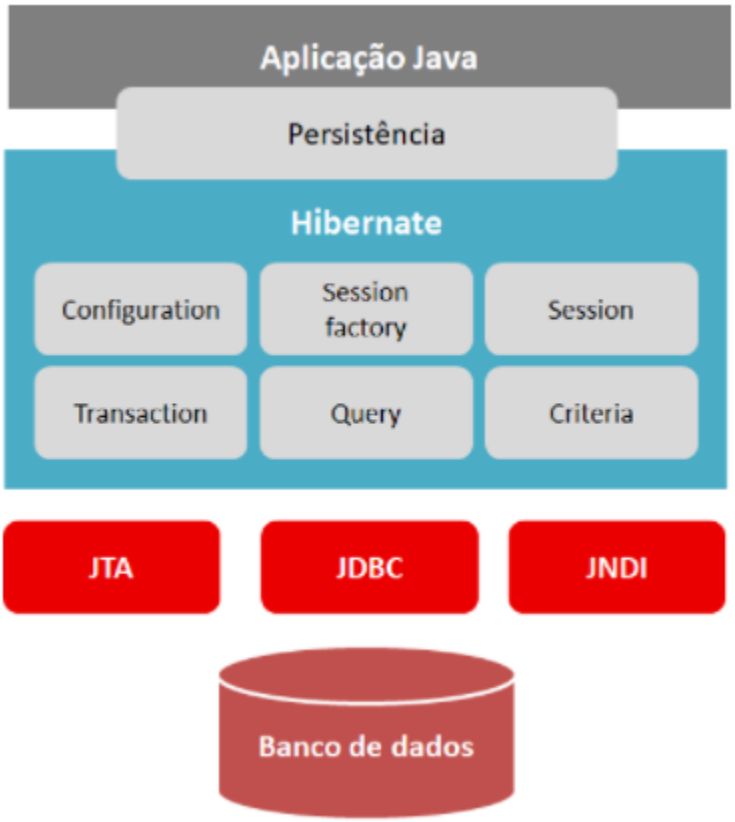
Certified Tech Developer
The Ultimate Degree

Hibernate

O Hibernate é um framework ORM que ajuda a realizar a persistência dos dados; Especificamente, ele possibilita a persistência de dados no que se refere a bancos de dados relacionais (RDBMS).

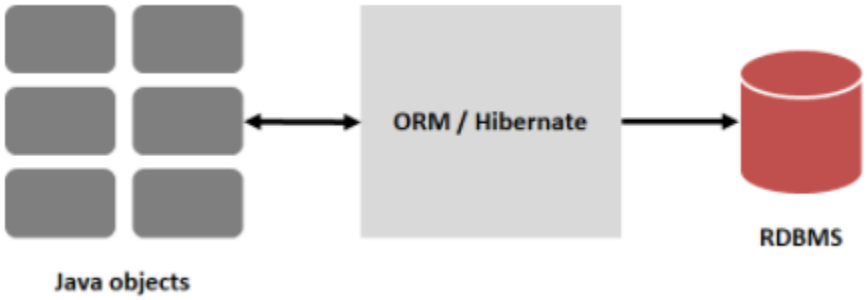


Mas… o que é **persistência**? É fazer com que nossos dados de aplicação sobrevivam ao processo da sua aplicação. Em termos de Java, gostaríamos que o estado de nossos objetos perpetuem além do escopo da JVM para que o mesmo estado esteja disponível posteriormente.

O Hibernate mapeia as classes no Java para tabelas de banco de dados e fornece mecanismos para consultar os dados.

O **mapeamento** é feito por meio de uma configuração (**.xml**) ou de **anotação**.

Se uma mudança no banco de dados for necessária, apenas o arquivo de configuração ou as anotações precisam ser alterados.



Por que o Hibernate?

- É open source: utiliza poucos recursos;
- Possui alto desempenho: é rápido porque utiliza cache internamente no framework;
- Possui queries independentes do DB: com HQL (Hibernate Query Language) é possível gerar queries independentes do DB utilizado. Antes do Hibernate, se o banco de dados fosse alterado, era necessário alterar todas as queries da aplicação;
- Cria tabelas automaticamente;
- Simplifica joins complexos: é possível buscar dados de múltiplas tabelas;
- Fornece estatísticas por meio da query cache e sobre o status do banco de dados.

Configurando o Hibernate com o Spring Boot

Vamos ver um exemplo!

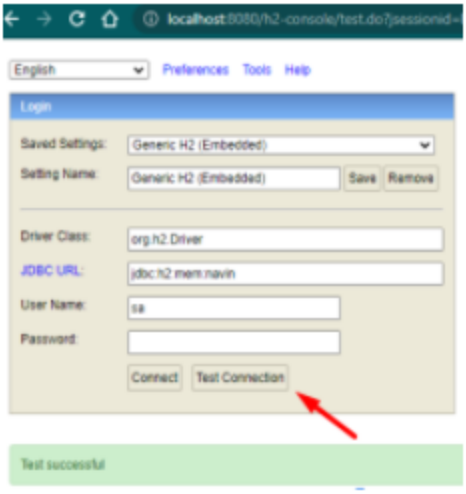
1. No pom.xml de nosso projeto Spring Boot, devemos adicionar as seguintes dependências:

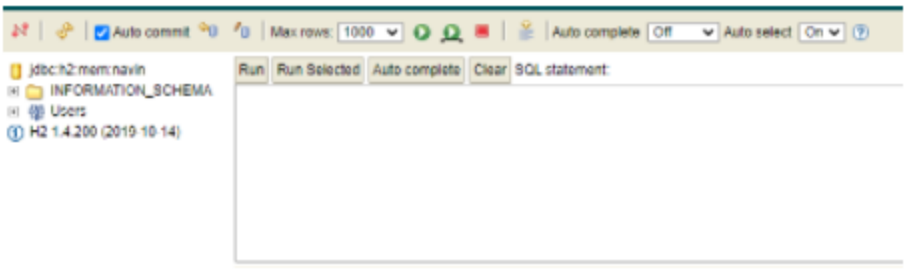
- **spring-boot-starter-data-jpa:** inclui a API JPA, a implementação JPA, JDBC e outras bibliotecas. Como a implementação padrão do JPA é o Hibernate, essa dependência também está inclusa.
- **com.h2database:** para poder realizar um teste rapidamente, podemos adicionar o H2 (um banco de dados na memória 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:

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





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 IStudentRepositoroy studentRepository;

   public StudentService(IStudentRepository studentRepository)

   {


   this.studentRepository = studentRepository;

   }

}
```

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