Módulo 09

Hibernate: Consultas com
SQLQuery, **Criteria** e **HQL**.

Raphaela Galhardo
raphaela@jeebrasil.com.br

Introdução

- Até Agora:
  - Consulta por chave primária;
  - Recuperação via mapeamentos de relacionamentos:
    - @OneToMany;
    - @ManyToOne;
    - @OneToOne;
    - @ManyToMany;
    - @CollectionOfElements.

Introdução

- Outras Possibilidades com Hibernate:
  - SQL Nativo (SQLQuery);
  - Consulta por Critérios (Criteria);
  - Hibernate Query Language (Query).

SQL NATIVE

Interface SQLQuery

SQL Nativo

- O Hibernate permite especificar SQL escrito manualmente:
  - Utilização da Interface **SQLQuery**;
  - A partir do objeto Session:
    - **Session.createSQLQuery()**.

SQL Nativo – Consultas Escalares

```
//... Consultas Escalares (*)
Session session = sf.openSession();

SQLQuery sqlQuery = session.createSQLQuery(
                          "SELECT * FROM ANOTACOES.DEPARTAMENTO");
List resultado = sqlQuery.list();

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

        //Retorna cada linha da tabela
        Object[] linha = (Object[]) resultado.get(i);
        System.out.print("ID = " + linha[0] + " --> ");
        System.out.println("NOME = " + linha[1] + "\n");

}
session.close();
//...
```

SQL Nativo – Consultas Escalares

```
Hibernate: SELECT * FROM ANOTACOES.DEPARTAMENTO

ID = 210 --> NOME = DEPARTAMENTO DE ENG. COMPUTACAO

ID = 270 --> NOME = DEPARTAMENTO DE HISTORIA

ID = 300 --> NOME = DEPARTAMENTO DE MATEMATICA

ID = 310 --> NOME = DEPARTAMENTO DIREITO
```

SQL Nativo – Consultas Escalares

```
//... Consultas Escalares (ESPECIFICANDO COLUNAS)
Session session = sf.openSession();

SQLQuery sqlQuery = session.createSQLQuery(
    "SELECT ID_DEPARTAMENTO, NOME FROM ANOTACOES.DEPARTAMENTO");
List resultado = sqlQuery.list();

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

        //Retorna cada linha da tabela
        Object[] linha = (Object[]) resultado.get(i);
        System.out.print("ID = " + linha[0] + " --> ");
        System.out.println("NOME = " + linha[1] + "\n");

}
session.close();
//...
```

SQL Nativo – Consultas Escalares

```
Hibernate: SELECT ID_DEPARTAMENTO, NOME
                  FROM ANOTACOES.DEPARTAMENTO

ID = 210 --> NOME = DEPARTAMENTO DE ENG. COMPUTACAO

ID = 270 --> NOME = DEPARTAMENTO DE HISTORIA

ID = 300 --> NOME = DEPARTAMENTO DE MATEMATICA

ID = 310 --> NOME = DEPARTAMENTO DIREITO
```

SQL Nativo – Sendo Mais Explícito

```
//... Consultas Escalares (ESPECIFICANDO COLUNAS)

Session session = sf.openSession();

SQLQuery sqlQuery = session.createSQLQuery(
        "SELECT * FROM ANOTACOES.DEPARTAMENTO");

sqlQuery.addScalar("ID_DEPARTAMENTO", Hibernate.INTEGER);
sqlQuery.addScalar("NOME", Hibernate.STRING);

List resultado = sqlQuery.list();
//...
```

```
Hibernate: SELECT ID_DEPARTAMENTO, NOME
                  FROM ANOTACOES.DEPARTAMENTO
```

SQL Nativo – Entity Queries

```
//... Entity Queries - Especificando o Tipo de Retorno

Session session = sf.openSession();

SQLQuery sqlQuery = session.createSQLQuery(
"SELECT * FROM ANOTACOES.DEPARTAMENTO WHERE ID_DEPARTAMENTO = ?");

sqlQuery.setInteger(0, 210); //Especificando parâmetros...

sqlQuery.addEntity(Departamento.class); //Especifica tipo do retorno

Departamento departamento =
        (Departamento) sqlQuery.uniqueResult(); //Retorna um resultado

session.close();

//...
```

SQL Nativo – Entity Queries

```
//... Entity Queries

Session session = sf.openSession();

SQLQuery sqlQuery = session.createSQLQuery(
                          "SELECT * FROM ANOTACOES.DEPARTAMENTO");

sqlQuery.addEntity(Departamento.class);

//Retorna todos os itens em uma coleção
Collection<Departamento> resultado = sqlQuery.list();

session.close();

//...
```

SQL Nativo – Join's

```
//... Entity Queries com Join

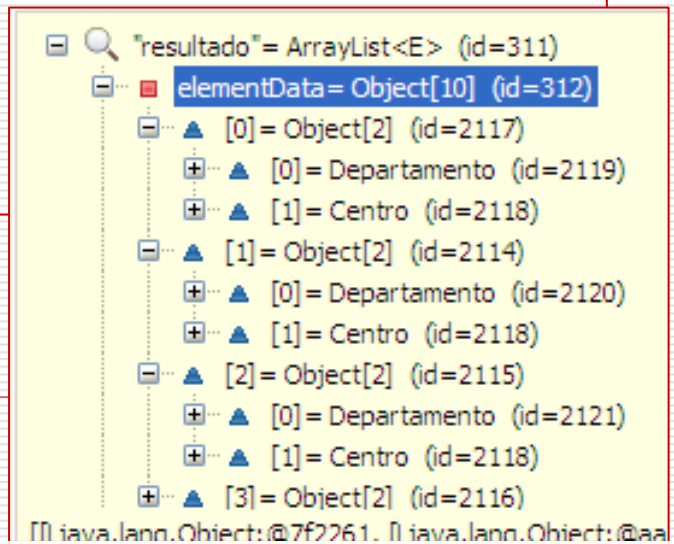
Session session = sf.openSession();

SQLQuery sqlQuery = session.createSQLQuery(
        "SELECT * FROM ANOTACOES.DEPARTAMENTO D, ANOTACOES.CENTRO C
         WHERE D.ID_CENTRO = C.ID_CENTRO");

sqlQuery.addEntity("departamento", Departamento.class);
sqlQuery.addJoin("centro", "departamento.centro");

List resultado = sqlQuery.list();


//...
```



SQL Nativo – Join's

```
//... Entity Queries com Join

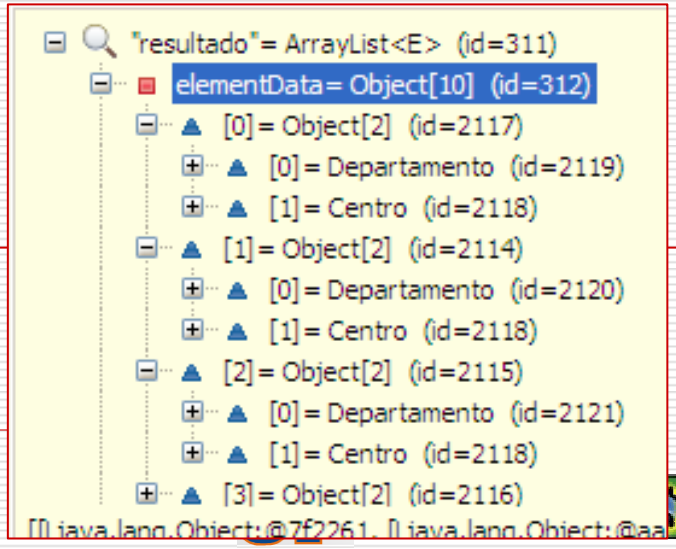
ArrayList<Departamento> departamentos = new ArrayList<Departamento>();
for (int i = 0; i < resultado.size(); i++) {

    Object[] lista = (Object[]) resultado.get(i);

    Departamento departamento = (Departamento) lista[0];
    departamento.setCentro((Centro) lista[1]);
    departamentos.add(departamento);

}
session.close();

//...
```



Exercícios

- Para o domínio da Clínica Veterinária:
  - AnimalDAO:
    - Todos os animais de determinado Cliente:
      - `public Collection<Animal> findByCliente(int idCliente){...}`
      - `SELECT * FROM ANIMAL WHERE ID_CLIENTE = ?`
    - Todos os animais atendidos por determinado Veterinário:
      - `public Collection<Animal> findByVeterinario(int idVeterinario){...}`
      - ```
        SELECT DISTINCT A.* FROM ANIMAL A, CONSULTA C
         WHERE C.ID_VETERINARIO = ? AND
         C.ID_ANIMAL = A.ID_ANIMAL
        ```
        ```

Exercícios

- ConsultaDAO:
  - Todos as consultas realizadas em determinado período:
    - `public Collection<Consulta> findByPeriodo(Date dataInicial, Date dataFinal){...}`
    - `SELECT * FROM CONSULTA WHERE DATA >= ? AND DATA <= ?`

HQL - Hibernate Query Language

Interface Query

HQL – Hibernate Query Language

- Linguagem de consulta;
- Sintaxe orientada a objetos;
- De fácil entendimento;
- Suporta:
  - Herança, polimorfismo;
  - Associações;
  - Agregações, ordenações, paginação;
  - Sub-consultas, joins;
  - Projeções, etc.

HQL – Hibernate Query Language

- Exemplos:

```
from Aluno //Busca todos

from Aluno as alu //Uso de alias

from Aluno aluno where aluno.matricula >= 35

from Endereco end
        where
           ( end.rua in
                ("Bernardo Vieira", "Prudente de Morais") )
      or ( end.numero between 1 and 100 )

from Professor p where p.nome like "João%"

from Aluno aluno order by aluno.nome asc
```

HQL – Hibernate Query Language

- Exemplos (JOIN's):
  - inner join;
  - left outer join;
  - right outer join;
  - full join.

HQL – Hibernate Query Language

- Exemplos (JOIN's):

```
from Cat as cat
      inner join cat.mate as mate
      left outer join cat.kittens as kitten
```

```
from Cat as cat left join cat.mate.kittens as kittens
```

```
from Formula form full join form.parameter param
```

```
from Cat as cat where cat.mate.name like '%s%'
```

HQL – Hibernate Query Language

- Cláusula select:

```
select mate from Cat as cat join cat.mate as mate
```

```
select cat.mate from Cat cat
```

```
select cat.name from DomesticCat cat where cat.name like 'fri%'
```

```
select cust.name.firstName from Customer as cust
```

HQL – Hibernate Query Language

- Cláusula select:

```
//Retorno em um array de Object[]
select mother, offspr, mate.name
        from DomesticCat as mother
        inner join mother.mate as mate
        left outer join mother.kittens as offspr
```

```
select new Family(mother, mate, offspr)
        from DomesticCat as mother
        join mother.mate as mate
        left join mother.kittens as offspr
```

HQL – Hibernate Query Language

- Cláusula where:

```
from Cat where name='Fritz'
```

```
from Cat as cat where cat.name='Fritz'
```

```
select foo from Foo foo, Bar bar where foo.startDate = bar.date
```

```
from Cat cat where cat.mate.name is not null
```

```
select cat, mate from Cat cat, Cat mate where cat.mate = mate
```

```
from AuditLog log, Payment payment
        where log.item.class = 'Payment' and log.item.id = payment.id
```

HQL – Hibernate Query Language

- Expressões:

```
from DomesticCat cat where cat.name between 'A' and 'B'
```

```
from DomesticCat cat
      where cat.name in ( 'Foo', 'Bar', 'Baz' )
```

```
from Cat cat where cat.alive = true
```

```
from Cat cat where size(cat.kittens) > 0
```

HQL – Hibernate Query Language

- Expressões:

```
from Empregado e where e.empresa.tipo = 1 and
       e.salario > 1500 and e.idade > e.esposa.idade
```

```
from DomesticCat cat where upper(cat.name) like 'FRI%'
```

```
from DomesticCat cat where cat.name
       not in ( 'Foo', 'Bar', 'Baz' )
```

HQL – Hibernate Query Language

- Funções de Agregação:

```
//Retorno em um array de Object[]
select max(bodyWeight) as max, min(bodyWeight) as min,
       count(*) as n from Cat cat
```

```
select avg(cat.weight), sum(cat.weight), max(cat.weight),
       count(cat) from Cat cat
```

```
select c.nome, count(c.id_departamento)
      from Curso c group by c.nome
```

HQL – Hibernate Query Language

- Funções de Agregação:

```
select distinct cat.name from Cat cat
```

```
select count(distinct cat.name), count(cat) from Cat cat
```

HQL – Hibernate Query Language

- Group By e Order By:

```
from DomesticCat cat
        order by cat.name asc, cat.weight desc, cat.birthdate
```

```
select cat.color, sum(cat.weight), count(cat)
from Cat cat group by cat.color
```

HQL – Hibernate Query Language

- Group By e Order By:

```
select cat.age, sum(cat.weight), count(cat)
       from Cat cat
       group by cat.color
       having cat.age >= 2
```

```
select cat
       from Cat cat
              join cat.kittens kitten
       group by cat.id, cat.name, cat.other, cat.properties
       having avg(kitten.weight) > 100
       order by count(kitten) asc, sum(kitten.weight) desc
```

HQL – Hibernate Query Language

- Subqueries:

```
from Cat as fatcat
      where fatcat.weight >
      (select avg(cat.weight) from DomesticCat cat)
```

```
from Cat as cat
      where not exists (
            from Cat as mate where mate.mate = cat)
```

```
from Cat as cat
      where not ( cat.name, cat.color ) in
            (select cat.name, cat.color
                  from DomesticCat cat)
```

HQL – Hibernate Query Language

- Usando...

Lista Todos os Alunos

```
Query q = session.createQuery("from Aluno");
q.list();
```

HQL – Hibernate Query Language

Todos os Cursos começados pela letra "A"

```
String hql = "from Curso c where c.nome like 'A%'";
Query q = session.createQuery(hql);
q.list();
```

Todos os Cursos que o departamento tem id = 2

```
String hql = "from Curso c where c.departamento.id = 2;
Query q = session.createQuery(hql);
q.list();
```

HQL – Join Estilo Theta

- Join estilo Theta:
  - Usado para fazer joins entre classes que não possuem associações no Hibernate.
  - Podem estar associadas no banco de dados.

```
String hql = "from Aluno a, LogRecord log
              where a.id_aluno = log.id_aluno";

Query q = session.createQuery(hql);
Iterator it = q.list.iterator();

while ( i.hasNext() ) {
    Object[] obj = (Object[]) i.next();
    Aluno aluno = (Aluno) obj[0];
    LogRecord log = (LogRecord) obj[1];
}
```

HQL – Paginação

- Consulta recupera do banco apenas as linhas dentro de uma faixa de resultados

```
Query q = session.createQuery("from Aluno");
q.setFirstResult(10);
q.setMaxResults(30);
q.list();
```

HQL – Nomeação de Parâmetros

- Possibilidade de nomear parâmetros para definí-los posteriormente

```
Query q = createQuery("from Aluno a
                                where a.nome = :nome");
q.setString("nome", "João");
List result = q.list();
```

HQL – Nomeação de Parâmetros

```
Query q = createQuery(
        "from Empregado e
                where e.empresa.tipo = :tipo and
                e.salario > :salario and
                e.idade > e.esposa.idade");

q.setInteger("tipo", 1);

q.setDouble("salario", 1500);

List result = q.list();
```

HQL – Projeções

```
String hql = "select c.id, c.nome, a.id, a.nome
        from Curso c, Aluno a where a.id_curso = c.id_curso";

Query q = session.createQuery(hql);
ArrayList resultado = new ArrayList();
Iterator it = q.list.iterator();
while ( i.hasNext() ) {
    Object[] obj = (Object[]) i.next();

    Curso curso = new Curso();
    curso.setId((Integer)obj[0]);
    curso.setNome((String)obj[1]);

    Aluno aluno = new Aluno();
    aluno.setId((Integer)obj[2]);
    aluno.setNome((String)obj[3]);

    aluno.setCurso(curso);
    resultado.add(aluno);
}
```

HQL – Atualizações

```
//...
Transaction tx = session.beginTransaction();
Query q = createQuery(
          "update Curso set nome = :novoNome "
        + "where id = :idCurso");
q.setString("novoNome", "Novo Nome");
q.setInteger("idCurso", 150);

q.executeUpdate();
tx.commit();
//...
```

HQL – Atualizações

```
//...
Transaction tx = session.beginTransaction();
Query q = createQuery(
        "delete from Aluno where curso.id = :idCurso");
q.setInteger("idCurso", 150);


q.executeUpdate();


tx.commit();
//...
```

Exercícios

- VeterinarioDAO:
  - Todos os veterinários da clínica
    - `public Collection<Veterinario> findAll(){...}`
  - Total de consultas já realizadas por um veterinário
    - `public int findTotalConsultas(int idVeterinario){...}`

Exercícios

- ConsultaDAO:
  - Todas as consultas de um veterinário e um animal:
    - `public Collection<Consulta> findByVeterinarioAnimal(int idVeterinario, int idAnimal){...}`
  - Todas as consultas de um veterinário e um animal em determinado período:
    - `public Collection<Consulta> findByVeterinarioAnimal(int idVeterinario, int idAnimal, Date dataInicial, Date dataFinal){...}`

Exercícios

- ConsultaDAO:
  - Total de consultas agrupadas por veterinário:
    - `public Collection<Veterinario> findTotalGroupByVeterinarios(){...}`
    - `Dicas: inclua atributo transiente em Veterinario para armazenar total.`

Exercícios

- TratamentoDAO:
  - Todos os tratamentos já realizados por um animal:
    - `public Collection<Tratamento> findByAnimal(int idAnimal){...}`
  - Todos os tratamentos em animais com idade entre duas informadas
    - `public Collection<Tratamento> findByIdades(int idade1, int idade2){...}`

Exercícios

- ClienteDAO:
  - Clientes com as maiores pontuações. Total de clientes a serem exibidos deve ser informado:
    - `public Collection<Cliente> findByMaioresPontuacoes(int totalClientes){...}`
    - `Dica: Usar setMaxResults()`

Consultas por Critérios

Interface Criteria

Criteria

- Permite construir queries pela manipulação de critérios em tempo de execução;
- Uso de objetos ao invés de *strings*;
- Mais Orientado a Objetos;
- Menos legível.

Criteria

Todas as Empresas

```
Criteria c = session.createCriteria(Empresa.class);
ArrayList<Empresa> empresas =
                        (ArrayList<Empresa>) c.list();
```

Todas as empresas com nome E1 e que endereço comece com A

```
Criteria c = session.createCriteria(Empresa.class);
c.add(Expression.eq("nome", "E1"));
c.add(Expression.like("endereco", "A%"));
ArrayList<Empresa> empresas =
                        (ArrayList<Empresa>) c.list();
```

Criteria

Todos os projetos com status 2 e que a data de cadastro não é igual a data atual

```
Criteria c = session.createCriteria(Projeto.class);
c.add(Expression.eq("status", 2));
c.add( Expression.not
          (
              Expression.eq("dataCadastro", new Date())
           )
      );
ArrayList<Projeto> projetos =
                        (ArrayList<Projeto>) c.list();
```

Criteria

Todos os projetos que o status é diferente de 2

```
Criteria c = session.createCriteria(Projeto.class);
c.add(Expression.ne("status", 2));
ArrayList<Projeto> projetos =
                        (ArrayList<Projeto>) c.list();
```

Criteria

Todos os funcionários sem data de nascimento ordenados Por nome

```
Criteria c = session.createCriteria(Funcionario.class);
c.add(Expression.isNull("dataNascimento"));
c.add(Order.asc("nome"));
ArrayList<Funcionario> funcionarios =
                        (ArrayList<Funcionario>) c.list();
```

Todos os projetos com data de finalização definida

```
Criteria c = session.createCriteria(Projeto.class);
c.add(Expression.isNotNull("dataFinalizacao"));
ArrayList<Projeto> projetos =
                        (ArrayList<Projeto>) c.list();
```

Criteria

Todos os funcionários com data de nascimento maior ou igual a data1 e menor ou igual a data2. Ordenados por data de Nascimento e ordem decrescente

```
//Date data1 = ..., Date data2 = ...
Criteria c = session.createCriteria(Funcionario.class);
c.add(Expression.ge("dataNascimento", data1));
c.add(Expression.le("dataNascimento", data2));
c.add(Order.desc("dataNascimento"));
ArrayList<Funcionario> funcionarios =
                        (ArrayList<Funcionario>) c.list();
```

Criteria

Projeto com valor de id igual a 23. Como é chave primária, Apenas retorna um elemento.

```
Criteria c = session.createCriteria(Projeto.class);
c.add(Expression.eq("id", 123));
Projeto projeto = (Projeto) c.uniqueResult();
```

Projetos status pertencentes aos seguintes valores:1, 2 e 3

```
Criteria c = session.createCriteria(Projeto.class);
c.add(Expression.in("status", new Integer[]{1, 2, 3}));
ArrayList<Projeto> projetos =
                    (ArrayList<Projeto>) c.list();
```

Criteria

Funcionários com data de nascimento menor do que data1 ou maior do que data2

```
//Date data1 = ..., Date data2 = ...
Criteria c = session.createCriteria(Funcionario.class);
c.add(Expression.or(
        Expression.lt("dataNascimento", data1),
        Expression.gt("dataNascimento", data2))
);
ArrayList<Empresa> empresas =
                        (ArrayList<Empresa>) c.list();
```

Criteria – Associações

Funcionários com empresa de id igual a 23

```
Criteria cFunc =
                session.createCriteria(Funcionario.class);
Criteria cEmp = cFunc.createCriteria("empresa");
cEmp.eq("id", 123);
ArrayList<Funcionario> funcionarios =
                        (ArrayList<Funcionario>) c.list();
```

Criteria – Associações

Projetos com data de cadastro menor do que a atual e que tenha engenheiros da empresa de id = 123

```
Criteria cProj = session.createCriteria(Projeto.class);
cProj.add(Expression.lt("dataCadastro", new Date()));

Criteria cEng = cProj.createCriteria("engenheiro");
Criteria cEmp = cEng.createCriteria("empresa");
cEmp.eq("id", 123);
ArrayList<Projeto> projetos =
                        (ArrayList<Projeto>) c.list();
```

Criteria – Paginação

Retorna do décimo ao trigésimo aluno considerando eles em ordem crescente de matrícula

```
Criteria criteria =
                session.createCriteria(Aluno.class);
Criteria.add(Order.asc("matricula"));

criteria.setFirstResult(10);
criteria.setMaxResults(30);

ArrayList<Aluno> alunos =
                (ArrayList<Aluno>) criteria.list();
```

Criteria - Consultas Por Exemplos

- Criação de um critério com base num objeto de domínio semi-preenchido.

```
Funcionario f = new Funcionario();
f.setNome("Nicolau");
f.setRenda(25000);

Criteria c = session.createCriteria(Funcionario.class);
c.add( Example.create( f ));
c.list()
```

Criteria - Consultas Por Exemplos

- Entre objetos associados:

```
Criteria c = session.createCriteria(Funcionario.class) ;
c.add( Example.create(f) );

Criteria cM = c.createCriteria("empresa");
cM.add( Example.create( f.getEmpresa() ) ) ;

c.list();
```

Criteria – Projeção

```
String campos[] = {"id", "status", "dataCadastro"};

Criteria criteria = session.createCriteria(Projeto.class);

//Seto as propriedades que quero carregar
ProjectionList p = Projections.projectionList().create();

for (int i = 0; i < campos.length; i++){
      p.add(Projections.property(campos[i]), campos[i]);
}
criteria.setProjection(p);
criteria.setResultTransformer(
        new AliasToBeanResultTransformer(Projeto.class));

List result = criteria.list();
```

Criteria – Projeção

Todos os cursos associados a polos. Apenas cursos distintos serão retornados

```
Criteria c = sessioncreateCriteria(Polo.class);
c.setProjection(
        Projections.distinct(Projections.property("curso"))
);
return c.list();
```

```
Classe Polo tem um atributo Curso curso
```

Criteria – Projeção

Todas as turmas de solicitações de matrículas de Determinado discente

```
Criteria c =
session.createCriteria(SolicitacaoMatricula.class);
c.add( Expression.eq("discente", discente) );

c.setProjection(Projections.property("turma"));

return c.list();
```

```
Classe SolicitacaoMatricula tem um atributo Turma turma
```

Criteria – Projeção, Agregação, group by

JavaRN
Grupo de Usuários Java do Rio Grande do Norte

```
Criteria criteria = session.createCriteria(Projeto.class);

//Seto as propriedades que quero carregar
ProjectionList p = Projections.projectionList().create();
p.add( Projetions.rowCount() );
p.add( Projetions.avg( "valor" ) );
p.add( Projetions.max( "valor" ) );
p.add( Projetions.groupProperty( "id" ) );

criteria.setProjection(p);

List = criteria.list();
```

UnP J2EEBRASIL

Criteria – Projeção, Agregação, group by

JAVARN
Grupo de Usuários Java do Rio Grande do Norte

```
Criteria c = session.createCriteria(MatriculaComponente.class);

c.add(Expression.eq("turma", new Turma(idTurma)));
c.add(
        Expression.or(
                Expression.eq(
                        "situacaoMatricula", SitMatricula.EM_ESPERA),
                Expression.eq(
                        "situacaoMatricula", SitMatricula.MATRICULADO)
        )
);

c.setProjection(Projections.sum("id"));
Integer total = (Integer) c.uniqueResult();
```

UnP J2EEBRASIL

Exercício

- AnimalDAO:
  - Todos os animais de determinado espécie, ordenados por nome:
    - `public Collection<Animal> findByEspecie(int idEspecie){...}`
  - Todos os animais com determinado sexo e idate:
    - `public Collection<Animal> findBySexoIdade(String sexo, int idade){...}`

Exercício

- TratamentoDAO:
  - Todos os tratamentos com valor acima de um especificado
    - `public Collection<Tratamento> findByValor(double valor){...}`
  - Todos os tratamentos realizados em determinado período
    - `public Collection<Tratamento> findByPeriodo(Date dataInicial, Date dataFinal){...}`