

---

# Programação com Banco de Dados – IMD0403 Aula 17 – Consulta com JPA

---

João Carlos Xavier Júnior

[jcxavier@imd.ufrn.br](mailto:jcxavier@imd.ufrn.br)

---

# Java Persistence Query Language

□ Java Persistence Query Language (JPQL) é uma **linguagem de consulta**:

- ❖ Orientada a objeto;
- ❖ Opera sobre classes e objetos, diferente do SQL que opera sobre tabelas;
- ❖ Independente de SGBD;
- ❖ Definida como parte da especificação Java Persistence API (JPA);
- ❖ Fortemente inspirada na linguagem SQL.

# Java Persistence Query Language

## ❑ Exemplos de consultas:

```
// Busca todos os alunos da tabela aluno
SQL: select * from aluno
JPQL: from Aluno
JPQL: from Aluno a
JPQL: from Alunos as alu
```

```
// Busca todos os alunos matricula >= 35
SQL: select * from aluno where matricula >= 35
JPQL: from Aluno a where a.matricula >= 35
```

```
// Todos os professores com nome começando com João
SQL: select * from professor where nome like "João%"
JPQL: from Professor p where p.nome like "João%"
```

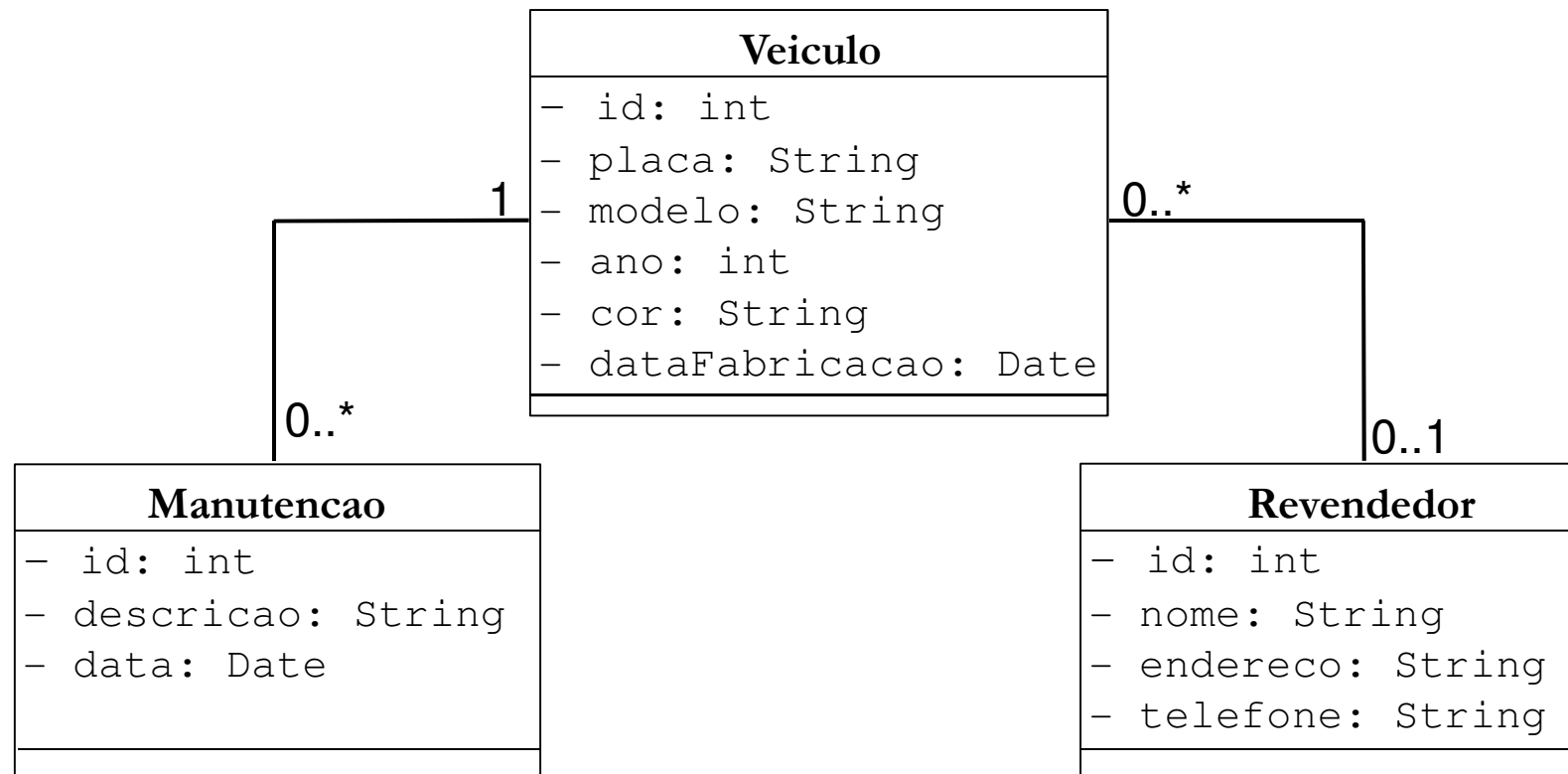
# Java Persistence Query Language

## ❑ Exemplos de consultas:

```
// Busca todos os alunos ordenados por nome (asc)  
SQL: select * from aluno order by nome asc  
JPQL: from Aluno order by nome asc
```

# Java Persistence Query Language

❑ Exemplo de domínio:



# Java Persistence Query Language

## ❑ Usando mapeamento @ManyToOne:

```
// Manutencoes de veículos do ano 2009
SQL: select m.* from veiculo v, manutencao m
      where v.id_veiculo = m.id_veiculo
      and v.ano = 2009
JPQL: select m from Manutencao m
      where m.veiculo.ano = 2009
```

```
// Veículos com nome do revendedor começando com A%
SQL: select r.* from veiculo v, revendedor r
      where v.id_revendedor = r.id_revendedor
      and r.nome like 'A%'
JPQL: from Veiculo v
      where v.revendedor.nome like 'A%'
```

# Java Persistence Query Language

## ❑ Usando mapeamento @ManyToOne:

```
// Manutenções com revendedor dos veículos com nome  
começando com A%
```

```
SQL: select m.* from manutencao m, veiculo v, revendedor r  
      where v.id_revendedor = r.id_revendedor  
      and m.id_veiculo = v.id_veiculo  
      and r.nome like 'A%'
```

```
JPQL: select m from Manutencao m  
       where m.veiculo.revendedor.nome like 'A%'
```

```
JPQL: select m from Manutencao m join m.veiculo v  
       where v.revendedor.nome like 'A%'
```

```
JPQL: select m from Manutencao m  
       join m.veiculo v join v.revendedor r  
       where r.nome like 'A%'
```

# Java Persistence Query Language

## ❑ Usando mapeamento @OneToMany:

```
// Revendedores de veiculos do ano = 2009
SQL: select distinct r.*
      from veiculo v, revendedor r
      where v.id_revendedor = r.id_revendedor
      and v.ano = 2009

JPQL: select distinct r from Revendedor r
      join r.veiculo v
      where v.ano = 2009
```



# Java Persistence Query Language

## ❑ Outras consultas:

```
// Conta quantos .....?  
SQL: select count(v.id_veiculo), v.ano  
      from veiculo v  
      group by v.ano  
  
JPQL: select count(v.id), v.ano  
        from Veiculo v  
        group by v.ano
```

```
SQL: select max(v.ano) from veiculo v  
  
JPQL: select max(v.ano) from Veiculo v
```

# Java Persistence Query Language

## ❑ Outras consultas:

```
JPQL: select v from Veiculo v  
      where year(dataFabricacao) = 2008
```

```
JPQL: select v from Veiculo v  
      where month(dataFabricacao) = 03
```

```
JPQL: select v from Veiculo v  
      where day(dataFabricacao) = 30
```

# Java Persistence Query Language

## ❑ Outras consultas:

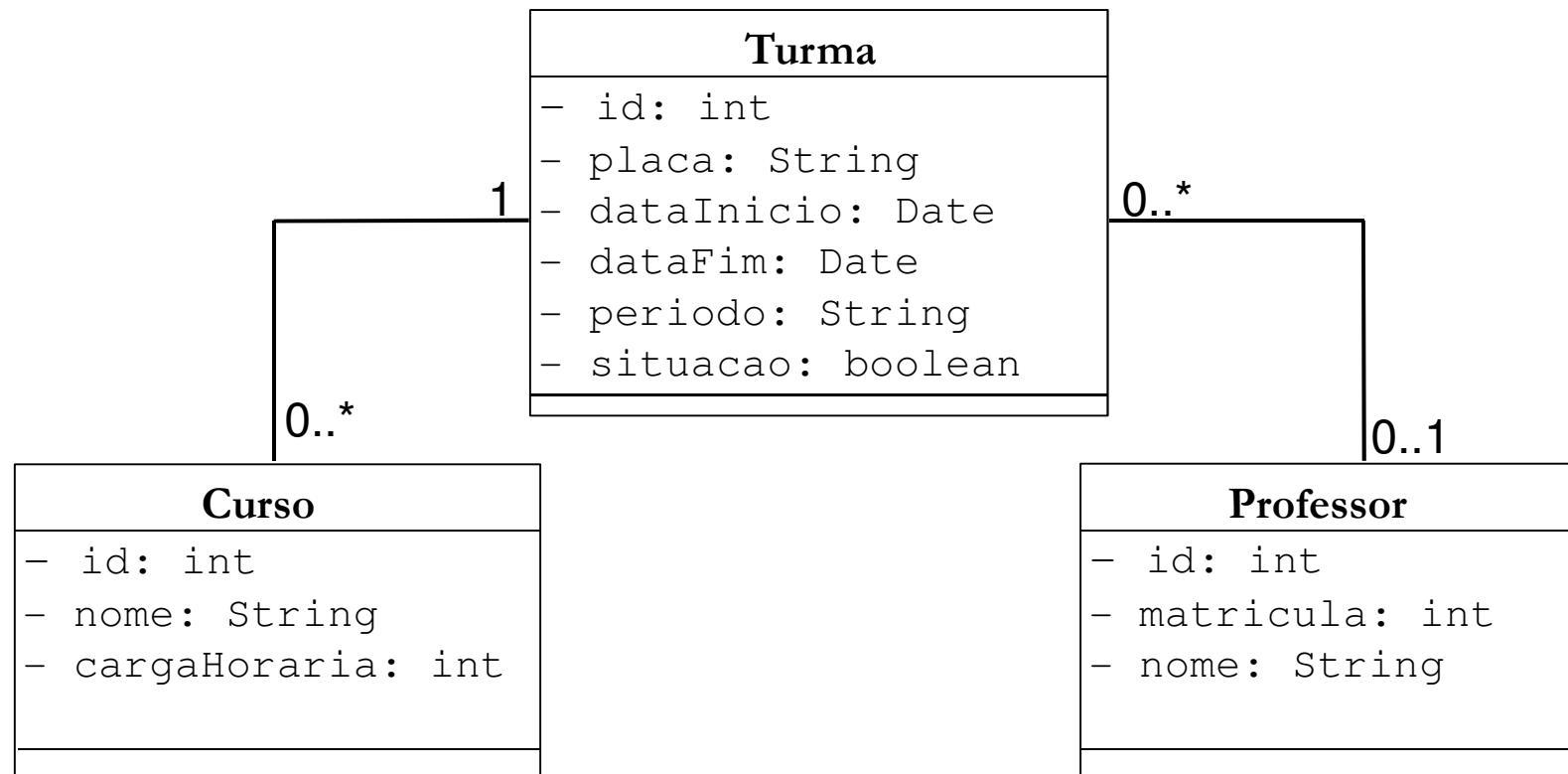
```
JPQL: select avg(p.peso), sum(p.peso), max(p.peso)
       from Pessoa p
```

```
JPQL: select max(p.idade), min(p.idade)
       from Pessoa p
```

```
JPQL: select p.idade, sum(p.peso), count(p)
       from Pessoa p
       group by p.idade
       having p.idade >= 12
       order by p.idade
```

# Java Persistence Query Language

## ❑ Exemplo de domínio 02:



# Java Persistence Query Language

## ❑ Usando mapeamento @ManyToMany:

```
// Todos os professores de turmas com periodo =  
// "Tarde" e com cursos de cargaHoraria > 40  
SQL: select p.* from professor p  
      join professor_turma pt on (pt.id_professor =  
      p.id_professor)  
      join turma t on (t.id_turma = pt.id_turma)  
      join curso c on (c.id_curso = t.id_curso)  
      where t.periodo = "Tarde"  
      and c.carga_horaria > 40
```

```
JPQL: select p from Professor p  
      join p.turma t  
      where t.periodo = "Tarde"  
      and t.curso.cargaHoraria > 40
```

# Java Persistence Query Language

## ❑ Usando JPQL na aplicação:

### ❖ Interface Query para listando todos os registros:

```
// Todos os alunos  
Query q = em.createQuery("from Aluno");  
q.getResultList();
```

```
// Todos os cursos começando pela letra A:  
String s = "from Curso c where c.nome like 'A%'";  
Query q = em.createQuery(s);  
q.getResultList();
```

```
// Todos os cursos com que tem o departamento = 2  
String s = "from Curso c where c.departamento.id = 2";  
Query q = em.createQuery(s);  
q.getResultList();
```

# Java Persistence Query Language

## ❑ Domínio Autores / Livros:

```
@Entity
public class Autor implements Serializable{
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Integer id;
    private String nome;
    private String cpf;

    @ManyToMany
    @JoinTable(name="autor_livro",
        joinColumns = @JoinColumn(name="id_autor"),
        inverseJoinColumns = @JoinColumn(name="id_livro"))
    private Collection<Livro> livros;

    // Getters and Setters
}
```

# Java Persistence Query Language

## ❑ Domínio Autores / Livros:

```
@Entity
public class Livro implements Serializable{
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Integer id;
    private String titulo;
    private String isbn;

    @ManyToMany
    @JoinTable(name="autor_livro",
        joinColumns = @JoinColumn(name="id_livro"),
        inverseJoinColumns = @JoinColumn(name="id_autor"))
    private Collection<Autor> autores;

    // Getters and Setters
}
```



# Java Persistence Query Language

## ❑ Domínio Autores / Livros:

```
public static void main(String[] args) {  
    EntityManagerFactory emf = Persistence.createEntityManagerFactory("JP_QL");  
    EntityManager em = emf.createEntityManager();  
  
    Autor a1 = new Autor();  
    a1.setNome("Francisco Pereira");  
    a1.setCpf("125.478.365-33");  
  
    Autor a2 = new Autor();  
    a2.setNome("Roberto Carlos da Silva");  
    a2.setCpf("521.587.745-47");  
  
    Autor a3 = new Autor();  
    a3.setNome("Angelina Jolie");  
    a3.setCpf("632.417.258-11");  
}
```

# Java Persistence Query Language

## ❑ Domínio Autores / Livros:

```
Livro l1 = new Livro();
l1.setTitulo("Malévola");
l1.setIsbn("4567-6333");
l1.setAutores(new ArrayList<Autor>());
l1.getAutores().add(a3);
l1.getAutores().add(a1);

Livro l2 = new Livro();
l2.setTitulo("Second World War");
l2.setIsbn("4714-5877");
l2.setAutores(new ArrayList<Autor>());
l2.getAutores().add(a2);

em.getTransaction().begin();
em.persist(a1);
em.persist(a2);
em.persist(a3);
em.persist(l1);
em.persist(l2);
em.getTransaction().commit();
System.out.println("Dados cadastrados!!!");
```

# Java Persistence Query Language

□ Tabelas Autores / Livros / Autor\_Livro:

Edit Data - PostgreSQL 9.3 (localhost:5432) - bd_heranca - autor			
File Edit View Tools Help			
	<b>id</b> [PK] integer	<b>cpf</b> character varying(255)	<b>nome</b> character varying(255)
1	10	125.478.365-33	Francisco Pereira
2	11	521.587.745-47	Roberto Carlos da Silva
3	12	632.417.258-11	Angelina Jolie

Edit Data - PostgreSQL 9.3 (localhost:5432) - bd_heranca - livro			
File Edit View Tools Help			
	<b>id</b> [PK] integer	<b>isbn</b> character varying(255)	<b>titulo</b> character varying(255)
1	20	4567-6333	Malévola
2	21	4714-5877	Second World War

Edit Data - PostgreSQL 9.3 (localhost:5432) - bd_heranca - autor_livro		
File Edit View Tools Help		
	<b>id_autor</b> integer	<b>id_livro</b> integer
1	12	20
2	10	20
3	11	21

# Java Persistence Query Language

## ❑ Consultando Livros:

```
System.out.println("Consultas: ");
@SuppressWarnings("unchecked")
List<Livro> lista = em.createQuery("from Livro").getResultList();
for (Livro livros : lista) {
    System.out.println(livros.getId() + " " + livros.getTitulo() +
        " " + livros.getIsbn());
}
```

```
Dados cadastrados!!!
Consultas:
Hibernate:
    select
        livro0_.id as id1_1_,
        livro0_.isbn as isbn2_1_,
        livro0_.titulo as titulo3_1_
    from
        Livro livro0_
20 Malévola 4567-6333
21 Second World War 4714-5877
```

# Java Persistence Query Language

## ❑ Consultando Livros:

```
System.out.println("Consultas: ");
@SuppressWarnings("unchecked")
List<Livro> lista = em.createQuery("select li from Livro li").getResultList();
for (Livro livros : lista ) {
    System.out.println(livros.getId() + " " + livros.getTitulo() +
        " " + livros.getIsbn());
}

Dados cadastrados!!!
Consultas:
Hibernate:
    select
        livro0_.id as id1_1_,
        livro0_.isbn as isbn2_1_,
        livro0_.titulo as titulo3_1_
    from
        Livro livro0_
20 Malévola 4567-6333
21 Second World War 4714-5877
```

# Java Persistence Query Language

- ❑ É possível definir nome aos parâmetros:

```
// Lista de objetos Aluno
Query q = em.createQuery("from Aluno a where a.nome = :nome");
q.setParameter("nome", "João");
List result = q.getResultList();
```

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

q.setParameter("tipo", 1);
q.setParameter("salario", 1500);
List result = q.getResultList();
```

# Java Persistence Query Language

## ❑ Retorno de um objeto único:

```
// Único objeto como resultado
Query q = em.createQuery("
    select a from Aluno a where a.id = 109");
Aluno a = (Aluno)q.getSingleResult();
```

# Dúvidas...





# Exercício

## □ Mapear o domínio abaixo:

- ❖ Persistir as informações;
- ❖ Elaborar e executar as seguintes consultas:
  - Retornar todos os veículos;
  - Retornar todos os revendedores do modelo Astra.
  - Retornar todos os veículos que possuírem manutenção igual a “troca do ar-condicionado”.

