Banco de Dados

Profª. Ana Luiza Scharf
ana.scharf@ifsc.edu.br

LABORATÓRIO – Introdução a JPA + Spring Boot-Parte 01

Objetivo

Este laboratório tem como objetivo apresentar os primeiros passos para trabalhar com **Spring Boot + Spring Data JPA**, criando uma aplicação Java capaz de:

- conectar-se a um banco MySQL;
- traduzir uma tabela para uma entidade Java;
- realizar operações CRUD (Create, Read, Update, Delete);
- usar Lombok para reduzir código repetitivo.

Criação do Projeto no Spring Initializr

Para iniciar uma aplicação usando Spring Boot:

1. Acesse: https://start.spring.io
2. Configure:
    - **Project:** Gradle – Groovy
    - **Language:** Java
    - **Spring Boot:** 3.5.3
    - **Group:** ads.bcd
    - **Artifact:** Lab_BCD_01
    - **Java:** 21
3. Adicione as dependências:
    - Spring Data JPA
    - MySQL Driver
    - Spring Boot DevTools
4. Faça o download do arquivo .zip, extraia-o e abra no IntelliJ ou VS Code.

O DevTools permite hot reload, reiniciando automaticamente quando arquivos são alterados.

Servidor MySQL

Para executar esse exemplo, é necessário que tenha um servidor MySQL disponível. Você pode subir um rapidamente dentro de um contêiner com o Docker. **Dica: no Windows, o Docker precisa estar rodando — ou seja, é necessário abrir a ferramenta antes de usar.** .Basta executar o comando abaixo:

```
docker run -d --rm -p 3306:3306 -e MYSQL_ROOT_PASSWORD=senhaRoot \
-e MYSQL_DATABASE=bcd -e MYSQL_USER=aluno -e MYSQL_PASSWORD=aluno \
-e MYSQL_ROOT_HOST='%' --name meumysql mysql/mysql-server:latest
```

Cabe lembrar que sempre que o contêiner for parado, ele será excluído (opção *–rm*) e todos os dados serão perdidos. Se quiser que os dados continuem mesmo depois da parada e exclusão do contêiner, passe o parâmetro *-v $(pwd)/db_data:/var/lib/mysql*, que fará o mapeamento do diretório usado pelo MySQL no contêiner para um diretório no computador hospedeiro.

Configuração de Conexão com o Banco

Arquivo `application.properties`:

```
spring.datasource.url=jdbc:mysql://localhost:3306/bcd
spring.datasource.username=aluno
spring.datasource.password=aluno
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=create
```

Configurações do `application.properties`

- `spring.datasource.url=jdbc:mysql://localhost:3306/bcd`
  - Define a URL de conexão com o banco de dados.
  - MySQL executando em `localhost`, porta `3306`.
  - `bcd` é o nome do schema.
- `spring.datasource.username=aluno`
  - Usuário do banco de dados MySQL.
- `spring.datasource.password=aluno`
  - Senha associada ao usuário do banco.
- `spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver`
  - Driver JDBC utilizado para comunicação com o MySQL.
- `spring.jpa.hibernate.ddl-auto=create`
  - Hibernate recria as tabelas a cada execução.
  - Baseado nas entidades JPA (`@Entity`).
  - Indicado para testes e desenvolvimento.

Opções do ddl-auto

- **create**: recria o esquema a cada execução (ideal para testes).
- **update**: atualiza sem apagar dados existentes.
- **none**: não modifica o esquema (ideal para produção).

Lombok

O Lombok elimina código repetitivo, como getters, setters, construtores, toString, equals e hashCode.
No arquivo `build.gradle`:

```
plugins {
    id 'java'
    id 'org.springframework.boot' version '3.5.3'
    id 'io.spring.dependency-management' version '1.1.7'
    id "io.freefair.lombok" version "8.4"
}
```

Mapeamento da Tabela ALUNO para Java (JPA)

Estrutura SQL:

```
ALUNO (
    matricula BIGINT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100),
    matriculaAtiva BOOLEAN,
    dataInicio DATE
)
```

Classe Java equivalente:

```java
package ifsc.eng.Lab_Ana_BCD;


import jakarta.persistence.*;
import java.time.LocalDate;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.AllArgsConstructor;

@Entity // Diz ao JPA que esta classe é uma tabela
@Data // Gera Getters, Setters, ToString, etc.
@NoArgsConstructor // Construtor padrão (OBRIGATÓRIO para o JPA)
@AllArgsConstructor // Construtor com todos os campos (útil para criar objetos)
public class Aluno {

    @Id // Marca como Chave Primária
    @GeneratedValue(strategy = GenerationType.IDENTITY) // Auto-incremento no banco
    private Long matricula;

    private String nome;

    private Boolean matriculaAtiva;

    private LocalDate dataInicio;
}
```

Repository

```java
package ifsc.eng.Lab_Ana_BCD;

import ifsc.eng.Lab_Ana_BCD.Aluno;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

// CrudRepository<TIPO_DA_ENTIDADE, TIPO_DO_ID>
@Repository
public interface AlunoRepository extends CrudRepository<Aluno, Long> {

    // O Spring Data JPA já implementou para você:
    // save(), findById(), findAll(), delete(), count(), etc.

}
```

O Spring Data JPA fornece automaticamente:

- save()
- findById()
- findAll()
- delete()
- count()

Laboratório CRUD com CommandLineRunner

```java
package ifsc.eng.Lab_Ana_BCD;


import ifsc.eng.Lab_Ana_BCD.Aluno;
import ifsc.eng.Lab_Ana_BCD.AlunoRepository;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

import java.time.LocalDate;
import java.util.Optional;

@SpringBootApplication
public class LabAnaBcdApplication {

  public static void main(String[] args) {
    SpringApplication.run(LabAnaBcdApplication.class, args);
  }


  // O Spring injeta (passa) automaticamente o AlunoRepository para este método.
  @Bean
  public CommandLineRunner brincarComDados(AlunoRepository repository) {
    return args -> {

      // --- 1. CREATE: Inserindo (save) ---
      System.out.println("--- 1. INSERT: Salvando Alunos ---");

      // Usamos o construtor gerado pelo Lombok @AllArgsConstructor
      // null para 'matricula' (PK), pois ela é gerada pelo banco.
      Aluno alunoA = new Aluno(null, "Mariana Luz", true, LocalDate.of(2024, 2, 1));
      Aluno alunoB = new Aluno(null, "Pedro Rocha", true, LocalDate.of(2023, 8, 15));

      repository.save(alunoA);
      repository.save(alunoB);

      System.out.println("Aluno A salvo. Matrícula: " + alunoA.getMatricula());
      System.out.println("Aluno B salvo. Matrícula: " + alunoB.getMatricula());

      // --- 2. READ ALL: Buscando todos (findAll) ---
      System.out.println("\n--- 2. SELECT ALL: Todos os Alunos ---");
      repository.findAll().forEach(System.out::println);

      // --- 3. READ BY ID: Buscando por Chave Primária (findById) ---
      System.out.println("\n--- 3. SELECT BY ID: Buscando aluno 1 ---");
      Optional<Aluno> encontrado = repository.findById(1L);

      encontrado.ifPresent(a -> {
        System.out.println("Aluno encontrado: " + a);

        // --- 4. UPDATE: Atualizando um registro ---
        System.out.println("\n--- 4. UPDATE: Mudando o status do aluno 1 ---");
        a.setMatriculaAtiva(false); // Usa o Setter gerado pelo Lombok
        repository.save(a); // O save() atualiza se a PK já existe

        System.out.println("Status atualizado! Novo dado: " + repository.findById(1L).get());
      });

      // --- 5. DELETE: Removendo um registro ---
      if (alunoB.getMatricula() != null) {
        System.out.println("\n--- 5. DELETE: Removendo o aluno B ---");
        repository.deleteById(alunoB.getMatricula());
        System.out.println("Aluno B removido. Total de alunos restantes: " + repository.count());
      }

    };
  }
}
```

Resumo

Este laboratório introduziu:

- criação de projeto Spring Boot com JPA;
- configuração de conexão com MySQL;
- uso de Lombok;
- tradução de tabela para entidade Java;
- operações CRUD via Repository;
- execução automática usando CommandLineRunner.