

# DESENVOLVIMENTO DE SOFTWARE PARA A WEB 1

## Módulo 7 Java Persistence API (JPA)

**Delano Medeiros Beder**  
**[delano@dc.ufscar.br](mailto:delano@dc.ufscar.br)**

# Persistência de Dados



- Tabelas
- Colunas
- Registros
- Chaves



- Arquivos
- Bancos de dados Hierárquicos
- **Bancos de dados Relacionais**
- Bancos de dados Orientados a Objetos
- Modelos mais recentes (NoSQL)...

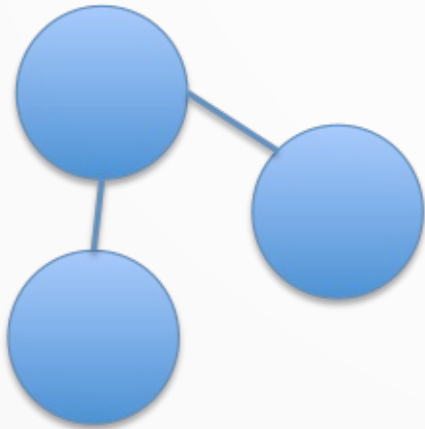


# Persistência de Dados

Programação Orientada a Objetos



Banco de dados relacional

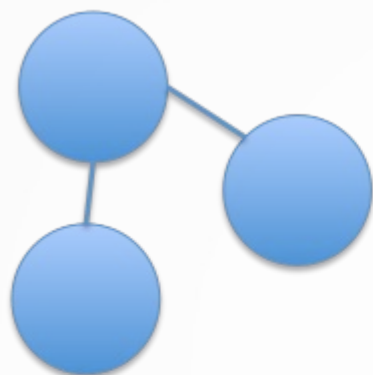


```
public void save(Produto produto){
    String sql = "INSERT INTO produtos (nome ,preco) VALUES ('"
        + produto.getNome() + "', " + produto.getPreco() + ")";
    Connection conn = null;
    Statement stmt = null;
    try {
        Class.forName("org.gjt.mm.mysql.Driver");
        conn = DriverManager.getConnection("jdbc:mysql://host/bd",
            "us", "ps");
        stmt = conn.createStatement();
        stmt.executeUpdate(sql);
    } catch (ClassNotFoundException e) {
        //...
    } catch (SQLException e) {
        //...
    } finally { //... }
```

Programação Orientada a Objetos



Banco de dados relacional



Soluções de ORM (Object Relational Mapping):  
•Hibernate, KODO, JDO, TopLink, EJB EntityBean...



**JPA**

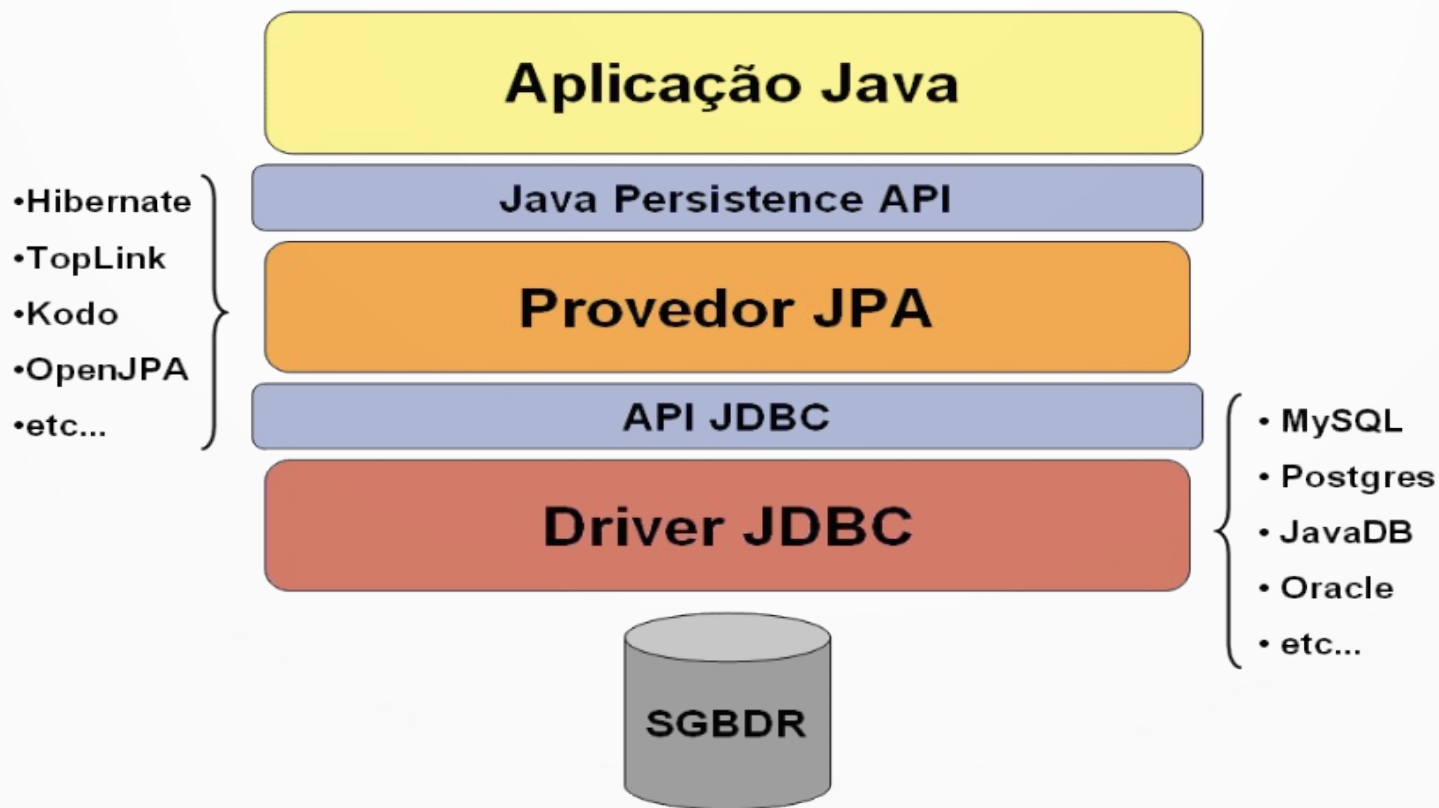


# Java Persistence API (JPA)

- API para abstração da camada de persistência das aplicações orientadas a objetos
  - v1.0 - 2006 / v2.0 - 2009
  - Integração com diversos frameworks (não pode ser usado isoladamente)
- Mapeamento objeto / relacional
  - Classe para tabela
  - Atributo para coluna
  - Relacionamento para chave estrangeira
- API para salvar, atualizar, excluir os objetos do banco

# Java Persistence API (JPA)

JPA depende de JDBC e de um framework (provedor JPA) de persistência



# Como usar JPA

- Configurar bibliotecas no projeto
  - JPA + Framework + Driver SQL
- Mapear as classes para as devidas tabelas/colunas e os relacionamentos para as chaves estrangeiras
  - Anotações Java
- Configurar uma unidade de persistência
  - persistence.xml
- Usar as classes da API JPA para gerenciar os objetos no banco de dados



# (1) Configurar bibliotecas no projeto

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>4.3.8.Final</version>
  <scope>compile</scope>
</dependency>

<!-- Implementação de EntityManager da JPA -->

<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-entitymanager</artifactId>
  <version>4.3.8.Final</version>
  <scope>compile</scope>
</dependency>

<!-- Driver JDBC (banco de dados MySQL) -->

<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.21</version>
  <scope>runtime</scope>
</dependency>
```

Provedor JPA

EntityManager JPA

Driver JDBC

## (2) Mapeando as entidades de persistência

```
@Entity
@Table(name = "Disciplina")
public class Disciplina {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true, length = 30)
    private String nome;

    @Column(nullable = false, unique = true, length = 5)
    private String sigla;

    @ManyToMany(targetEntity = Aluno.class, mappedBy = "disciplinas", fetch = FetchType.EAGER)
    private Set<Aluno> alunos;

    @OneToOne
    private Professor professor;
```

## (2) Mapeando as entidades de persistência

Anotações para mapeamento de tabelas e colunas

@Entity

@Table (name="")

@Column(name="")

@Id

@GeneratedValue(strategy=GenerationType.X)

➤ X → IDENTITY, SEQUENCE, TABLE, AUTO

@Temporal(TemporalType.X)

➤ X → TIMESTAMP, TIME, DATE

@Transient

## (3) Configurando a unidade de persistência

- Unidade de persistência (Persistence Unit)
  - Qual é o banco de dados ?
  - Quais são as classes persistentes ?
  - Como as transações são gerenciadas ?, etc
- Configurada no arquivo **persistence.xml**
  - Deve estar no diretório **META-INF** na raiz da estrutura de pacotes das classes persistentes.

# (3) Configurando a unidade de persistência

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
  version="2.0">

  <persistence-unit name="JPAPU">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <properties>
      <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/JPA" />
      <property name="javax.persistence.jdbc.user" value="root" />
      <property name="javax.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver" />
      <property name="javax.persistence.jdbc.password" value="root" />
      <property name="javax.persistence.schema-generation.database.action" value="drop-and-create" />
      <property name="hibernate.show_sql" value="true" />
      <property name="hibernate.format_sql" value="true"/>
      <property name="hbm2ddl.auto" value="create"/>
    </properties>
  </persistence-unit>
</persistence>
```

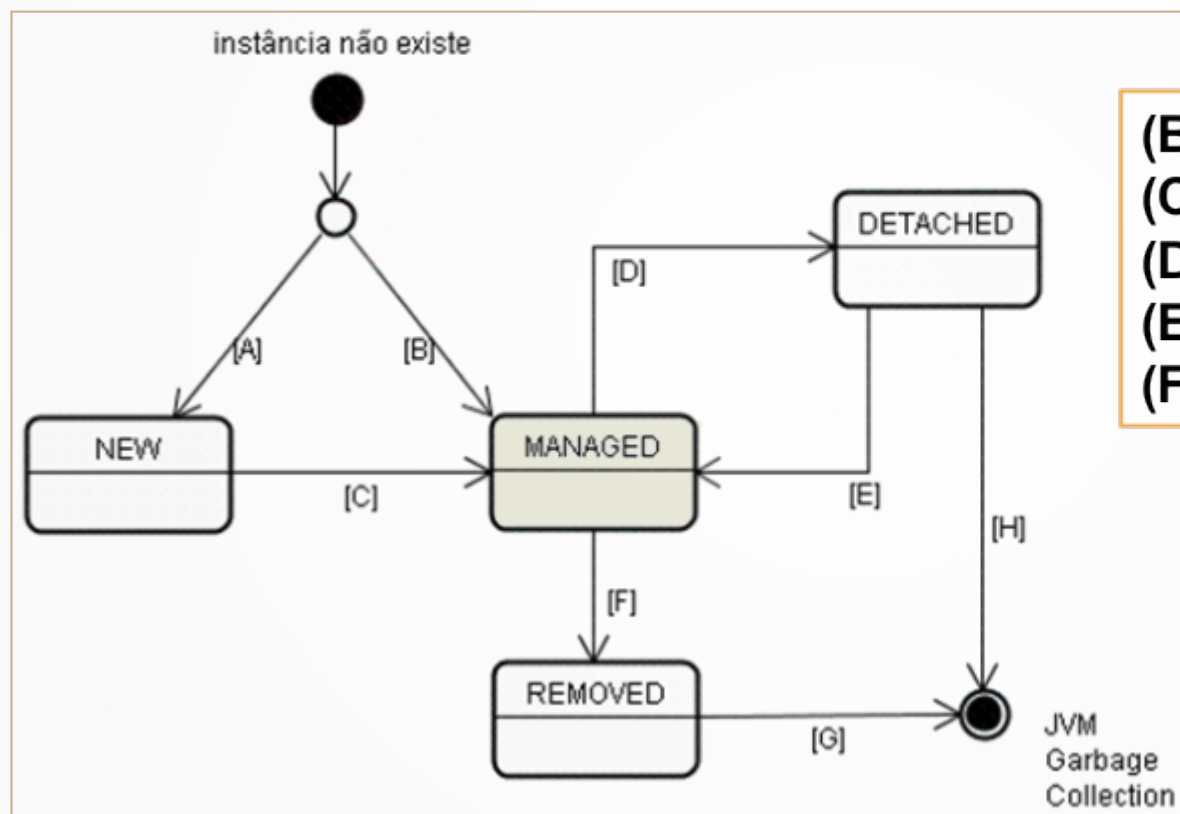
## (4) EntityManager

- **EntityManagerFactory** (pacote *javax.persistence*)
  - Fábrica de **EntityManagers**
- Apenas uma instância por aplicação
  - alto custo de inicialização
- Definida em uma unidade de persistência



## ➤ *EntityManager* (*javax.persistence*)

- Métodos para gerenciar os objetos no banco de dados



**(B) find / query**

**(C) persist**

**(D) clear**

**(E) merge**

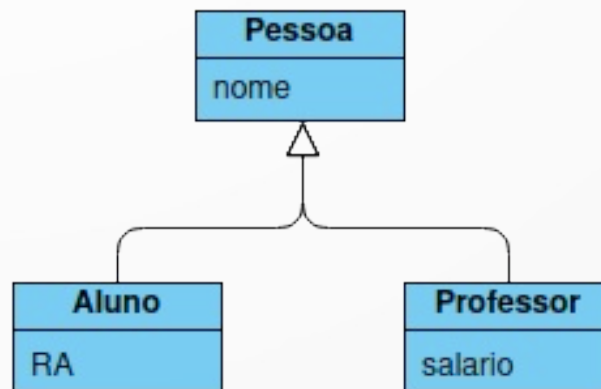
**(F) remove**

## (4) Uso das classes da API JPA

```
public class Main {  
  
    public static void main(String[] args) {  
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("JPAPU");  
        EntityManager em = emf.createEntityManager();  
        EntityTransaction tx = em.getTransaction();  
  
        Departamento departamento = new Departamento("Computação", "dc");  
  
        tx.begin();  
        em.persist(departamento);  
        tx.commit();  
        em.close();  
        emf.close();  
    }  
}
```

# Relacionamentos

- JPA permite mapear os relacionamentos entre as classes para tabelas
- Associações
  - @OneToOne, @OneToMany, @ManyToOne
- Herança
  - Estratégias:
    - MappedSuperclass
    - Table per Class
    - Joined Table
    - Single Table



# Estratégia: Mapped Superclass

**Estratégia mais simples. Ela mapeia cada classe concreta para sua própria tabela. Porém essa estratégia não permite relacionamentos e consultas polimórficas que envolvem a classe mãe (abstrata).**

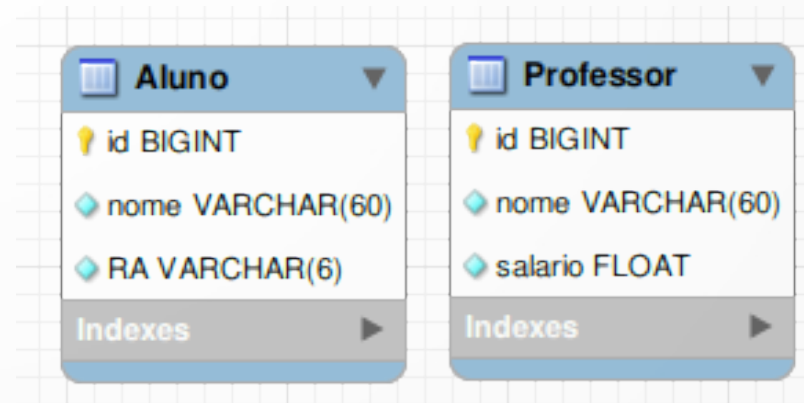
```
@MappedSuperclass
public abstract class Pessoa {

    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true, length = 60)
    private String nome;
}
```

```
@Entity
@Table(name = "Aluno")
public class Aluno extends Pessoa {

    @Column(nullable = false, unique = true, length = 6)
    private String RA;
}
```



```
@Entity
@Table(name = "Professor")
public class Professor extends Pessoa {

    @Column(nullable = false)
    private float salario;
}
```

# Estratégia: Table Per Class

Essa estratégia mapeia cada entidade concreta para sua tabela que contém todos os atributos da entidade (inclusive os atributos herdados). O esquema resultante é similar em relação a estratégia **MappedSuperclass**, porém essa estratégia permite a associação e consultas polimórficas com todas as entidades presentes na hierarquia.

```
@Entity
@Table(name = "Pessoa")
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class Pessoa {

    @Id @GeneratedValue(strategy=GenerationType.TABLE)
    protected Long id;

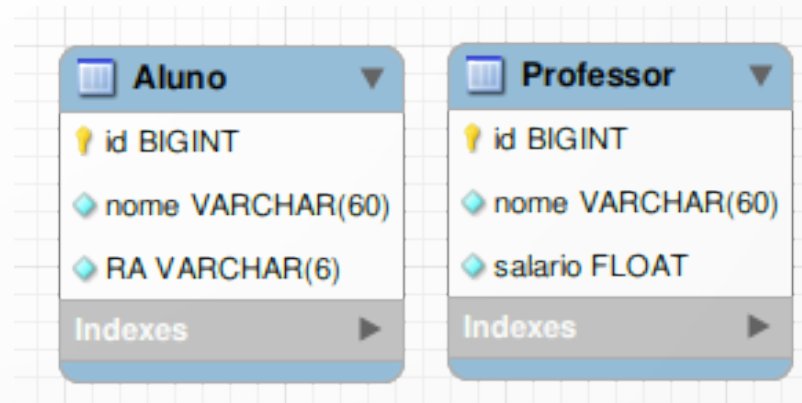
    @Column(nullable = false, unique = true, length = 60)
    protected String nome;
}
```

```
@Entity
@Table(name = "Aluno")
public class Aluno extends Pessoa {

    @Column(nullable = false, unique = true, length = 6)
    private String RA;
}
```

```
@Entity
@Table(name = "Professor")
public class Professor extends Pessoa {

    @Column(nullable = false)
    private float salario;
}
```



# Estratégia: Single Table

Cria uma única classe para toda a hierarquia de classes. É a estratégia *default* adotada pelo JPA, caso não seja definida outra explicitamente.

```
@Entity
@Table(name = "Pessoa")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
public abstract class Pessoa {

    @Id @GeneratedValue(strategy=GenerationType.AUTO)
    protected Long id;

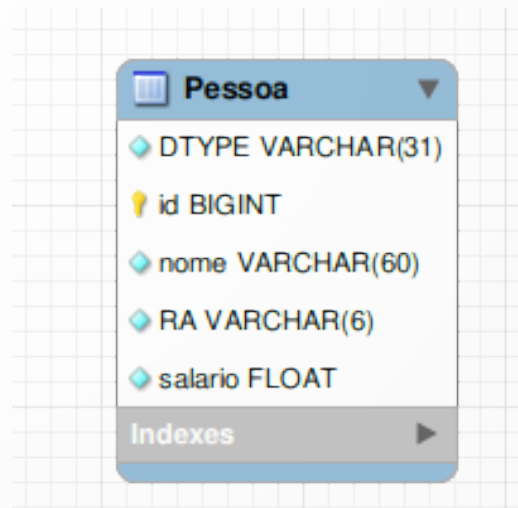
    @Column(nullable = false, unique = true, length = 60)
    protected String nome;
}
```

```
@Entity
@Table(name = "Aluno")
public class Aluno extends Pessoa {

    @Column(nullable = false, unique = true, length = 6)
    private String RA;
}
```

```
@Entity
@Table(name = "Professor")
public class Professor extends Pessoa {

    @Column(nullable = false)
    private float salario;
}
```





# Estratégia: Joined Table

Cada classe na hierarquia é mapeada para uma tabela. A única coluna compartilhada que se repete em todas as tabelas é o identificador (id), que é utilizada para realizar a operação de *join* quando necessário.

```
@Entity
@Table(name = "Pessoa")
@Inheritance(strategy = InheritanceType.JOINED)
public abstract class Pessoa {

    @Id @GeneratedValue(strategy=GenerationType.AUTO)
    protected Long id;

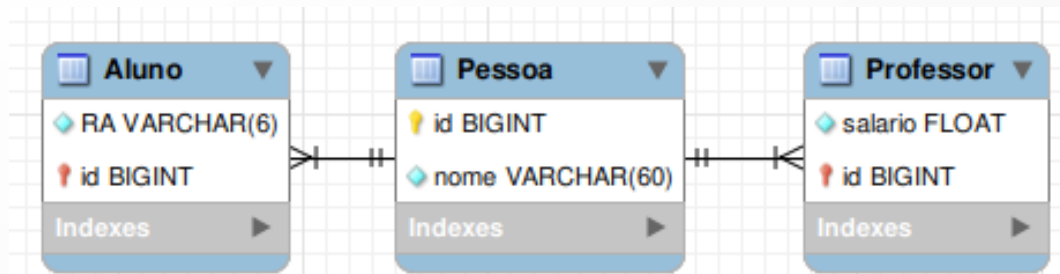
    @Column(nullable = false, unique = true, length = 60)
    protected String nome;
}
```

```
@Entity
@Table(name = "Aluno")
public class Aluno extends Pessoa {

    @Column(nullable = false, unique = true, length = 6)
    private String RA;
}
```

```
@Entity
@Table(name = "Professor")
public class Professor extends Pessoa {

    @Column(nullable = false)
    private float salario;
}
```



# @OneToMany / @ManyToOne

```
@Entity
@Table(name = "Professor")
public class Professor extends Pessoa {

    @Column(nullable = false)
    private float salario;

    @ManyToOne
    @JoinColumn(name = "departamento_id")
    private Departamento departamento;
}
```

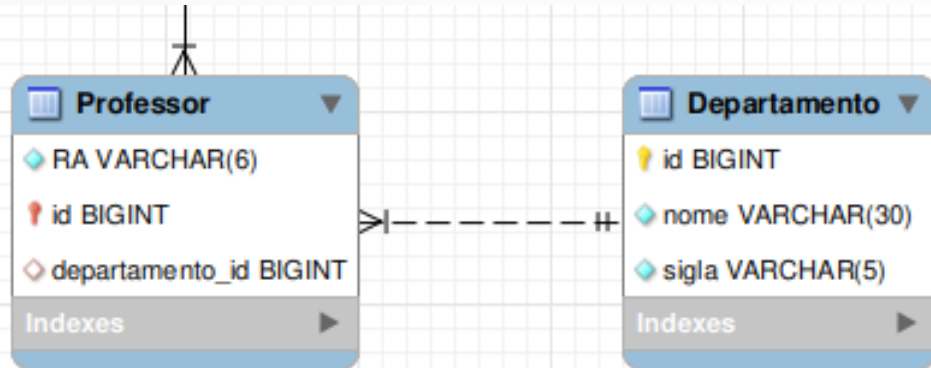
```
@Entity
@Table(name = "Departamento")
public class Departamento {

    @Id @GeneratedValue(strategy=GenerationType.AUTO)
    private Long id;

    @Column(nullable = false, unique = true, length = 30)
    private String nome;

    @Column(nullable = false, unique = true, length = 5)
    private String sigla;

    @OneToMany(mappedBy = "departamento")
    private List<Professor> professores;
}
```



**N** Professores  
Trabalham em apenas  
**1** Departamento

Chave estrangeira:

Professor ➡ Departamento

# @ManyToMany

```
@Entity
@Table(name = "Aluno")
public class Aluno extends Pessoa {

    @Column(nullable = false, unique = true, length = 6)
    private String RA;

    @ManyToMany(targetEntity = Disciplina.class)
    private Set<Disciplina> disciplinas;
}
```

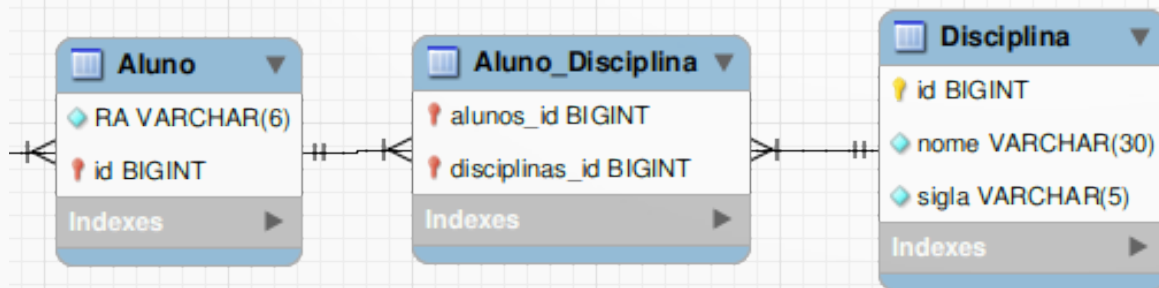
```
@Entity
@Table(name = "Disciplina")
public class Disciplina {

    @Id @GeneratedValue(strategy=GenerationType.AUTO)
    private Long id;

    @Column(nullable = false, unique = true, length = 30)
    private String nome;

    @Column(nullable = false, unique = true, length = 5)
    private String sigla;

    @ManyToMany(targetEntity=Aluno.class, mappedBy = "disciplinas")
    private Set<Aluno> alunos;
}
```



Relacionamento N x N entre Alunos e Disciplinas

Tabela **Aluno\_Disciplina**

# @OneToOne

```
@Entity
@Table(name = "Professor")
public class Professor extends Pessoa {

    @Column(nullable = false)
    private float salario;

    @ManyToOne
    @JoinColumn(name = "departamento_id")
    private Departamento departamento;

    @OneToOne(mappedBy = "professor")
    private Disciplina disciplina;

}
```

```
@Entity
@Table(name = "Disciplina")
public class Disciplina {

    @Id @GeneratedValue(strategy=GenerationType.AUTO)
    private Long id;

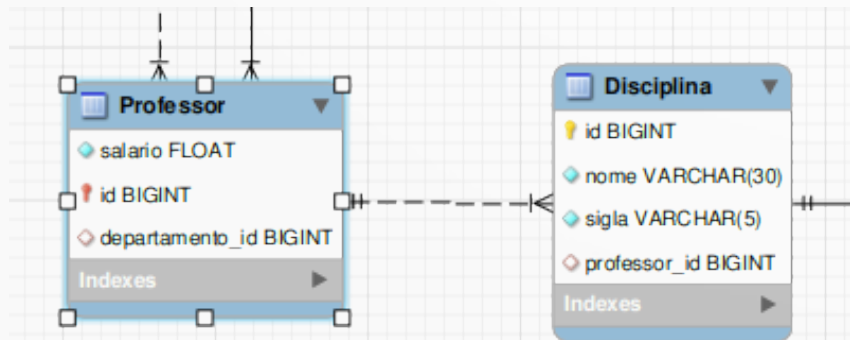
    @Column(nullable = false, unique = true, length = 30)
    private String nome;

    @Column(nullable = false, unique = true, length = 5)
    private String sigla;

    @ManyToMany(targetEntity=Aluno.class, mappedBy = "disciplinas")
    private Set<Aluno> alunos;

    @OneToOne
    private Professor professor;

}
```



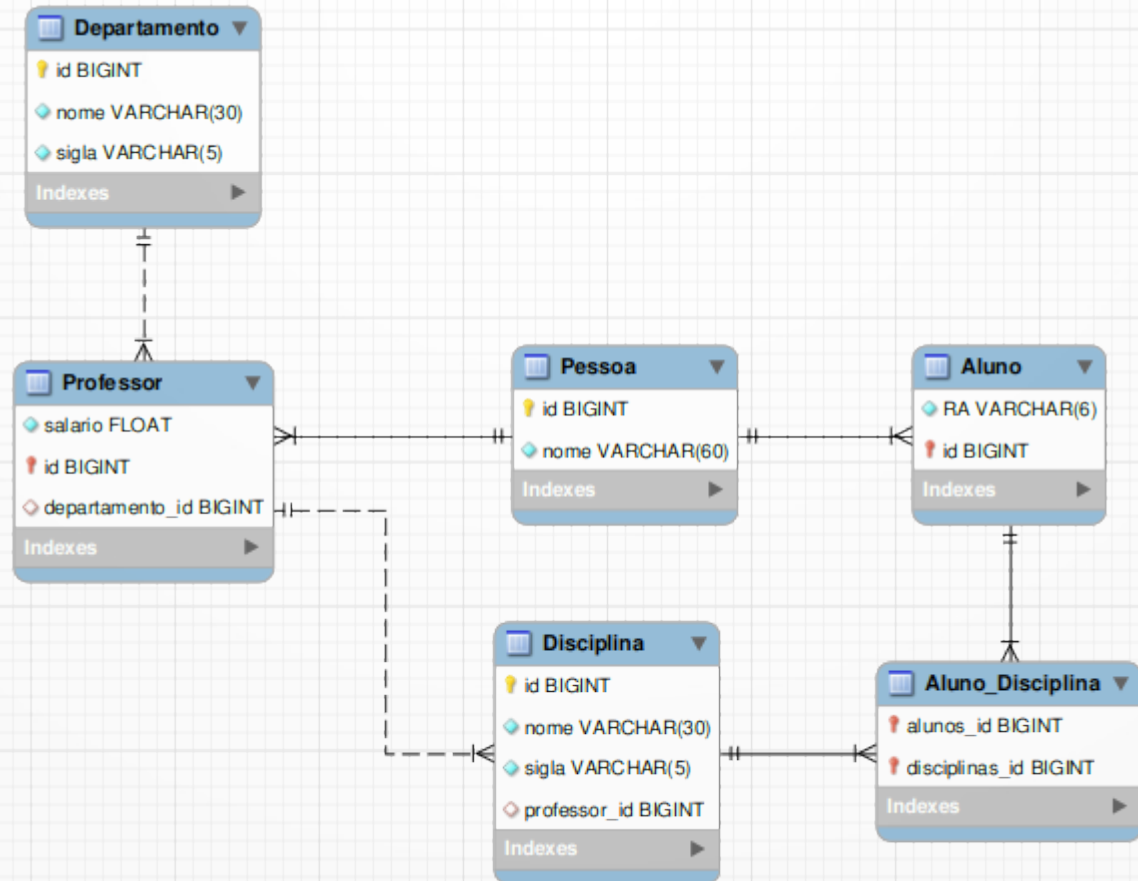
Relacionamento 1 x 1 entre  
Professor e Disciplina

Situação hipotética:

1 professor apenas leciona 1 disciplina

1 disciplina é lecionada por apenas 1 professor

# Modelo Final





# Spring Data JPA

- Spring Data é um projeto Spring de alto nível cujo objetivo é unificar e facilitar o acesso a diferentes tipos de armazenamentos de persistência, tanto sistemas de banco de dados relacionais quanto armazenamentos de dados NoSQL.
- Spring Data JPA, parte da família Spring Data, torna mais fácil implementar repositórios baseados em JPA. Este módulo trata do suporte aprimorado para camadas de acesso a dados baseadas em JPA. Torna mais fácil construir aplicativos com tecnologia Spring que usam tecnologias de acesso a dados.
- A dependência ***spring-boot-starter-data-jpa*** fornece uma maneira rápida de começar. Ele fornece as seguintes funcionalidades principais:
  - Hibernate: uma das implementações de *JPA provider* mais populares.
  - Spring Data JPA: Ajuda a implementar repositórios baseados em JPA.
  - Spring ORM: Suporte ao ORM do Spring Framework.



# Spring Data JPA: Entidades

- Tradicionalmente, as classes de “Entidade” JPA são especificadas em um arquivo **persistence.xml**.
  - Com Spring Data JPA, este arquivo não é necessário e uma “procura automática de entidades” é usada em seu lugar.
- Todas as classes anotadas com @Entity, @Embeddable ou @MappedSuperclass são consideradas como classes de “Entidade” JPA.

# Spring Data: Repositórios JPA

O objetivo da abstração de *Repository JPA* do Spring Data é reduzir o esforço para implementar camadas de acesso a dados para vários armazenamentos de persistência de maneira significativa.

Os repositórios JPA são interfaces que você pode definir para acessar os dados. As consultas JPA são criadas automaticamente a partir dos nomes dos seus métodos.

Para consultas mais complexas, você pode anotar seu método com a anotação Query.

Hierarchy :

```
Interface JpaRepository<T, ID>  
    interface PagingAndSortingRepository<T, ID>  
        interface CrudRepository<T, ID>  
            interface Repository<T, ID>
```

```
@Query("select u from User u where u.name=?1")  
User findByUserName(String name);  
@Query("select u from User u where u.name like%:name%")  
User findByUserName(@Param("name") String name);  
@Query(value = "select * from user where name=?1", nativeQuery = true)  
User findByUserName(String name);
```

# Spring Data: Repositórios JPA

```
package br.ufscar.dc.dsw.dao;

import java.util.List;

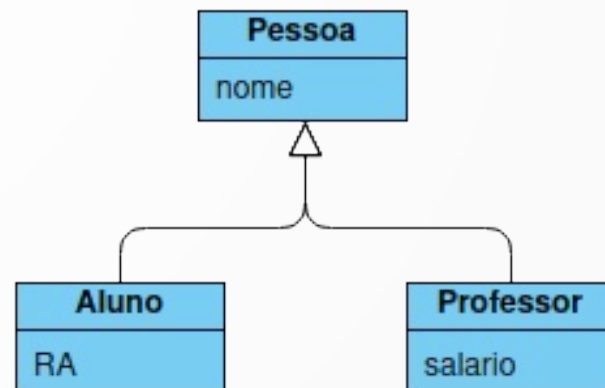
@SuppressWarnings("unchecked")
public interface IPessoaDAO extends CrudRepository<Pessoa, Long>{

    Pessoa findById(long id);

    List<Pessoa> findAll();

    Pessoa save(Pessoa pessoa);

    void deleteById(Long id);
}
```



```
package br.ufscar.dc.dsw.dao;

import java.util.List;

public interface IProfessorDAO extends CrudRepository<Professor, Long> {
    List<Professor> findByDepartamento(Departamento departamento);
}
```

**FIM**