

Persistência de Objetos

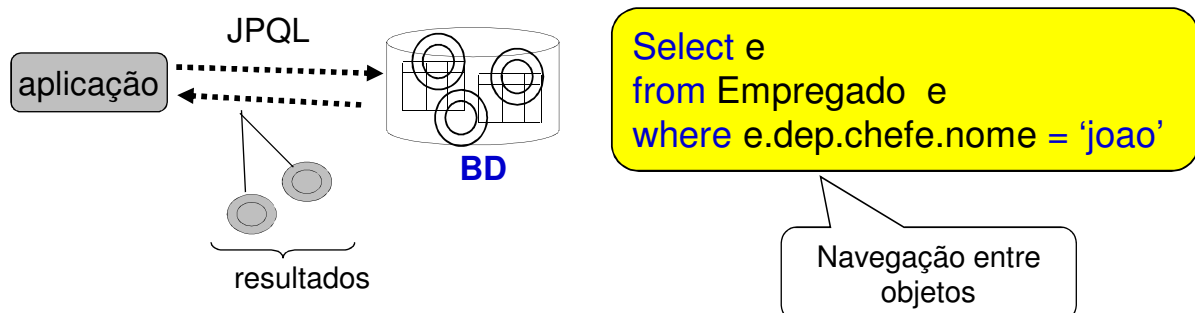
Fausto Maranhão Ayres

10 Persistência com JPA (JPQL)

JPQL - Java Persistence Query Language

- Linguagem Declarativa Orientada a Objetos
SELECT objetos
FROM coleção
WHERE condição
- O BD Relacional é tratado como um **BDOO**. O **esquema** utilizado nas consultas são classes e não tabelas

Ex:



PARTE 1: Submissão de consultas JPQL

Submetendo consulta em Java (Query)

- recuperando vários Objetos

```
Query q = manager.createQuery(  
    "select e from Empregado e");
```

*retorna
List<Object>*

```
List<Empregado> empregados = q.getResultList();
```

- recuperando um único Objeto

```
Query q = manager.createQuery(  
    "select e from Empregado e where e.id = 100");
```

```
Empregado emp = (Empregado) q.getSingleResult();
```

*retorna
Object*

Consultas tipadas (TypedQuery)

- Pode-se tipar o resultado de uma consulta com TypedQuery.

```
TypedQuery<Empregado> q = manager.createQuery(  
    "select e from Empregado e", Empregado.class);
```

*Tipo do
resultado*

```
List<Empregado> empregados = q.getResultList();
```

Exceções

getSingleResult()



javax.persistence.NoResultException
javax.persistence.NoneUniqueResultException
java.lang.IllegalStateException

getResultList()



java.lang.IllegalStateException

limitando Resultados

```
Query q = manager.createQuery(
    "select e from Empregado e");

List<Empregado> empregados = q.setMaxResults(20)
    .setFirstResult(10)
    .getResultList();
```

Consultas Parametrizadas

- Uso de parâmetros por Nomeação

```
Query q = manager.createQuery(
    "select e from Empregado e where e.nome like :n");
q.setParameter("n", "joao%");

List<Empregado> empregados = q.getResultList();
```

- Uso de parâmetros por Posição

```
Query q = manager.createQuery(
    "select e from Empregado e where e.salario > ?1");
q.setParameter(1, 1000.0);

List<Empregado> empregados = q.getResultList();
```

Obs: Pode-se passar objeto como parâmetro

Consultas Parametrizadas (data)

```
public class Empregado {  
    ...  
    private LocalDate admissão = ....  
}  
  
Query q = manager.createQuery(  
    "select e from Empregado e where e.admissao > ?1");  
  
LocalDate dataqualquer = LocalDate.of(2000,12,30);  
q.setParameter(1, dataqualquer);  
  
List<Empregado> empregados = q.getResultList();
```

Consultando um atributo


```
Query q = manager.createQuery(  
    "select e.nome from Empregado e");  
  
List<String> nomes = q.getResultList();  
  
Query q = manager.createQuery(  
    "select e.id from Empregado e" );  
  
List<Integer> matriculas = q.getResultList();
```

Consultando vários atributos

```
Query q = manager.createQuery(
    "select e.nome, e.salario from Empregado e");

List<Object[]> lista = q.getResultList();

for (int i =0; i<lista.size(); i++) {
    Object[] resultado = (Object[]) lista.get(i);
    String nome = (String) resultado [0];
    double salario = (Double) resultado [1];
    System.out.println(nome + salario);
}
```



Consultas em SQL

```
Query q = manager.createNativeQuery(
    "select * from Empregado where e.salario>1000",
    Empregado.class
)
```

Atualizações em “Lote” no banco

- São alterações executadas num processo (thread) em separado (não é garantida a ordem de execução das outras operações)

```
manager.getTransaction().begin();

Query q = manager.createQuery(
    "delete from Departamento d where d.empregados is
    empty");

int linhas = q.executeUpdate();

manager.getTransaction().commit();
```

Obs:

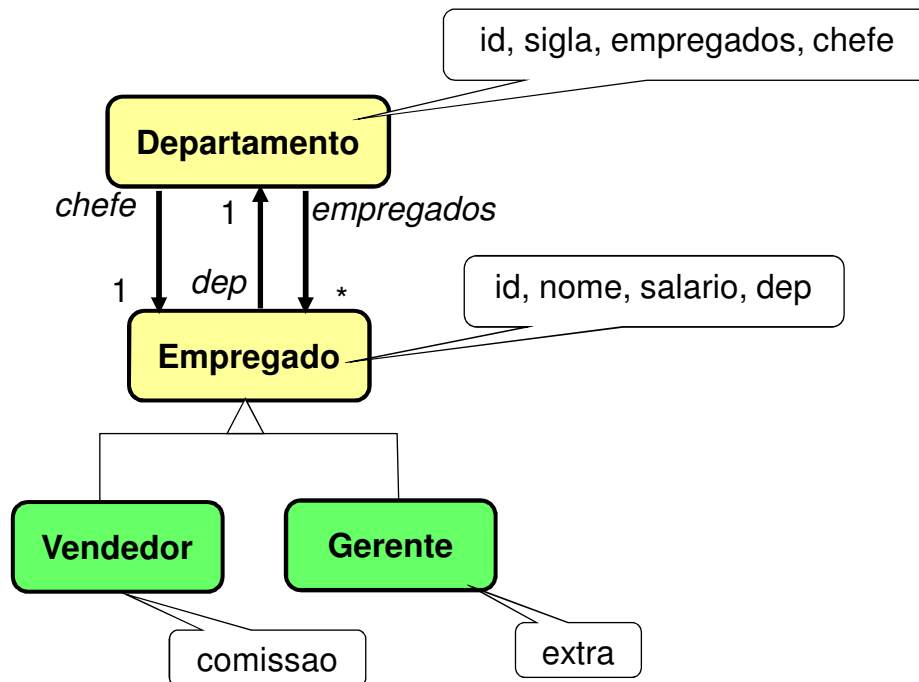
linhas = quantidade de objetos atualizados

fausto.ayres@ifpb.edu.br

13

PARTE 2: Sintaxe JPQL

Modelo usado como exemplo



fausto.ayres@ifpb.edu.br

15

JPQL

■ Palavras Reservadas (*case insensitive*)

SELECT, FROM, WHERE, UPDATE, DELETE, JOIN,
OUTER, INNER, LEFT, GROUP, BY, HAVING, **FETCH**,
DISTINCT, **OBJECT**, NULL, TRUE, FALSE, NOT, AND,
OR, BETWEEN, LIKE, IN, AS, **EMPTY**, **MEMBER**, **OF**,
NEW, **IS**, ORDER, BY, ASC, DESC, EXISTS, ALL, ANY,
SOME, AVG, MAX, MIN, SUM, COUNT

MOD, **UPPER**, **LOWER**, **TRIM**, **POSITION**,
CHARACTER_LENGTH, **CHAR_LENGTH**, **BIT_LENGTH**,
CURRENT_TIME, **CURRENT_DATE**, **CURRENT_TIMESTAMP**

fausto.ayres@ifpb.edu.br

16

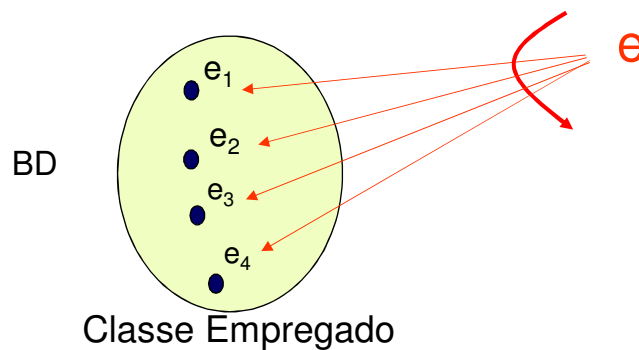
Variável de iteração

- Um Cursor sobre uma coleção de objetos, similar a variável de iteração do *for-each* java

```
select e from Empregado e
```

Classe

Variável de iteração



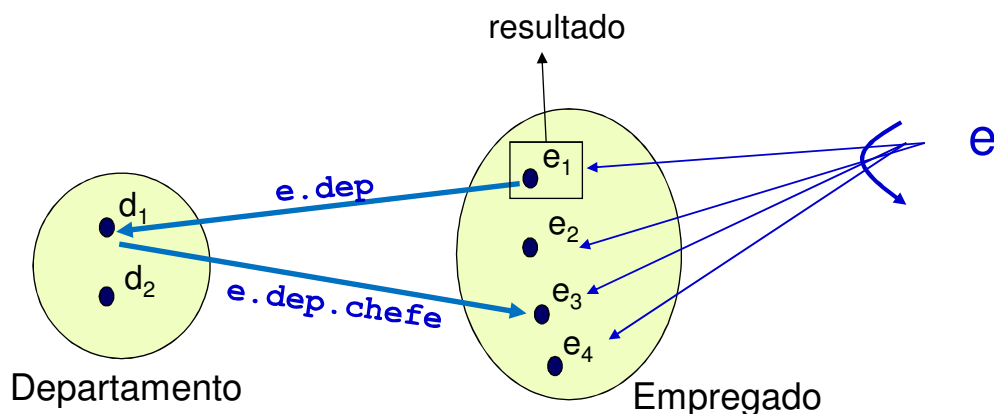
fausto.ayres@ifpb.edu.br

17

Navegação pelo relacionamento 1:1

- Quais *empregados* que *ganham mais que seu chefe*?

```
select e
from Empregado e
where e.salario > e.dep.chefe.salario
```



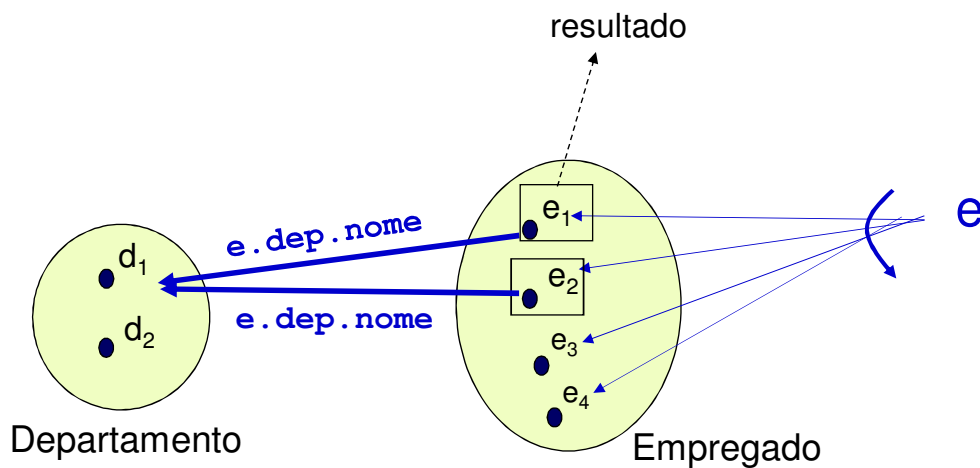
fausto.ayres@ifpb.edu.br

18

Navegação pelo relacionamento 1:1

Os empregados do departamento de eletrônica?

```
select e
from Empregado e
where e.dep.nome = 'eletronica'
```



19

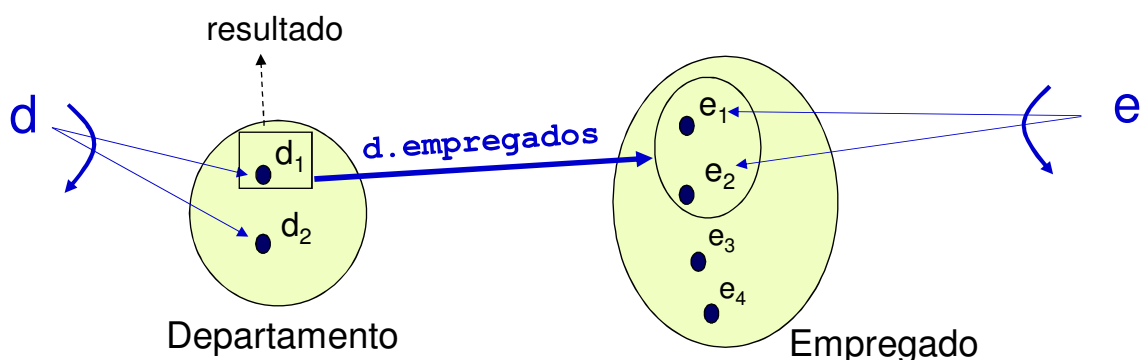
Navegação por relacionamentos 1:* (JOIN)

Os departamentos que possuem empregados com salários > 1000?

```
select d
from Departamento d JOIN d.empregados e
where e.salario > 1000
```

2 cursores:

Para cada objeto d existem vários objetos e



IN = JOIN

- Pode-se usar IN() no lugar de JOIN

```
select d
from Departamento d, IN(d.empregados) e
where e.salario > 1000
```

NULL

- Referências Nulas

Exemplo: *empregados sem departamento*

```
select e from Empregado e
where e.dep IS NULL           // <> NULL
```

EMPTY, MEMBER

- Coleções Vazias: **IS [NOT] EMPTY**

Exemplo: *departamentos sem empregados*

```
select d from Departamento d
where d.empregados IS EMPTY
```

- Pertinência: **[NOT] MEMBER OF**

- Exemplo: *departamentos que NÃO contem o empregado x*

```
select d from Departamento d
where :x NOT MEMBER OF d.empregados
```

Funções de Agregação

Exemplo:

```
select count(e)
from Empregado e
```

```
select count(e)
from departamentod JOIN d.empregados e
Where d.nome = 'eletrônica'
```

```
select min(e.salario), max(e.salario),
       sum(e.salario), avg(e.salario)
from Empregado e
```

ORDER BY, GROUP BY e HAVING

Exemplo:

- *A media salarial dos departamentos que tem media acima de R\$ 2000,00*

```
SELECT  d.nome, AVG(e.salario) as media
FROM    Departamento d JOIN d.employees e
GROUP BY d.nome
HAVING  AVG(e.salario) > 2000.0
ORDER BY d.nome
```

Sub-consultas

Ex: *Quais os empregados que ganham mais que o joao?*

```
select e1 from Empregado e1
where e1.salario >
      (select e2.salario from Empregado e2
       where e2.nome='joao')
```

Funções internas

Function	Applicability
UPPER(s), LOWER(s)	String values; returns a string value
CONCAT(s1, s2)	String values; returns a string value
SUBSTRING(s, offset, length)	String values (offset starts at 1); returns a string value
TRIM([[BOTH LEADING TRAILING] char [FROM]] s)	Trims spaces on BOTH sides of s if no char or other specification is given; returns a string value
LENGTH(s)	String value; returns a numeric value
LOCATE(search, s, offset)	Searches for position of ss in s starting at offset; returns a numeric value
ABS(n), SQRT(n), MOD(dividend, divisor)	Numeric values; returns an absolute of same type as input, square root as double, and the remainder of a division as an integer
SIZE(c)	Collection expressions; returns an integer, or 0 if empty

Exemplos

```
select u from User u where concat(u.name, 's') = 'Walters'

select u from User u where substring(u.name, 1, 1) = 'W'

select u from User u where trim(leading 'a' FROM u.name) = 'W'

select u from User u where lower(u.name) = 'walter'

select u from User u where upper(u.name) = 'WALTER'

select u from User u where length(u.name) = 6

select u from User u where locate('a', u.name) = 2

select u from User u where abs(u.age) >= 5.00


select u from User u where sqrt(u.age) >= 1000

select u from User u where mod(u.age, 10) = 0
```

Funções internas

```
select d from Departamento d where SIZE(d.empregados) > 0
```

```
select p from Produto p where p.validade > CURRENT_DATE()
```



CURRENT_DATE()
CURRENT_TIME()
CURRENT_TIMESTAMP()

Consultas Polimórficas

```
select e from Empregado e
```

TODOS
empregados,
vendedores e gerentes

```
select v from Vendedor v
```

só vendedores

```
select g from Gerente g
```

só gerentes

Consultas Polimórficas Restritas

```
select e from Empregado e
where Type(e) = Empregado
```

Só empregados

```
select e from Empregado e
where Type(e) IN (Empregado, Gerente)
```

empregados ou
gerentes

```
select e from Empregado e
where e.salarario > 1000 or
      Treat(e as Gerente).gratificacao > 200
```

Casting de
objeto

fausto.ayres@ifpb.edu.br

31

UPDATE e DELETE

■ Exemplos

```
UPDATE Empregado e
  set e.salarario = e.salarario * 1.10
  where e.dep.nome = 'informatica'
```

```
DELETE from Empregado e
  where e.dep.nome = 'informatica'
```

```
DELETE from Departamento d
  where size(d.empregados) = 0
      // d.empregados is empty
```

fausto.ayres@ifpb.edu.br

32

Comparativo entre JPQL e OQL

JPQL (2007)

```
select e  
from Empregado e  
where e.salario > e.dep.chefe.salario
```

```
select d  
from Departamento d JOIN d.empregados e  
where e.salario>1000
```

```
select d  
from Departamento d  
where SIZE(d.contratados)>2
```

OQL (1980)

```
select e  
from e IN Empregado  
where e.salario > e.dep.chefe.salario
```

```
select d  
from d IN Departamento d, e IN d.empregados  
where e.salario>1000
```

```
select d  
from d IN Departamento  
where COUNT(d.contratados)>2
```

Novidades JPQL

Novidades JPQL

- O método **getResultStream()** permite obter os resultados sob demanda ao invés de obtê-los todos de uma só vez, como acontece com o **getResultList()**

Exemplos:

```
Stream<Autor> autores = manager.createQuery("SELECT a  
FROM Autor a", Autor.class)  
.getResultStream();
```

```
Stream<Autor> autores = manager.createQuery("SELECT a  
FROM Autor a", Autor.class)  
.setMaxResults(5)  
.setFirstResult(10)  
.getResultStream();
```

fausto.ayres@ifpb.edu.br

35

Novidades JPQL

Feature Name	Description	Example
Date, time, and timestamp literals	JDBC syntax was adopted: {d 'yyyy-mm-dd'} {t 'hh-mm-ss'} {ts 'yyyy-mm-dd hh-mm-ss'}	SELECT c FROM Customer c WHERE c.birthdate < {d '1946-01-01'}
Non-polymorphic queries – TYPE	Can query across specific subclasses of a superclass	SELECT p FROM Project p WHERE TYPE(p) = DesignProject OR TYPE(p) = QualityProject
Map support - KEY, VALUE, ENTRY	Allow comparison and selection of keys and values and selection of entries	SELECT e.name, KEY(p), VALUE(p) FROM Employee e JOIN e.phones p WHERE KEY(p) IN ('Work', 'Cell')
Collection input parameters	Allow parameter arguments to be collections	SELECT e FROM Employee e WHERE e.lastName IN :names
CASE statement	Can be in either of two forms: 1) CASE {WHEN conditional THEN scalarExpr}+ ELSE scalarExpr END 2) CASE pathExpr {WHEN scalarExpr THEN scalarExpr}+ ELSE scalarExpr END	UPDATE Employee e SET e.salary = CASE WHEN e.rating = 1 THEN e.salary * 1.1 WHEN e.rating = 2 THEN e.salary * 1.05 ELSE e.salary * 1.01 END

36

Novidades JPQL

NULLIF, COALESCE	Additional CASE variants: COALESCE(scalarExpr {, scalarExpr}+) NULLIF(scalarExpr, scalarExpr)	SELECT COALESCE(d.name, d.id) FROM Department d
Scalar expressions in the SELECT clause	Return the result of performing a scalar operation on a selected term	SELECT LENGTH(e.name) FROM Employee e
INDEX in a List	Refer to an item's position index in a list	SELECT p FROM Flight f JOIN f.upgradeList p WHERE f.num = 861 AND INDEX(p) = 0
Variables in SELECT constructors	Constructors in SELECT clause can contain identification vars	SELECT new CustInfo(c.name, a) FROM Customer c JOIN c.address a