



Certified Tech Developer

The Ultimate Degree

Configurando o Hibernate com o Spring Boot

Vamos ver um exemplo!

1. No pom.xml de nosso projeto Spring Boot, devemos adicionar as seguintes dependências:

- **spring-boot-starter-data-jpa:** inclui a API JPA, a implementação JPA, JDBC e outras bibliotecas. Como a implementação padrão do JPA é o Hibernate, essa dependência também está inclusa.
- **com.h2database:** para poder realizar um teste rapidamente, podemos adicionar o H2 (um banco de dados na memória muito leve). Em **application.properties** habilitamos o console do banco de dados H2 para acessá-lo por meio de uma UI.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>

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

<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>
```

2. Além de incluir as dependências no pom.xml, devemos incluir as propriedades do banco de dados no arquivo **application.properties**. Este é um arquivo de configuração e é onde o **Spring** relaciona nosso projeto ao banco de dados que queremos usar. Portanto, para se conectar ao banco de dados, devemos disponibilizar algumas informações no arquivo application.properties:

url: url do serviço e o nome da base de dados.

driveClassName: indica o driver/lib para conectar o Java ao H2.

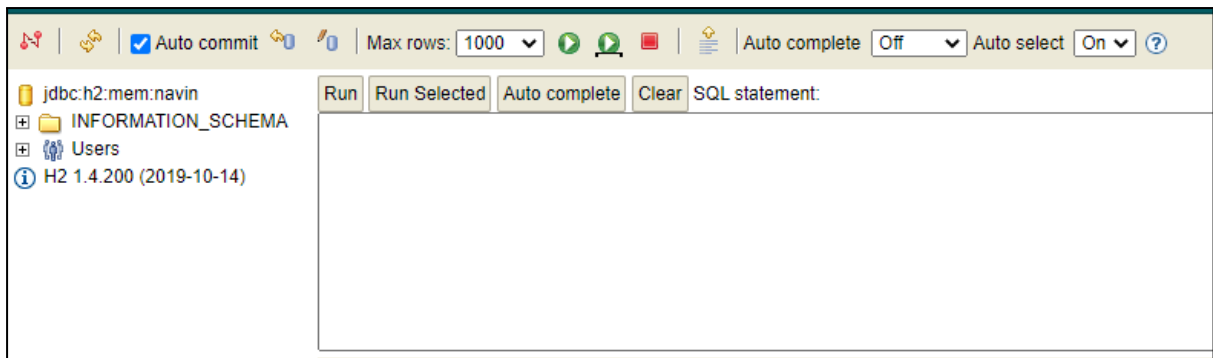
```
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.url=jdbc:h2:mem:navin
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.show-sql
spring.jpa.hibernate.ddl-auto=create-drop
```

show-sql: exibe no console as instruções feitas no banco de dados.

ddl-auto: indica que quando a aplicação é executada pela primeira vez, deve-se criar todas as tabelas na base de dados automaticamente, caso já exista a tabela a mesma será eliminada e criada novamente.

3. Executamos a aplicação e verificamos se ela estabelece a conexão no navegador.

The screenshot shows the H2 console web interface in a browser. The address bar shows 'localhost:8080/h2-console/test.do?sessionId=...'. The interface includes a language dropdown set to 'English' and links for 'Preferences', 'Tools', and 'Help'. The 'Login' section has a 'Saved Settings' dropdown set to 'Generic H2 (Embedded)'. Below it, the 'Setting Name' is also 'Generic H2 (Embedded)' with 'Save' and 'Remove' buttons. The configuration fields are: 'Driver Class' set to 'org.h2.Driver', 'JDBC URL' set to 'jdbc:h2:mem:navin', 'User Name' set to 'sa', and an empty 'Password' field. At the bottom of the configuration section are 'Connect' and 'Test Connection' buttons. A red arrow points to the 'Test Connection' button. Below the configuration section, a green banner displays the message 'Test successful'.



4. Criamos a entidade Student:

```
@Entity
public class Student {

    @Id
    @GeneratedValue(strategy=generationType.SEQUENCE)
    private Long id;
    private String dni;
    private String name;
    private String lastName;
}
```

5. Para acessar os dados com o Spring Data, só precisamos criar os repositórios. Por exemplo, para criar o repositório para a classe Student, apenas definimos a interface, **IStudentRepository** que se estende de **JpaRepository**.

```
public interface IStudentRepository extends JpaRepository <Student , Long> {
}
```

6. O que fizemos foi criar uma interface que se estende de **JpaRepository<T, ID>** onde:

- **T**: deve ser o nome da classe que será utilizada para criar o repositório (em nosso exemplo seria Student).
- **ID**: o tipo de dado que será usado como identificador ou chave primária no banco de dados (em nosso caso, Long).

Com esse Spring Data, serão criadas as operações CRUD para a entidade Student. Isso significa que agora podemos: criar, ler, atualizar e excluir um aluno (Student) do banco de dados.

7. Criamos um serviço para o repositório, no qual é injetado por meio do construtor:

```
public class StudentService {  
    private final IStudentRepository studentRepository;  
    public StudentService(IStudentRepository studentRepository)  
    {  
        this.studentRepository = studentRepository;  
    }  
}
```

8. Executamos a aplicação e conectamos novamente no banco de dados (<http://localhost:8080/h2-console>).

The screenshot shows the H2 database console interface. On the left, a tree view displays the database structure. A red box highlights the 'STUDENT' table, which has columns: ID, DNI, LASTNAME, and NAME. Another red box highlights the 'HIBERNATE_SEQUENCE' sequence. On the right, the SQL statement 'SELECT * FROM STUDENT' is entered in the 'SQL statement:' field. Below the statement, the results are displayed as a table with columns: ID, DNI, LASTNAME, and NAME. The status bar at the bottom indicates '(no rows, 5 ms)'. Two red callout boxes provide additional context: one points to the STUDENT table columns with the text 'Foi criada uma tabela com o nome e colunas que representam os atributos da classe.', and another points to the HIBERNATE_SEQUENCE sequence with the text 'Também foi criada uma sequence.'

localhost:8080/h2-console/login.do?jsessionid=33d71862cf14c023c

Auto commit Max rows: 1000 Auto complete

Run Run Selected Auto complete Clear SQL statement:

jdbc:h2:mem:navin

STUDENT

- ID
- DNI
- LASTNAME
- NAME

Indexes

INFORMATION_SCHEMA

Sequences

HIBERNATE_SEQUENCE

Users

H2 1.4.200 (2019-10-14)

SELECT * FROM STUDENT

Foi criada uma tabela com o nome e colunas que representam os atributos da classe.

Também foi criada uma sequence.

SELECT * FROM STUDENT;

ID	DNI	LASTNAME	NAME
----	-----	----------	------

(no rows, 5 ms)