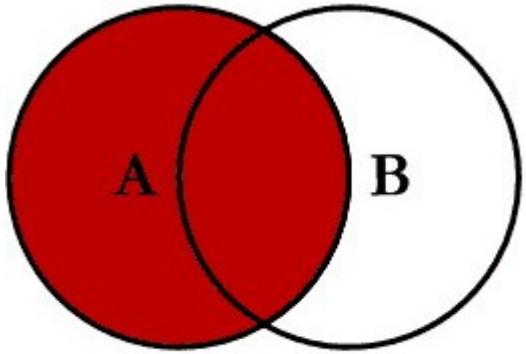
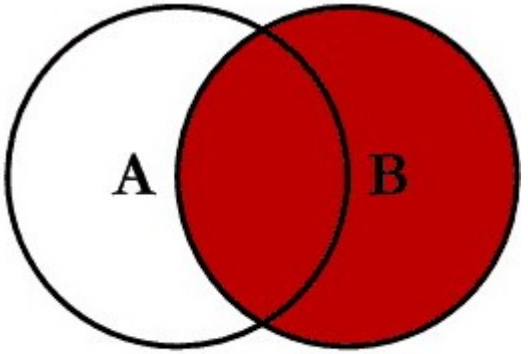
Persistência com JDBC e JPA




```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

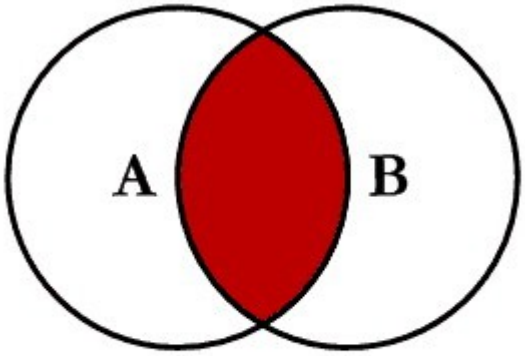
Aula 5




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```

Marcos Alberto Lopes da Silva

(malopes21@gmail.com)




```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

JPA QL

- Dialeto POO de SQL;
- JPA QL é um subconjunto padronizado de HQL;
- Utilizado quase sempre para recuperação de objetos;
- Permite expressar quase tudo o que pode ser expresso em SQL, com classes e objetos;
- Interface ***javax.persistence.Query***;

JPA QL

- Pode-se navegar pelo grafo de objetos dentro da query;
- Efetuar projeção;
- Ordenar e paginar os resultados;
- Agregar com group by, having e funções de agregação (sum, min, max);
- Efetuar junções externas;
- Possui capacidade de chamar funções SQL padronizadas;
- Efetuar consultas aninhadas

Criteria

- Permite especificar restrições dinamicamente sem manipulações de caracteres;
- Normalmente é preferida pelos desenvolvedores, por ser mais OO (sintaxe analisada e validada em tempo de compilação);
- Padronizada a partir do JPA 2.0;
- Suporta QBE (*query by example);*
- *Interface javax.persistence.criteria.CriteriaBuilder*

Criteria vs JPA-QL

- JPAQL é mais flexível pela sua natureza baseada em sequência de caracteres;
- Criteria costuma ser a solução preferida para consultas mais complexas, especialmente as dinâmicas;

Consultas básicas

- Projeto – **PosJava-Persistencia-Aula05**
- Classe – **ConsultasBasicas**
- Utilização de JPA-QL, Criteria e query nativo

Consultas básicas – persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
     xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
     xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
     <persistence-unit name="PosJava-ProjetoJPA-Aula05">
          <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
          <!-- <provider>org.hibernate.ejb.HibernatePersistence</provider> -->
          <!-- <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider> -->
          <class>domain.Produto</class>
          <class>domain.Fornecedor</class>
          <properties>
               <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost:3306/posjava_aula05" />
               <property name="javax.persistence.jdbc.user" value="root" />
               <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver" />
               <property name="javax.persistence.jdbc.password" value="" />
               <property name="javax.persistence.schema-generation.database.action"
                    value="create" />

               <property name="show_sql" value="true"/>
               <property name="format_sql" value="true"/>
               <property name="use_sql_comments" value="true"/>

               <property name="eclipselink.weaving" value="static" />
               <property name="eclipselink.logging.level.sql" value="FINEST" />
               <property name="eclipselink.logging.level" value="FINEST" />
               <property name="eclipselink.logging.level.cache" value="FINEST" />
          </properties>
     </persistence-unit>
</persistence>
```

Consultas básicas – tabelas

```java
@Entity
@Table(name="tbFornecedor")
public class Fornecedor implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="CODFRN")
    private Integer codigo;

    @Column(name="DESFRN",nullable=false,length=255)
    private String nome;
}
```

```
mysql> describe tbfornecedor;
+--------+--------------+------+-----+---------+----
| Field  | Type         | Null | Key | Default | Ext
+--------+--------------+------+-----+---------+----
| CODFRN | int(11)      | NO   | PRI | NULL    | aut
| DESFRN | varchar(255) | NO   |     | NULL    |
+--------+--------------+------+-----+---------+----
```

```java
@Entity
@Table(name = "tbProduto")
public class Produto {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "CODPRO")
    private Integer codigo;

    @Column(name = "DSCPRO")
    private String descricao;

    @ManyToOne
    @JoinColumn(name = "CODFRN", nullable = false, insertable = true, updatable = true)
    private Fornecedor fornecedor;
}
```

```
mysql> describe tbproduto;
+--------+--------------+------+-----+---------+-----
| Field  | Type         | Null | Key | Default | Extr
+--------+--------------+------+-----+---------+-----
| CODPRO | int(11)      | NO   | PRI | NULL    | auto
| DSCPRO | varchar(255) | YES  |     | NULL    |
| CODFRN | int(11)      | NO   | MUL | NULL    |
+--------+--------------+------+-----+---------+-----
```

Consultas básicas – popula tabelas

```
public class PopulaDados {
    public static void main(String args[]) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("PosJava-
ProjetoJPA-Aula05");
        EntityManager em = emf.createEntityManager();
        populaFornecedores(em);
    }

    private static void populaFornecedores(EntityManager em) {
        Fornecedor f1 = new Fornecedor("Alpargatas");
        Fornecedor f2 = new Fornecedor("Unilever");
        Fornecedor f3 = new Fornecedor("Sadia");
        Fornecedor f4 = new Fornecedor("Perdigão");
        em.getTransaction().begin();
        em.persist(f1);
        em.persist(f2);
        em.persist(f3);
        em.persist(f4);

        em.persist(new Produto("havaianas top", f1));
        em.persist(new Produto("havaianas color", f1));
        em.persist(new Produto("linguiça toscana", f3));
        em.getTransaction().commit();
    }
}
```

Consultas básicas – consultas

```
public class ConsultasBasicas {
    public static void main(String args[]) {
        EntityManagerFactory emf = Persistence
            .createEntityManagerFactory("PosJava-ProjetoJPA-Aula05");
        EntityManager em = emf.createEntityManager();
        consultaJpaQl(em);     consultaCriteria(em);
        consultaSQL(em);       emf.close();
    }

    private static void consultaJpaQl(EntityManager em) {
        Query q = em.createQuery("Select f from Fornecedor f", Fornecedor.class);
        List<Fornecedor> retorno = q.getResultList();
        System.out.println("fornecedores em JPA_QL");
        retorno.forEach((e) -> System.out.println(e));
    }

    private static void consultaCriteria(EntityManager em) {
        CriteriaBuilder cb = em.getCriteriaBuilder();
        CriteriaQuery<Fornecedor> cq = cb.createQuery(Fornecedor.class);
        Root<Fornecedor> root = cq.from(Fornecedor.class);
        List<Fornecedor> retorno = em.createQuery(cq.select(root)).getResultList();
        System.out.println("fornecedores em Criteria");
        retorno.forEach((e) -> System.out.println(e));
    }

    private static void consultaSQL(EntityManager em) {
        Query q = em.createNativeQuery("Select * from tbFornecedor", Fornecedor.class);
        List<Fornecedor> retorno = q.getResultList();
        System.out.println("fornecedores em SQL");
        retorno.forEach((e) -> System.out.println(e));
    }
}
```

Consultas básicas - parâmetros

- Evitar Sql Injection
- Parâmetros posicionais ou nomeados

```
private static void consultaJpaQl_Par(EntityManager em) {
     Query q = em
               .createQuery("Select f from Fornecedor f where f.codigo = :codigo",
Fornecedor.class);
     List<Fornecedor> retorno = q.setParameter("codigo", 1).getResultList();
     System.out.println("fornecedor 1 em JPA_QL");
     retorno.forEach((e) -> System.out.println(e));

}

private static void consultaSQL_Par(EntityManager em) {
     Query q = em.createNativeQuery("Select * from tbFornecedor where CODFRN = ?",
              Fornecedor.class);
     List<Fornecedor> retorno = q.setParameter(1, 1).getResultList();
     System.out.println("fornecedor 1 em SQL");
     retorno.forEach((e) -> System.out.println(e));
}
```

Consultas básicas – consultas nomeadas

- Definir em alguma classe do modelo;
- Identificador único para cada consulta;
- Anotação NamedQueries e/ou NamedNativeQueries, etc

```
@NamedQueries({ @NamedQuery(name = "buscaProdutosPorFornecedorJpa",
    query = "from Produto p where p.fornecedor = :fornecedor") })
@NamedNativeQueries({ @NamedNativeQuery(name = "buscaProdutosPorFornecedorSql",
    query = "Select * from tbProduto where codfrn = ?", resultClass =
Produto.class) })
@Entity
@Table(name = "tbProduto")
public class Produto {
//..
}
```

Consultas básicas – consultas nomeadas

```
private static void consultaJpaQl(EntityManager em) {
    Fornecedor f = new Fornecedor();
    f.setCodigo(1);
    Query q = em.createNamedQuery("buscaProdutosPorFornecedorJpa",Produto.class);
    q.setParameter("fornecedor",f);
    List<Produto> retorno = q.getResultList();
    System.out.println("produtos em JPA-QL");
    for(Produto p: retorno) {
        System.out.println(p);
    }
}

private static void consultaSQL(EntityManager em) {
    Fornecedor f = new Fornecedor();
    f.setCodigo(1);
    Query q = em.createNamedQuery("buscaProdutosPorFornecedorSql",Produto.class);
    q.setParameter(1, f.getCodigo());
    List<Produto> retorno = q.getResultList();
    System.out.println("produtos em SQL");
    for(Produto p: retorno) {
        System.out.println(p);
    }
}
```

TypedQuery e funções de agregação

Suponha que desejamos saber o maior código de uma listagem de fornecedores.

O resultado dessa consulta não deve ser uma lista, mas sim um valor numérico.

```
private static void listagem(EntityManager em) {
    TypedQuery<Integer> q = em
        .createQuery("select max(f.codigo) from Fornecedor f", Integer.class);
    Integer retorno = (Integer)q.getSingleResult();
    System.out.println("Maior Codigo fornecedor "+retorno);
}
```

AVG	Calcula a média de um conjunto de números
COUNT	Contabiliza o número de resultados
MAX	Recupera o maior elemento um conjunto de números
MIN	Recupera o menor elemento um conjunto de números
SUM	Calcula a soma de um conjunto de números

Resultados complexos

Algumas consultas possuem resultados complexos.

Suponha que desejamos obter uma listagem com os nomes dos produtos e o nome do seu fornecedor.

Nesse caso, o resultado será uma lista de array de Object.

```
private static void resultComplex(EntityManager em) {
    String jpaql = "Select p.descricao, p.fornecedor.nome  from Produto p";
    Query query = em.createQuery(jpaql);
    List<Object[]> lista = query.getResultList();
    for (Object[] tupla : lista) {
        System.out.println(" Produto : " + tupla[0]);
        System.out.println(" Fornecedor : " + tupla[1]);
    }
}
```

Resultados complexos - new

Para contornar a dificuldade de lidar com o posicionamento dos dados nos arrays,

podemos criar uma classe para modelar o resultado da nossa consulta e aplicar o operador new no código JPQL.

```
private static void resultComplex2(EntityManager em) {
    String jpaql = "Select new domain.ProdutoFornecedorVO(p.descricao,
p.fornecedor.nome)  from Produto p";
    TypedQuery<ProdutoFornecedorVO> query = em.createQuery(jpaql,
ProdutoFornecedorVO.class);
    List<ProdutoFornecedorVO> lista = query.getResultList();
    for (ProdutoFornecedorVO vo : lista) {
        System.out.println(" Produto : " + vo.getProduto());
        System.out.println(" Fornecedor : " + vo.getFornecedor());
    }
}
```

Paginação

Supondo que exista uma grande quantidade de produtos cadastrados no banco de dados, buscar todos vai sobrecarregar o tráfego da rede e a memória utilizada pela aplicação.

Nesses casos, podemos aplicar o conceito de paginação para obter os produtos aos poucos.

```
private static void paginacao(EntityManager em) {
    TypedQuery<Produto> q = em.createQuery("select p from Produto p", Produto.class);
    q.setFirstResult(1);
    q.setMaxResults(2);
    List<Produto> produtos = q.getResultList();
    produtos.forEach((p) -> System.out.println(p));
}
```

```
mysql> select * from tbproduto;
+--------+------------------+--------+
| CODPRO | DSCPRO           | CODFRN |
+--------+------------------+--------+
|      1 | linguiça toscana |      1 |
|      2 | havaianas top    |      4 |
|      3 | havaianas color  |      4 |
+--------+------------------+--------+
3 rows in set (0.00 sec)
```

O problema das N+1 consultas

Em alguns casos, o comportamento LAZY pode gerar um número excessivo de consultas, comprometendo o desempenho da aplicação.

Por exemplo, considere as entidades Fornecedor e Produto.

```
public class Fornecedor implements Serializable {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="CODFRN")
    private Integer codigo;

    @Column(name="DESFRN",nullable=false,length=255)
    private String nome;

    @OneToMany(mappedBy="fornecedor", fetch=FetchType.LAZY)
    private List<Produto> produtos;
}
```

O problema das N+1 consultas

Em muitos dos casos, quando buscamos uma lista de fornecedores, não precisamos dos dados dos seus produtos.

Assim, optamos pelo comportamento LAZY para o relacionamento.

No entanto, em alguns casos, estaremos sim interessados nos dados dos fornecedores e dos seus produtos.

Podemos realizar uma consulta para recuperar a lista de fornecedores.

Além disso, como o comportamento escolhido foi LAZY, uma consulta adicional será realizada pelo provedor para cada fornecedor a fim de recuperar os seus produtos.

O problema das N+1 consultas

```
private static void problMaisUm(EntityManager em) {
    TypedQuery<Fornecedor> query = em.createQuery("Select f from Fornecedor f",
Fornecedor.class);
    List<Fornecedor> fornecedors = query.getResultList();

    for(Fornecedor forn: fornecedors) {
        System.out.println("fornecedor: "+forn.getNome());
        for(Produto produto : forn.getProdutos()) {
            System.out.println(produto);
        }
    }
}
```

```
ss=Produto sql="SELECT CODPRO, DSCPRO, CODFRN FROM tbProduto WHERE (CODFRN = ?)")
nection pool [default].
Produto WHERE (CODFRN = ?)

tion pool [default].

ass=Produto sql="SELECT CODPRO, DSCPRO, CODFRN FROM tbProduto WHERE (CODFRN = ?)")
nection pool [default].
Produto WHERE (CODFRN = ?)

tion pool [default].

ass=Produto sql="SELECT CODPRO, DSCPRO, CODFRN FROM tbProduto WHERE (CODFRN = ?)")
```

O problema das N+1 consultas

Para solucionar esse problema, podemos utilizar o comando left join fetch na consulta que buscaria os fornecedores.

O uso de join fetch pode gerar duplicatas (usar distinct)!

```
private static void problMaisUmResolvido(EntityManager em) {
    //SELECT DISTINCT (d) FROM Departamento d LEFT JOIN FETCH d. funcionarios
    TypedQuery<Fornecedor> query = em.createQuery(
            "Select distinct(f) from Fornecedor f left join fetch f.produtos",
Fornecedor.class);
    List<Fornecedor> fornecedors = query.getResultList();

    for (Fornecedor forn : fornecedors) {
        System.out.println("fornecedor: " + forn.getNome());
        for (Produto produto : forn.getProdutos()) {
            System.out.println(produto);
        }
    }
}
```

```
1,5,main])--Connection acquired from connection pool [default].
|--SELECT DISTINCT t0.CODFRN, t0.DESFRN, t1.CODPRO, t1.DSCPRO, t1.CODFRN FROM tbFornecedor t0 LEFT OUTER JOIN tbProduto t1 ON (t1.CODFRN = t0
```

Algumas Funções padronizadas JPA QL

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

Comparações entre identificadores

- Referências do objeto podem ser comparadas diretamente;

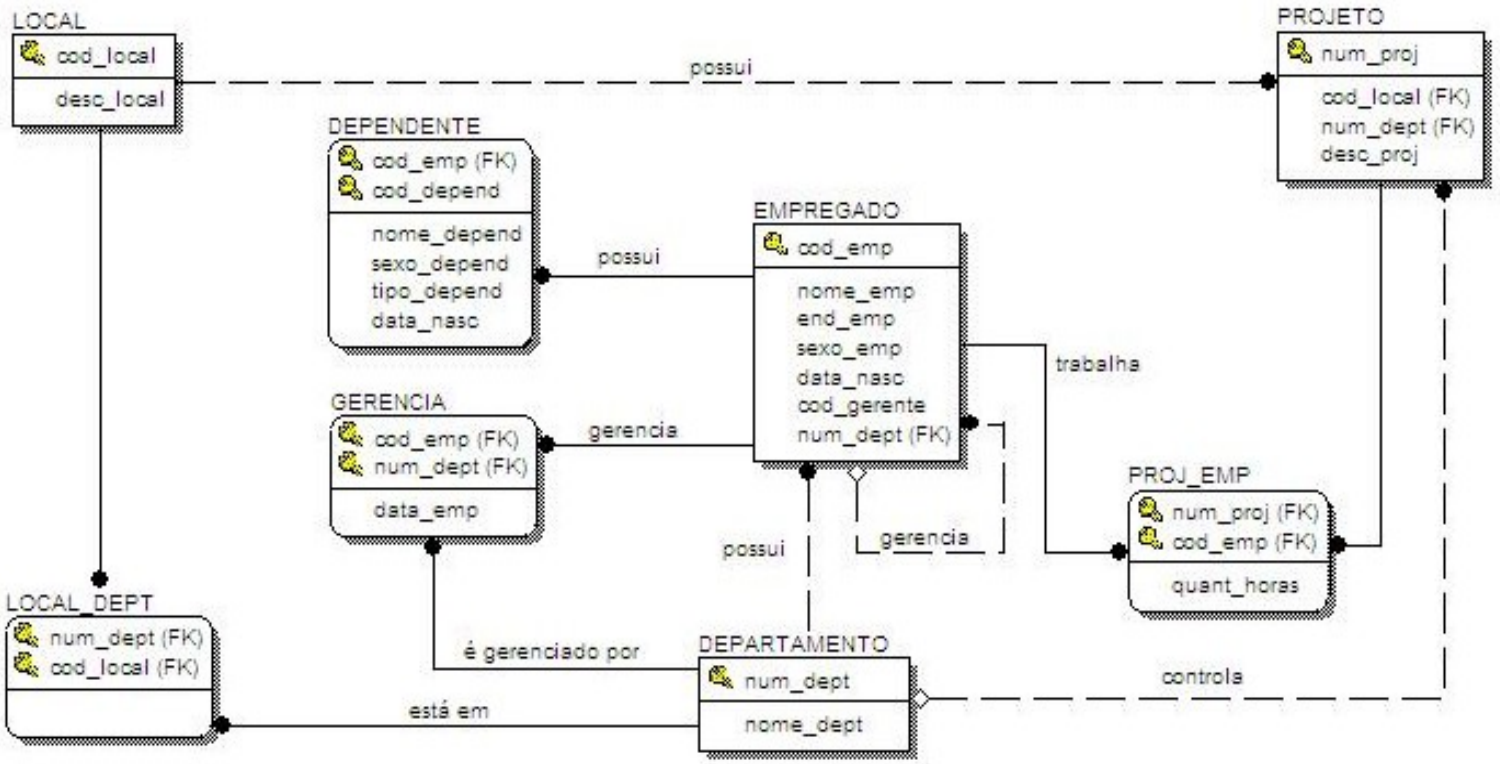
  ex:

  from Produto p, Fornecedor f where p.fornecedor = f

- Chaves estrangeiras podem ser implementadas como parâmetros de consulta

Exercício

Mapear o domínio ER abaixo para um domínio OO equivalente:



Exercício

Criar uma classe para polular os dados no BD (via JPA). Usar os mesmos valores abaixo:

```
select * from departamento;
```

Results | Messages

	num_dept	nome_dept
1	111	Financeiro
2	222	Administrativo
3	333	Pessoal
4	444	Compra

```
select * from local;
```

Results | Messages

	cod_local	desc_local
1	66	Ed. DiRoma - 10o. Andar
2	77	Ed. DiRoma - 1o. Andar
3	88	Ed. Central - 3o. andar
4	99	Ed. Central - 2o. andar

```
select * from dependente;
```

Results | Messages

	cod_emp	cod_depend	nome_depend	sexo_depend	tipo_depend	data_nasc
1	1	1	Camila	F	Filha	1988-10-10
2	1	2	Sergio	M	Filho	1990-02-20
3	3	1	Tais	F	Filha	1977-01-30
4	5	1	Fred	M	Filho	1992-08-30

```
select * from local_dept;
```

Results | Messages

	num_dept	cod_local
1	111	66
2	222	66
3	222	77
4	333	88
5	444	99

Exercício

Mais dados (via JPA):

```
select * from empregado;
```

Results | Messages

	cod_emp	nome_emp	end_emp	sexo_emp	data_nasc	cod_gerente	num_dept
1	1	Joao	Rua 10 nro 10	M	01/12/66	1	444
2	2	Maria	Rua 20 nro 20	F	02/13/65	3	111
3	3	Ana	Rua 30 nro 30	F	03/14/55	3	222
4	4	Carlos	Rua 40 nro 40	M	04/15/70	1	111
5	5	Silvio	Rua 50 nro 50	M	03/16/68	1	333
6	6	Amanda	Rua 60 nro 60	F	12/20/72	3	444
7	7	Emilia	Rua 70 nro 70	F	05/12/69	3	333
8	8	Jose	Rua 80 nro 80	M	08/13/76	3	444

```
select * from gerencia;
```

Results | Messages

	cod_emp	num_dept	data_emp
1	1	444	1982-01-01
2	3	222	1978-01-01
3	4	111	1980-01-01
4	5	333	1980-01-01

```
select * from proj_emp;
```

Results | Messages

	num_proj	cod_emp	quant_horas
1	10	2	40
2	10	4	20
3	20	4	20
4	30	1	30
5	30	6	35
6	30	8	30
7	40	7	40

```
select * from projeto;
```

Results | Messages

	num_proj	cod_local	num_dept	desc_proj
1	10	77	111	Pesquisa Economica
2	20	99	111	Treinamento
3	30	99	444	Sistema Cotacao
4	40	88	NULL	Avaliacao

Exercício

Criar outra classe com métodos (um pra cada) com consultas para as questões:

a) Quais os nomes dos departamentos existentes na empresa?

b) Quais são as informações de todos os empregados da empresa?

c) Qual o nome do departamento de cada empregado?

d) Qual o nome dos dependentes de cada empregado?

e) Qual o código dos locais de cada projeto em desenvolvimento?

f) Qual o código do departamento responsável por cada projeto?

g) Qual o nome do departamento responsável por cada projeto?

h) Qual a data que cada empregado começou a gerenciar o departamento? Apresente o nome do empregado, o nome do departamento e a data

Exercício

i) Quais os códigos dos empregados que trabalham no projeto de código 30?

j) Quais os nomes dos empregados que trabalham no projeto de código 20?

k) Qual o nome do departamento do empregado de código 7?

l) Qual o nome do departamento que controla o projeto onde o empregado de código 4 trabalha?

m) Qual o nome dos empregados que trabalham em algum projeto ?

n) Qual o nome dos empregados que trabalham em algum projeto e que supervisionam outros empregados?

o) Qual o nome dos empregados que não estão trabalhando em projetos?

p) Qual o nome dos empregados que não estão trabalhando em projetos, mas que gerenciam algum departamento?

q) Qual a descrição da ou das localizações do departamento de código 222?

Exercício

r) Qual a quantidade de horas trabalhadas pelo empregado de código 2?

s) Qual a quantidade de horas trabalhadas pelo empregado de código 4?

t) Qual a quantidade de horas trabalhadas pela empregada de nome Emília?

u) Qual a média de horas trabalhadas pelos empregados desta empresa?

v) Qual a quantidade de empregados existentes na empresa?

x) Quais os nomes dos dependentes de todas as empregadas da empresa (apenas empregadas)?

y) Qual o nome dos projetos que não são controlados por algum departamento?

z) Qual o nome dos departamentos que estão localizados em dois ou mais locais diferentes?

Referências bibliográficas

[1] Bauer, Christian e King, Gavin – Java persistence com Hibernate. Rio de Janeiro, Ed. Ciência Moderna, 2007;

[2] ”Jpa 2: os novos recursos inspirados no Hibernate” - Mundo Java 39;

[3] ”Jpa 2 – persistência à toda prova” - Java Magazine 81;

[4] Apostila k19 – Persistência com JPA2 e Hibernate

http://www.k19.com.br/downloads/apostilas