes
  - **public static void** listarTodosEstudantes(EntityManager em)
- Roteiro
  - Criar a sentença JPQL
  - Instanciar o TypedQuery
  - Executar o TypedQuery
  - Obter o resultado numa Lista

Programa aí!

Residência em Software

Como fica

```
public static void listarTodosEstudantes(EntityManager em) {
    String jpql = "select e from Estudante e";
    TypedQuery<Estudante> typedQuery =
            em.createQuery(jpql, Estudante.class);
    List<Estudante> lista = typedQuery.getResultList();
    for (Estudante e:lista)
        System.out.println(e);
}
```

Selecionar um objeto

- Método selecionaUmEstudante()
- Buscas pela chave primária ou por chaves alternativas
  - Retorna apenas um objeto
    - Método getSingleResult() do Typed Query
- Roteiro (bem parecido)
  - Criar a sentença JPQL (...where Id=1)
  - Instanciar o TypedQuery
  - Executar o TypedQuery
  - Obter o resultado num Objeto

Programa aí!

De volta ao contexto de persistência

- O objeto que acabou de ser retornado
  - Estudante com Id=1
- Pode ser alterado
  - Método setEmail
- E pode ser persistido novamente
  - Transação

Programa aí!

Residência em Software

Como fica

```
public static void alterarEstudante(EntityManager em) {

    String jpql = "select e from Estudante e where Id=1";
    TypedQuery<Estudante> typedQuery =
            em.createQuery(jpql, Estudante.class);
    Estudante e = typedQuery.getSingleResult();
    System.out.println(e);
    em.getTransaction().begin();
    e.setEmail("totonho@tutu");
    em.persist(e);
    em.getTransaction().commit();
}
```

Residência em Software

Seleção

- Não somos obrigados a retornar um objeto completo
  - Select nome from Estudante
- O TypedQuery agora se associa a objetos do tipo String
  - TypedQuery<String> typedQuery = em.createQuery(jpql, String.**class**);

Método ilustrando

- Método listarNomesEstudantes()
- Roteiro
  - Criar a sentença JPQL
  - Instanciar o TypedQuery (associado a String)
  - Executar o TypedQuery (String.class)
  - Obter o resultado numa Lista (de String)

Programa aí!

Residência em Software

Como fica

```
public static void listarNomesDosEstudantes(EntityManager em) {
    String jpql = "select e.Nome from Estudante e";
    TypedQuery<String> typedQuery =
            em.createQuery(jpql, String.class);
    List<String> lista = typedQuery.getResultList();
    for (String e:lista)
        System.out.println(e);
}
```

DTO

- Data Transfer Object
  - Usado para devolver à camada de aplicação apenas o dado que ela precisa
    - Omitir o que não é necessário (senha, salário, etc)
- Nosso exemplo
  - Criar a classe EstudanteDTO (package dto)
    - Nome
    - Matrícula
    - Getters, Stters, Construtores

Adaptando a sentença

- Queremos que o JPQL instancie objetos EstudanteDTO
  - JPQL não sabe em que package está a classe
  - JPQL não sabe quais objetos devem ir no construtor
  - select new dao.EstudanteDTO(e.Nome, e.Email, e.Matricula) from Estudante e
- TypedQuery associado a EstudanteDTO

Método para testar o DTO

- Método gerarEstudanteDTO()
- Roteiro
  - Criar a sentença JPQL (new dto.EstudanteDTO...)
  - Instanciar o TypedQuery (associado a EstudanteDTO)
  - Executar o TypedQuery (EstudanteDTO.class)
  - Obter o resultado numa Lista (de EstudanteDTO)

Join implícito

- Se quisermos pegar o nome do curso do estudante
  - SQL ... Where e.curdo=c.Id
- JPQL
  - "select new dao.EstudanteDTO(e.Nome, e.Email, e.Matricula, **e.Curso.Nome**) from Estudante e";
- Claro que agora é necessário haver a String Curso em EstudanteDTO

Programa aí!

Residência em Software

Como fica

```
public static void gerarEstuanteDTO(EntityManager em) {
    String jpql = "select new dao.EstudanteDTO(e.Nome, "
            + "e.Email, e.Matricula, e.Curso.Nome) from Estudante e";
    TypedQuery<EstudanteDTO> typedQuery =
            em.createQuery(jpql, EstudanteDTO.class);
    List<EstudanteDTO> lista = typedQuery.getResultList();
    for (EstudanteDTO e:lista)
        System.out.println(e);
}
```

Residência em Software

Parâmetros

- É pssível passar e receber parâmetros
  - Identificados por : na sentença JPQL
  - Substituídos pelo valor correto com o método setParameter() do Typed Query
- Exemplo
  - Select c from Curso where NumSemestres = :num
  - typedQuery.setParameter(“num”, 8)
- Executará como
  - Select c from Curso where NumSemestres =8

Mais um método de teste

- Verificar a possibilidade de listar todos os cursos com menos que 9 semestres
- Roteiro
  - Criar a sentença JPQL (usar parâmetro)
  - Instanciar o TypedQuery
  - Executar o TypedQuery Obter o resultado numa Lista

Residência em Software

Consultas agregadas

- Podemos usar um objeto instanciado pelo JPQL (ou pelo EntityManager)
  - Estará no contexto de referência
- E usá-lo como parâmetro para numa nova consulta
- Equivale ao subquery (ou ao join)

Um método de exemplo

- Objetivo: selecionar os estudantes
- A partir do código de um curso
- Roteiro
  - Selecionar o objeto curso (usando o parâmetro)
  - Construir a sentença de busca dos estudantes
    - Usando o curso como parâmetro
  - Construir a lista de estudantes

Programa aí!

Residência em Software

Como fica

```
public static void mostrarEstudantesPorCurso(EntityManager em) {
    String jpql = "select c from Curso c where c.Id = 1";
    TypedQuery<Curso> typedQuery =
            em.createQuery(jpql, Curso.class);
    Curso c = typedQuery.getSingleResult();
    String jpqlEstudante = "select e from Estudante "
            + "e where e.Curso = :curso";
    TypedQuery<Estudante> typedQueryEstudante =
            em.createQuery(jpqlEstudante, Estudante.class);
    typedQueryEstudante.setParameter("curso", c);
    List<Estudante> lista = typedQueryEstudante.getResultList();
    for (Estudante e: lista)
        System.out.println(e);
}
```

Residência em Software

Exercício

- Crie um DAO para operações CRUD de Estudante
  - Inclua busca por Id, busca por nome e por matrícula
- Crie um DAO para operações CRUD de Curso
  - Inclua busca por Nome
  - Inclua um método que retorna a lista de estudantes por curso (dado o nome ou o Id)