



INSTITUTO FEDERAL
Rio Grande do Sul

Desenvolvimento de Sistemas I

Persistência de dados

Professor: Jezer Machado de Oliveira



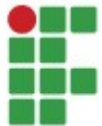
Persistência

- Vital para o desenvolvimento de aplicações
- Quase todas as aplicações necessitam que dados sejam persistidos
- Necessidades
 - Armazenamento
 - Busca
 - Organização
 - Compartilhamento dos dados



Banco de Dados

Coleção de dados que se relacionam entre si, para produzir informações sobre um domínio

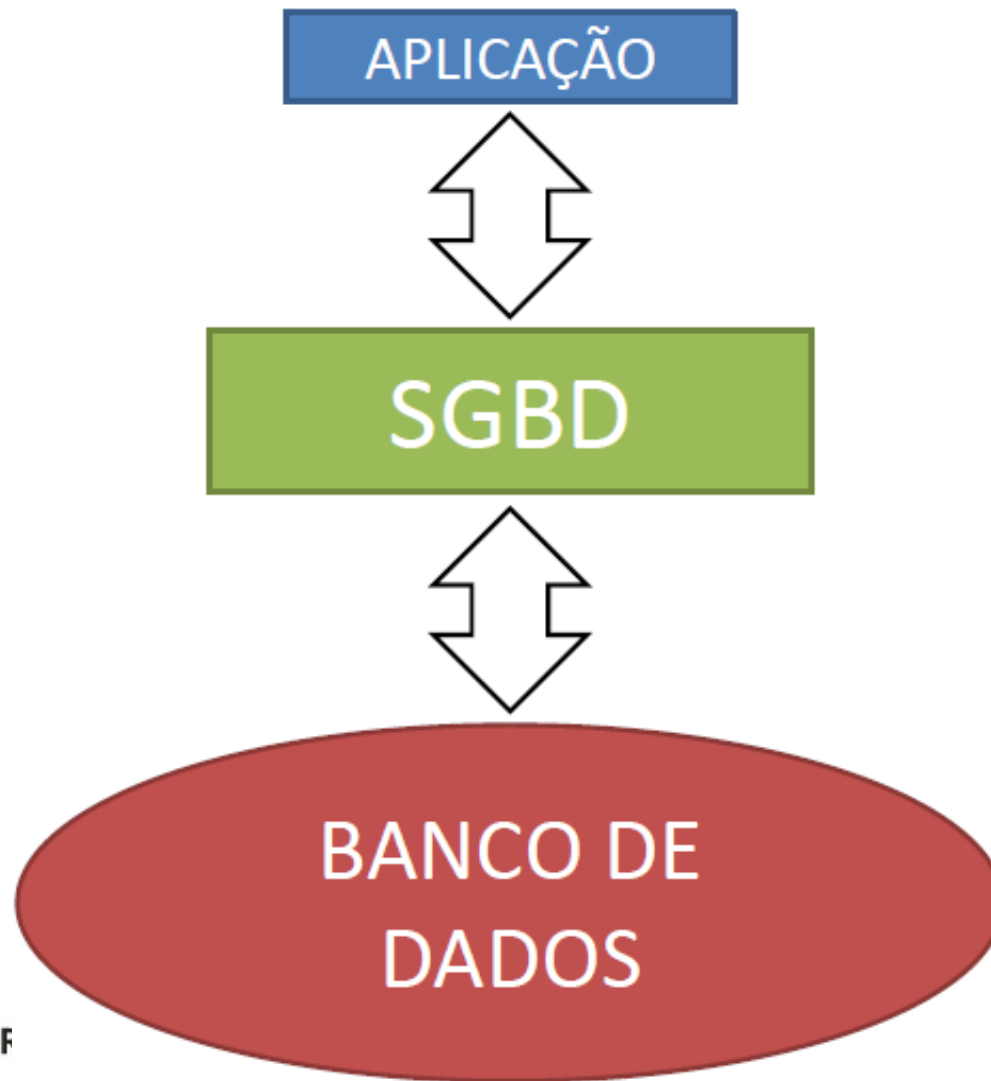


SGBD

É um conjunto de programas que gerenciam a estrutura do banco de dados e controlam o acesso aos dados armazenados



SGBD



Vantagem do SGBD

- Independência
- Acesso eficiente aos dados
- Integridade
- Segurança
- Acesso concorrente
- Recuperação de falhas



Banco de dados relacional

- Proposto por Edgar Frank Codd em 1970
- É um modelo lógico formal, baseado na teoria matemática das relações
- Tornou-se um padrão para aplicações comerciais
- Formada por dois elementos principais
 - Entidade
 - Relacionamento



Entidade (Tabela)

- Possui um nome único
- Conjunto de atributo
 - Nome (coluna)
 - Domínio (tipo)
- Tuplas (linhas)
 - Lista ordenada de valores representa um registro
 - Representada por uma chave



Relacionamento

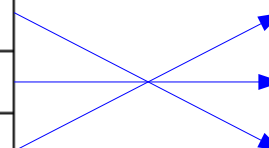
- Representa uma associação entre entidades distintas
- Possui uma cardinalidade
- Representada pela chave estrangeira



Banco de dados relacional

Disciplinas			
Id	Nome	CargaHoraria	Professor
1	Banco de Dados II	80	3
2	Programação III	120	2
3	Sistemas Operacionais	80	1

Professores	
Id	Nome
1	Rodrigo
2	Gleison
3	Jezer



Classe x Tabela

Classe

Produto
-id: int -nome: String -valor: double
+getId(): int +setId(id:int) +getNome(): String +setNome(nome:String) +getValor(): double +setValor(valor:double)

?

==

Tabela

produto
id : int(11)
nome : varchar(255)
valor : double



Classe x Tabela

Tabelas e classes são elementos diferentes, assim como o modelo relacional e orientado a objeto.



Convertendo tabela em objetos

```
@RestController
public class Produtos {

    @RequestMapping(path = "/produtos", method = RequestMethod.GET)
    public ArrayList<Produto> listar() {
        try {
            Connection connection = DriverManager.getConnection("jdbc:mysql://localhost/dsl", "root", "");
            CallableStatement prepareCall = connection.prepareCall(
                "SELECT id, nome, valor, endereco_id FROM produto");
            ResultSet executeQuery = prepareCall.executeQuery();
            ArrayList<Produto> produtos = new ArrayList<>();
            while(executeQuery.next()){
                Produto produto = new Produto();
                produto.setId(executeQuery.getInt("id"));
                produto.setNome(executeQuery.getString("nome"));
                produto.setValor(executeQuery.getDouble("valor"));
                produtos.add(produto);
            }
            connection.close();
            return produtos;
        } catch (Exception e) {
            return null;
        }
    }
}
```

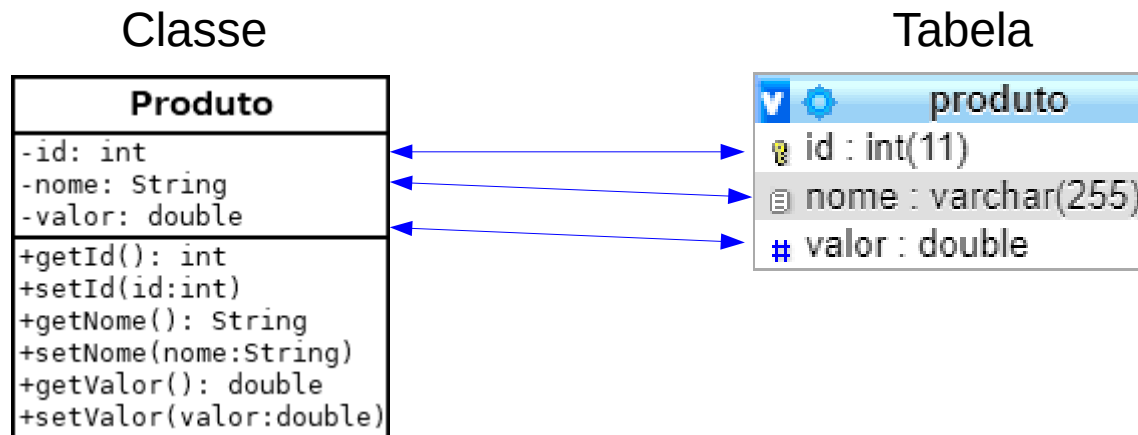


Modelo orientado a objeto

- Final dos anos 80
- Informação representada através de objetos como na programação OO: classes, herança...
- Mais complexo que o modelo relacional
- Poucas implementações puras
- Bancos de dados objeto-relacionais são híbridos dos dois modelos: IBM DB2, Oracle, SQL Server



Mapeamento objeto-relacional



Mapeamento objeto-relacional

- Permite a persistência de objetos em tabelas de banco de dados relacional
- Automática e transparente
- Utiliza metadados para descrever o relacionamento entre os objetos e a base de dados
 - XML
 - Annotations



Mapeamento objeto-relacional

- Necessita:
 - Maneira para a especificação de metadados
 - API para realização de operações básicas (CRUD)
 - Linguagem ou API para a realização de consultas



JPA

- Java Persistence API (JPA)
- Especificação elaborada pelo Java Community Process para persistência em Java
- Baseou-se em diversas soluções existentes
- Frameworks existentes passaram a implementar a especificação
- Recursos são um sub-conjunto dos encontrados nos frameworks que a implementam



JPA

- Não há a necessidade de se herdar de uma superclasse ou de implementar um interface
 - POJO - Plain Ordinary Java Objects
- Classes podem ser reutilizadas fora do contexto de persistência
 - Interface com o usuário
 - Testes de unidades
- As entidades não precisam nem saber que serão persistidas
 - Grande nível de portabilidade



Hibernate

- Framework de mapeamento Objeto-Relacional
- Implementa a API JPA
- Mais utilizado e documentado
- Mantido pela Jboss sob a licença LGPL
- Suporta classes desenvolvidas com agregação, herança, polimorfismo, composição e coleções
- Suporta uma série de banco de dados



Mapeamento JPA

@Entity

```
public class Produto {
```




@Id

```
private int id;  
private String nome;  
private float valor;
```

```
public int getId() {  
    return id;  
}
```



Mapeamento JPA

v		ds1 produto
		id : int(11)
		nome : varchar(255)
#		valor : float



Mapeamento JPA

Java	Tipo SQL
int or java.lang.Integer	INTEGER
long or java.lang.Long	BIGINT
short or java.lang.Short	SMALLINT
float or java.lang.Float	FLOAT
double or java.lang.Double	DOUBLE
java.math.BigDecimal	NUMERIC
java.lang.String	CHAR(1)
java.lang.String	VARCHAR
byte or java.lang.Byte	TINYINT
boolean or java.lang.Boolean	BIT



Mapeamento JPA

Java	Tipo SQL
java.util.Date or java.sql.Date	DATE
java.util.Date or java.sql.Time	TIME
java.util.Date or java.sql.Timestamp	TIMESTAMP
java.util.Calendar	TIMESTAMP
java.util.Calendar	DATE



JPA

- `@Entity`
 - Especifica que a classe é uma entidade persistida
- `@Table`
 - Customiza elementos da tabela
 - `@Table(name="nome")`
 - Especifica um nome diferente do da classe para tabela do banco



JPA

- @Id
 - Define a chave primária da entidade
- @GeneratedValue(strategy = tipo)
 - Define como a chave primária é gerada
 - GenerationType.SEQUENCE
 - GenerationType.IDENTITY
 - GenerationType.TABLE
 - GenerationType.AUTO



JPA

- IDENTITY

- Suporte para colunas IDENTITY nos bancos de dados DB2, MySQL, MS SQL Server, Sybase e HypersonicSQL
- Os tipos devem ser long, short ou int

- SEQUENCE

- Utiliza um Sequence para a geração de chaves
- Suporte para DB2, PostgreSQL e Oracle dentre outros
- Nome default da sequence é hibernate_sequence



JPA

- TABLE

- Cria uma tabela no banco para gerenciar as id
- Suportado por todos os bancos

- AUTO

- Estratégia escolhida depende do banco de dados utilizado
- Utilizado para manter a portabilidade



JPA

- @Column(name, unique , nullable, ...)
 - Define propriedade das colunas
 - name – altera o nome da coluna na tabela
 - unique – define o valor da coluna como única
 - nullable – define se a coluna aceita valores nulos



JPA

- `@Temporal(TemporalType)`
 - ➔ Define a coluna dos tipos `java.util.Date` ou `java.util.Calendar`
 - `TemporalType.DATE` – somente data
 - `TemporalType.TIME` – somente hora
 - `TemporalType.TIMESTAMP` – data e hora



JPA

@Entity

public class Produto {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@Column(nullable = false)

private String nome;

@Temporal(TemporalType.DATE)


private Date validade;

private float valor;



JPA

Tabela: produtos

#	Nome	Tipo	Agrupamento (Collation)	Atributos	Nulo
1	id 	int(11)			Não
2	nome	varchar(255)	utf8_unicode_ci		Sim
3	valor	float			Não
4	validade	datetime			Sim

JPA

@Entity

public class Produto {

 @Id

 @GeneratedValue(strategy = GenerationType.IDENTITY)

 private int id;

 @Column(nullable = false)

 private String nome;

 @Temporal(TemporalType.DATE)

 private Date validade;

 private float valor;

 private List<String> marcas;



Classe x Tabela

***Banco de dados não aceitam
colunas multivaloradas***



JPA

- @ElementCollection
 - Define que elemento é uma coleção de elementos
 - Suporta os tipos List ,Set e Map
 - Cria uma tabela auxiliar
- @CollectionTable
 - Permite configurar a tabela auxiliar produzida pelo @ElementCollection



JPA

@Entity

```
public class Produto {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private int id;
```

```
    @Column(nullable = false)
```

```
    private String nome;
```

```
    @Temporal(TemporalType.DATE)
```

```
    private Date validade;
```

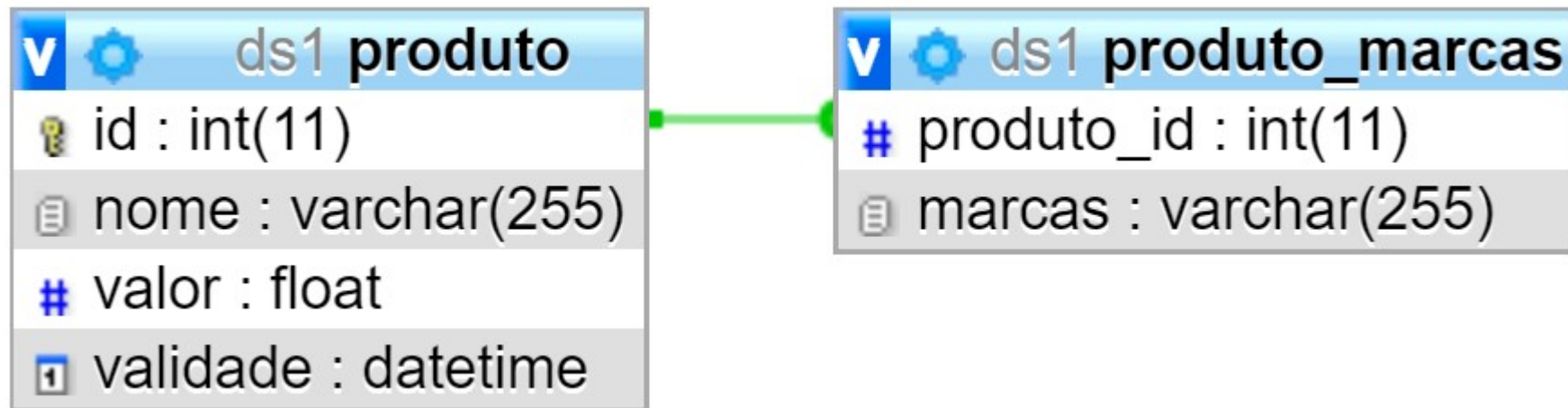
```
    private float valor;
```

```
    @ElementCollection
```

```
    private List<String> marcas;
```



JPA



Spring Data

- Sub-framework feito para lidar com persistência no Spring
- Simplifica o acesso aos mais variados bancos de dados
- DAO são interfaces genéricas implementadas por injeção de dependência
- Composto de vários subprojetos

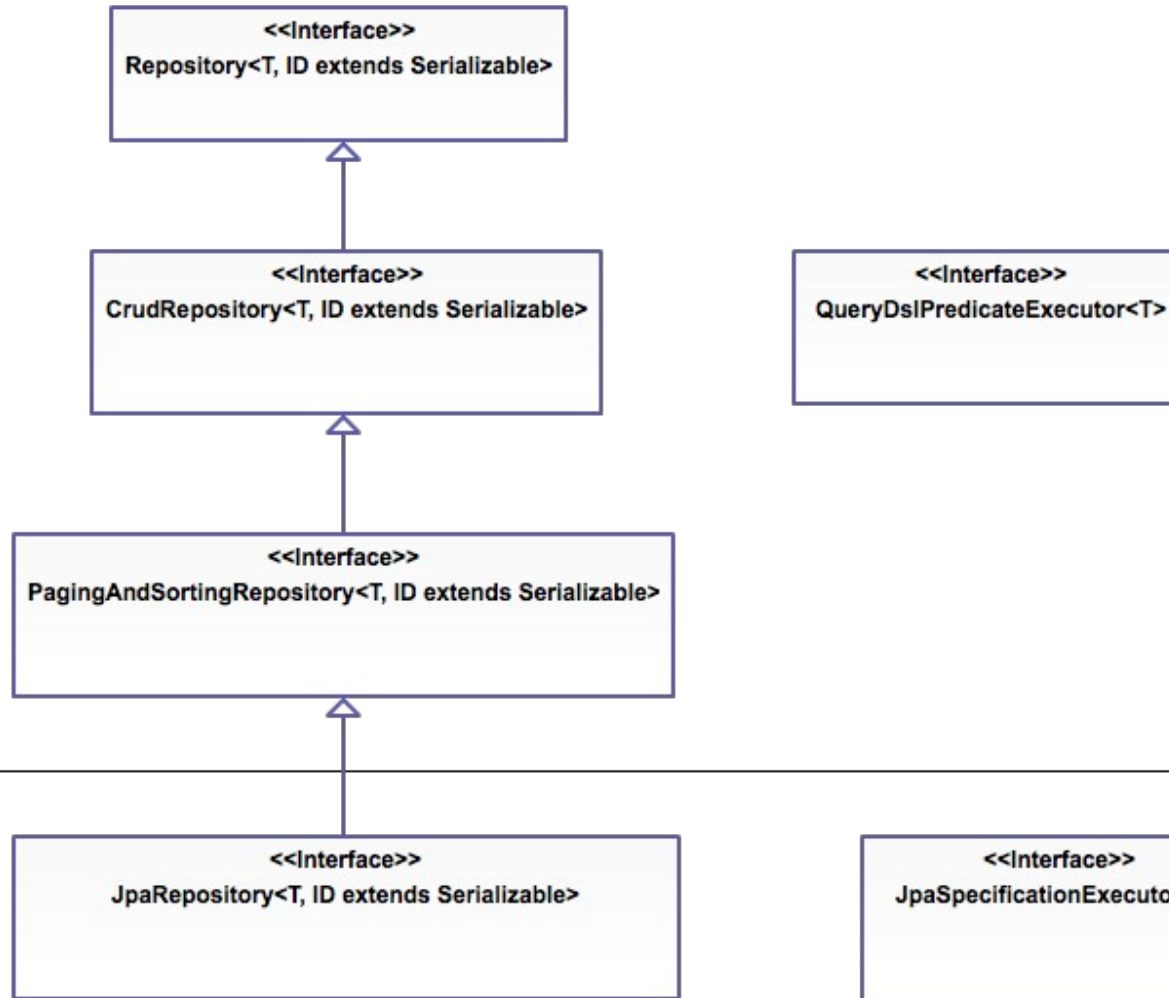


Spring Data - subprojetos

- JPA
- JDBC Extensions
- Hadoop
- GemFire
- Redis
- Riak
- MongoDB
- Neo4j
- Blob
- Commons



Spring Data - Interfaces

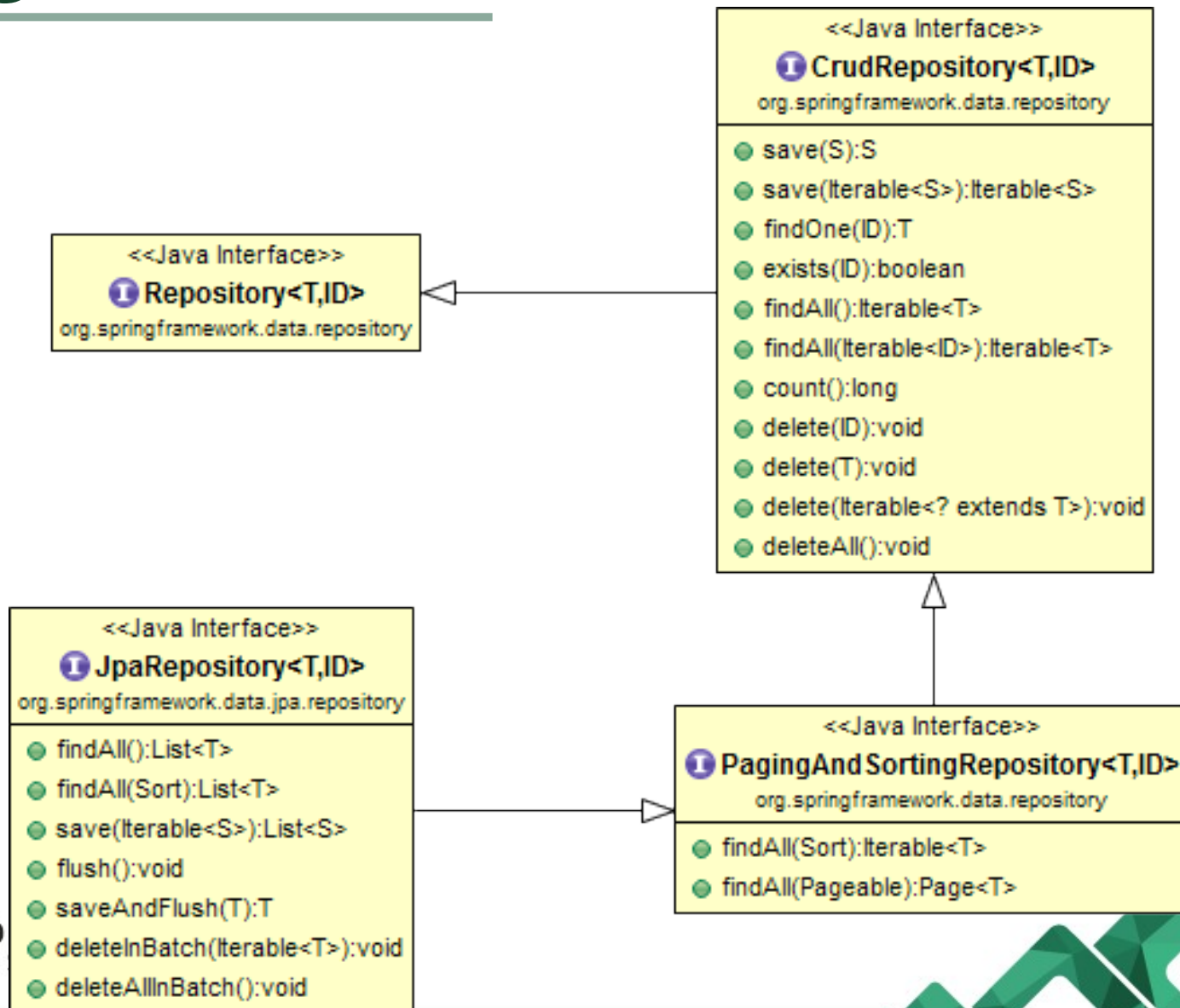


Spring Data Commons

Spring Data JPA



Spring Data - Interfaces



Spring Data - CrudRepository

```
public interface CrudRepository<T, ID> extends Repository<T, ID> {  
    public <S extends T> S save(S s);  
    public <S extends T> Iterable<S> saveAll(Iterable<S> itrbl);  
    public Optional<T> findById(ID id);  
    public boolean existsById(ID id);  
    public Iterable<T> findAll();  
    public Iterable<T> findAllById(Iterable<ID> itrbl);  
    public long count();  
    public void deleteById(ID id);  
    public void delete(T t);  
    public void deleteAll(Iterable<? extends T> itrbl);  
    public void deleteAll();  
}
```



Spring Data – dependência

<https://start.spring.io/>

Generate a Maven Project ▾ with Java ▾ and Spring Boot 1.5.6 ▾

Project Metadata

Artifact coordinates

Group

com.example

Artifact

demo

Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

Web, Security, JPA, Actuator, Devtools...

Selected Dependencies

JPA ×

MySQL ×

Web ×

Generate Project alt + ⌘

Spring Data – dependência

pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
```



Spring Data – dependência

application.properties

```
spring.datasource.url=jdbc:mysql://localhost/ds1?createDatabaseIfNotExist=true  
spring.datasource.username=root  
spring.datasource.password=  
spring.datasource.driverClassName=com.mysql.jdbc.Driver  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.database-platform: org.hibernate.dialect.MySQL5InnoDBDialect
```



Spring Data

@Repository

```
public interface ProdutoDAO extends CrudRepository<Produto, Integer>{  
  
}
```



Spring Data

@RestController

public class Produtos {

@Autowired

ProdutoDAO produtoDAO;

@RequestMapping(path = "/produtos/", method = RequestMethod.GET)

public Iterable<Produto> listar() {

return produtoDAO.findAll();

}

