



SPRING DATA

FAPESC – DESENVOLVEDORES PARA TECNOLOGIA DA INFORMAÇÃO

HERCULANO DE BIASI

herculano.debiasi@unoesc.edu.br



TÓPICOS

- Motivação
- Spring Data
- Repositórios
- Configuração
- *Query builder*
- Anotação @Query
- Paginação de resultados

MOTIVAÇÃO

- Muitos SGBDs relacionais e não relacionais (NoSQL) no mercado



MOTIVAÇÃO

■ JDBC vs. JPA vs MongoDB: Vários modelos diferentes de programação

```
public void insere(TipoProduto tipoProduto){
    Connection con = new ConexaoUtil().criarConexao();
    PreparedStatement stmt;
    try {
        stmt = con.prepareStatement("insert into tip_produto "
            + " (NOMTIPPRODUTO, CLASSIFICACAO) "
            + " values (?,?)");
        stmt.setString(1, tipoProduto.getNome());
        stmt.setString(2, tipoProduto.getClassificacao());
        stmt.executeUpdate();
    } catch (SQLException ex) {
        //Aqui tratar erro
    }
}
```

```
Mongo mongo = new Mongo();
DB db = mongo.getDB("tutorial");
DBCollection employees = db.getCollection("employees");
employees.insert(new BasicDBObject().append("name", "Diana Hightower")
    .append("gender", "f").append("phone", "520-555-1212").append("age", 30));
DBCursor cursor = employees.find();
while (cursor.hasNext()) {
    DBObject object = cursor.next();
    System.out.println(object);
}
```



```
private void inserir(ModelClass objeto) {
    JPAUtil.getEm().getTransaction().begin();
    JPAUtil.getEm().persist(objeto);
    JPAUtil.getEm().getTransaction().commit();
}

public List<TipoProduto> getTipoProduto(String nome){
    return JPAUtil.getEm().createQuery("select p from "
        + "TipoProduto p where p.nome like :nome")
        .setParameter("nome", nome)
        .getResultList();
}
```

MOTIVAÇÃO

- Unificação em torno de um único modelo



The diagram illustrates the motivation for Spring Data by showing a variety of databases grouped into two layers. The top layer, enclosed in a light green oval, contains logos for Hadoop, MongoDB, CouchDB, Neo4j, RAVENDB, and Redis. The bottom layer, also enclosed in a light green oval, contains logos for Apache HBASE, Cassandra, Oracle, MySQL, and PostgreSQL. A small cartoon character is visible in the bottom right corner of the diagram. The text 'Spring Data' is prominently displayed in the center, overlapping both layers.

Spring Data

MOTIVAÇÃO

- Unificação em torno de um único modelo

- MongoDB

```
@Document
public class Pessoa{

    @Id
    private BigInteger id;

    private String nome;

    private int idade;

    //Metodos get/set e construtores
}
```

- Redis

```
@RedisHash("Pessoa")
public class Pessoa{

    @Id
    private BigInteger id;

    private String nome;

    private int idade;

    //Metodos get/set e construtores
}
```

- Jpa (MariaDb)

```
@Entity
public class Pessoa{

    @Id
    private BigInteger id;

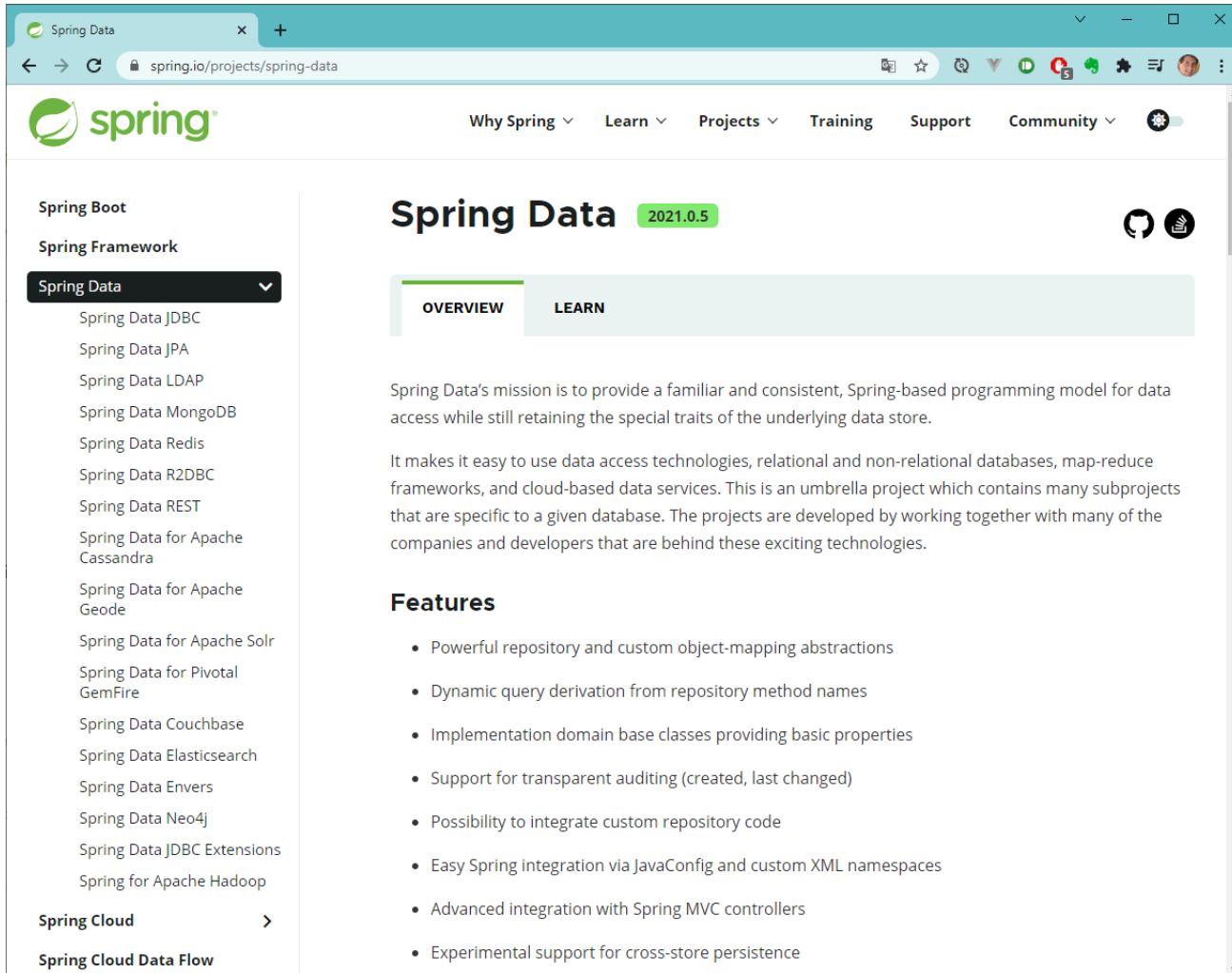
    private String nome;

    private int idade;

    //Metodos get/set e construtores
}
```

SPRING DATA

- Spring Data é o modelo de programação dentro do Spring *Framework* para acesso e manipulação de dados



The screenshot shows the Spring Data website. The left sidebar contains a navigation menu with the following items: Spring Boot, Spring Framework, Spring Data (selected), Spring Data JDBC, Spring Data JPA, Spring Data LDAP, Spring Data MongoDB, Spring Data Redis, Spring Data R2DBC, Spring Data REST, Spring Data for Apache Cassandra, Spring Data for Apache Geode, Spring Data for Apache Solr, Spring Data for Pivotal GemFire, Spring Data Couchbase, Spring Data Elasticsearch, Spring Data Envers, Spring Data Neo4j, Spring Data JDBC Extensions, and Spring for Apache Hadoop. The main content area features the Spring Data logo and the version 2021.0.5. Below the logo, there are tabs for OVERVIEW and LEARN. The OVERVIEW tab is active, displaying the mission statement: "Spring Data's mission is to provide a familiar and consistent, Spring-based programming model for data access while still retaining the special traits of the underlying data store." It also includes a paragraph about the project's scope and a list of features.

Spring Data 2021.0.5

OVERVIEW **LEARN**

Spring Data's mission is to provide a familiar and consistent, Spring-based programming model for data access while still retaining the special traits of the underlying data store.

It makes it easy to use data access technologies, relational and non-relational databases, map-reduce frameworks, and cloud-based data services. This is an umbrella project which contains many subprojects that are specific to a given database. The projects are developed by working together with many of the companies and developers that are behind these exciting technologies.

Features

- Powerful repository and custom object-mapping abstractions
- Dynamic query derivation from repository method names
- Implementation domain base classes providing basic properties
- Support for transparent auditing (created, last changed)
- Possibility to integrate custom repository code
- Easy Spring integration via JavaConfig and custom XML namespaces
- Advanced integration with Spring MVC controllers
- Experimental support for cross-store persistence

SPRING DATA

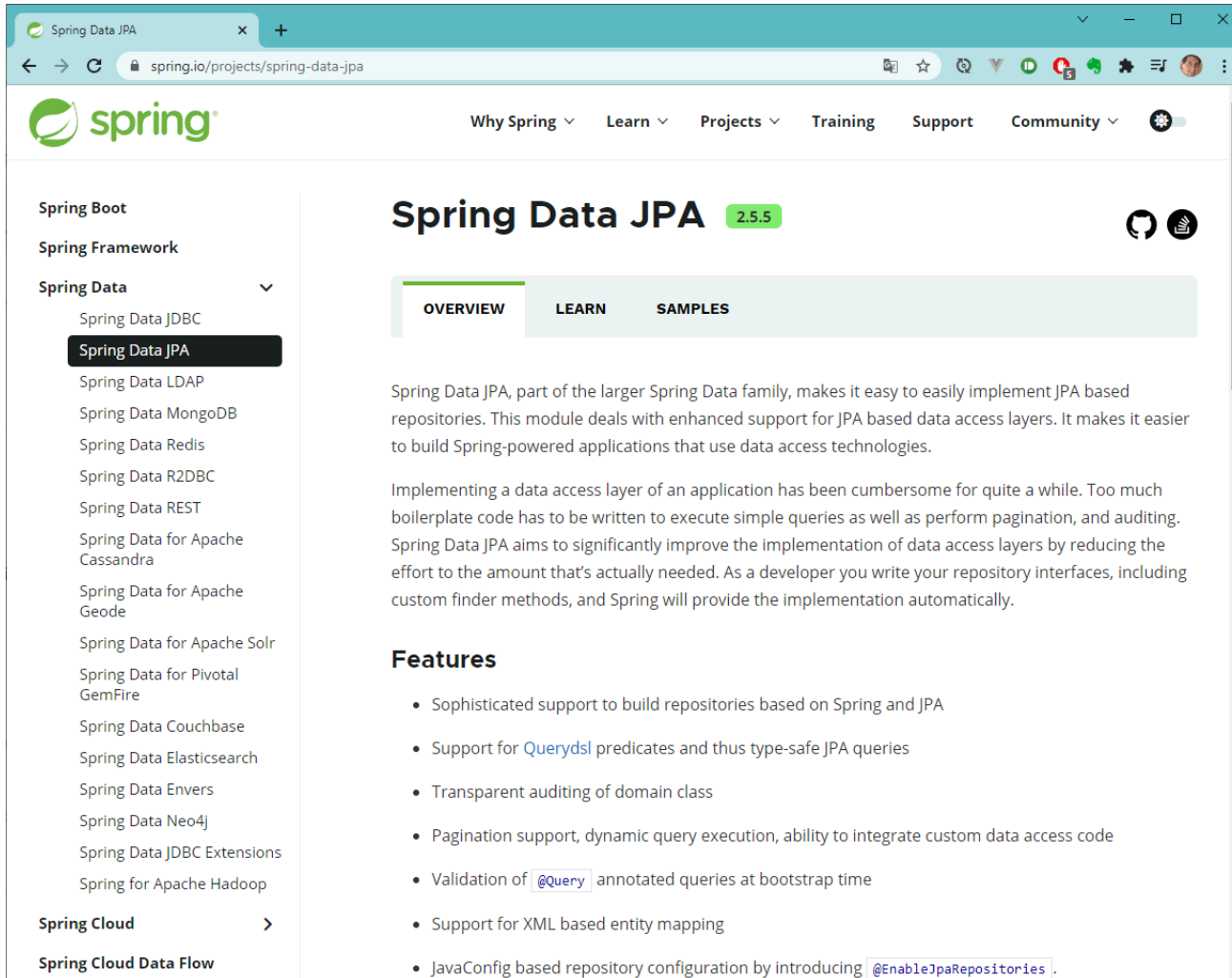
- O objetivo do Spring Data é simplificar o desenvolvimento de aplicações que usam diferentes tecnologias de acesso a dados (bancos relacionais e não relacionais (NoSQL), *frameworks* [map-reduce](#), serviços de dados baseados em nuvem, etc), além do suporte aperfeiçoado a bancos de dados relacionais
- O Spring Data fornece grande suporte para as necessidades de persistência, trazendo já prontas operações CRUD, com as quais pode-se criar, ler, alterar e deletar dados, contando também com funcionalidades como paginação e ordenação de dados
- Alguns subprojetos do Spring Data
 - Spring Data JPA: Para bancos relacionais, como MySQL/MariaDB e PostgreSQL
 - Spring MongoDB: Banco de dados orientado a documentos
 - Spring Data Redis: Banco de dados do tipo chave-valor, em memória

SPRING DATA

- O Spring Data Commons é o projeto que provê a infraestrutura compartilhada por todos os outros projetos do Spring Data
 - Contém *interfaces* e meta modelos de dados para as classes de persistência Java
 - Geração de *queries* (consultas) dinâmicas
 - Suporte a auditoria transparente (criação, última mudança)
 - Possibilidade de integrar código customizado de repositórios

SPRING DATA

- O principal componente do Spring Data é o [Spring Data JPA](#), que implementa o modelo em camadas baseado em repositórios



The screenshot shows the Spring Data JPA project page on the spring.io website. The page is titled "Spring Data JPA" with a version indicator "2.5.5". The left sidebar lists various Spring Data projects, with "Spring Data JPA" highlighted. The main content area has tabs for "OVERVIEW", "LEARN", and "SAMPLES". The "OVERVIEW" tab is active, displaying a description of Spring Data JPA and a list of features.

Spring Data JPA 2.5.5

OVERVIEW | LEARN | SAMPLES

Spring Data JPA, part of the larger Spring Data family, makes it easy to easily implement JPA based repositories. This module deals with enhanced support for JPA based data access layers. It makes it easier to build Spring-powered applications that use data access technologies.

Implementing a data access layer of an application has been cumbersome for quite a while. Too much boilerplate code has to be written to execute simple queries as well as perform pagination, and auditing. Spring Data JPA aims to significantly improve the implementation of data access layers by reducing the effort to the amount that's actually needed. As a developer you write your repository interfaces, including custom finder methods, and Spring will provide the implementation automatically.

Features

- Sophisticated support to build repositories based on Spring and JPA
- Support for [Querydsl](#) predicates and thus type-safe JPA queries
- Transparent auditing of domain class
- Pagination support, dynamic query execution, ability to integrate custom data access code
- Validation of `@Query` annotated queries at bootstrap time
- Support for XML based entity mapping
- JavaConfig based repository configuration by introducing `@EnableJpaRepositories`

REPOSITÓRIOS

- Um repositório no Spring Data é uma abstração destinada a reduzir de forma significativa a quantidade de *boiler plate* necessária para implementar camadas de acesso a dados para diferentes sistemas de armazenamento de dados
- A interface `CrudRepository` provê funcionalidades de CRUD sofisticadas para a entidade que está sendo gerenciada

```
public interface CrudRepository<T, ID extends Serializable>  
    extends Repository<T, ID> {  
  
    <S extends T> S save(S entity);  
  
    T findOne(ID primaryKey);  
  
    Iterable<T> findAll();  
  
    Long count();  
  
    void delete(T entity);  
  
    boolean exists(ID primaryKey);  
  
    // ... more functionality omitted.  
}
```

- ① Saves the given entity.
- ② Returns the entity identified by the given id.
- ③ Returns all entities.
- ④ Returns the number of entities.
- ⑤ Deletes the given entity.
- ⑥ Indicates whether an entity with the given id exists.

REPOSITÓRIOS

- Implementação de um repositório JPA

```
public interface ProdutoRepository  
    extends JpaRepository<Produto, Integer> {
```

```
}
```

```
@Service  
public class ProdutoRegrasImpl implements ProdutoRegras {
```

```
@Autowired  
private ProdutoRepository produtoRepository;
```

```
public void salvar(Produto produto){  
    produtoRepository.save(produto);  
}
```

```
}
```

CONFIGURAÇÃO

■ Dependências no *pom.xml*

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
</dependency>
```

■ Arquivo *application.properties*

```
1 spring.jpa.hibernate.ddl-auto=update
2 spring.jpa.show-sql=true
3 spring.jpa.properties.dialect=org.hibernate.dialect.PostgreSQLDialect
4 spring.jpa.properties.hibernate.temp.use_jdbc_metadata_defaults=false
5 spring.datasource.url=jdbc:postgresql://localhost:5432/eleicao
6 spring.datasource.username=postgres
7 spring.datasource.password=postgres
8 spring.datasource.driverClassName=org.postgresql.Driver
```

CONFIGURAÇÃO

- Há 5 opções do Spring para gerenciar a evolução do banco de dados
 - `update`: Sempre que a aplicação iniciar o Spring Data verifica se as classes entidades estão de acordo com o banco – caso não estejam, será feito um *update* no banco adicionando novas colunas na tabela dessa entidade
 - É importante notar que o *update* não remove ou renomeia colunas
 - Por isso, não é recomendado de jeito algum utilizar o Spring Data para gerenciar a evolução do seu banco, para isso existem ferramentas especializadas, por exemplo, o FlyWay
 - `create`: Sempre que a aplicação for iniciada o Spring Data vai apagar tudo e recriar novamente do zero
 - `create-drop`: Semelhante ao `create`, mas sempre que a aplicação é parada ele apaga tudo que foi criado
 - `validate`: Faz uma validação sempre que a aplicação inicia e verifica se o banco confere com as suas classes de entidade
 - `none`: Basicamente é nenhuma das opções acima, ou seja, significa que não se quer que o Spring Data faça alterações no banco – é a recomendada para produção, para que não ocorra problemas do Spring modificar o banco

QUERY BUILDER

- *Query builder*: Construção de uma *query* utilizando somente o nome do método

```
public interface FilmeRepository
    extends JpaRepository<Filme, Long> {

    List<Filme> findByName(String nome);
}
```

QUERY BUILDER

■ Métodos do query builder

Keyword	Sample
And	<code>findByLastnameAndFirstname</code>
Or	<code>findByLastnameOrFirstname</code>
Between	<code>findByStartDateBetween</code>
LessThan	<code>findByAgeLessThan</code>
GreaterThan	<code>findByAgeGreaterThan</code>
IsNull	<code>findByAgeIsNull</code>
IsNotNull,NotNull	<code>findByAge(Is)NotNull</code>
Like	<code>findByFirstnameLike</code>
NotLike	<code>findByFirstnameNotLike</code>
OrderBy	<code>findByAgeOrderByLastnameDesc</code>
Not	<code>findByLastnameNot</code>
In	<code>findByAgeIn(Collection<Age> ages)</code>
NotIn	<code>findByAgeNotIn(Collection<Age> age)</code>

ANOTAÇÃO @QUERY

- Com a anotação @Query é possível utilizar **JPQL** (Java Persistence Query Language) para a escrita de consultas personalizadas

```
public interface UserRepository extends JpaRepository<User, Long> {  
  
    @Query("select u from User u where u.firstname = :firstname or u.lastname = :lastname")  
    User findByLastnameOrFirstname(@Param("lastname") String lastname,  
                                    @Param("firstname") String firstname);  
}
```

PAGINAÇÃO DE RESULTADOS

- Com as classes `Page` e `Pageable` pode-se fazer paginação nos resultados das consultas de maneira bem fácil

```
public interface ClienteRepository
    extends JpaRepository<Cliente, Long> {

    Page<Cliente> findByNomeLike(String nome, Pageable pageable);

}
```

- Utilização

```
Pageable paginacao = new PageRequest(page, pageSize);
Page<Cliente> clientes = clienteRepository.findAll(paginacao);
List<Cliente> clientesList = clientes.getContent();
```