



## Back End I

# Configurando Hibernate con Spring Boot

## ¡Hagamos un ejemplo!

1) En el **pom.xml** de nuestro proyecto Spring Boot debemos agregar las siguientes **dependencias**:

- **spring-boot-starter-data-jpa**: esta incluye la API JPA, la implementación de JPA, JDBC y otras librerías. Como la implementación por defecto de JPA es Hibernate, esta dependencia también lo trae incluido.
- **com.h2database**: para hacer una prueba rápida, podemos agregar H2 (una base de datos en memoria muy liviana). En **application.properties** habilitamos la consola de la BD H2 para poder acceder a ella a través de una 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) Además de agregar las dependencias en el pom.xml, se deben agregar las properties de la base de datos en el archivo **application.properties**. Este es un archivo de configuración y es donde **Spring** relaciona nuestro proyecto con la base de datos que deseamos utilizar. Entonces, para poder conectarnos con la base de datos debemos indicar en el archivo application.properties lo siguiente:

**url:** URL donde está el servicio de tu MySQL y el nombre de la base de datos.

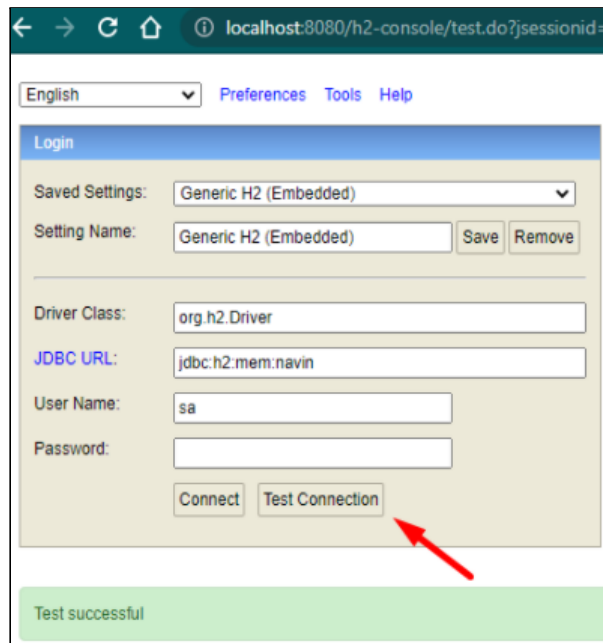
**driverClassName:** indica el driver/lib para conectar Java a MySQL.

```
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
```

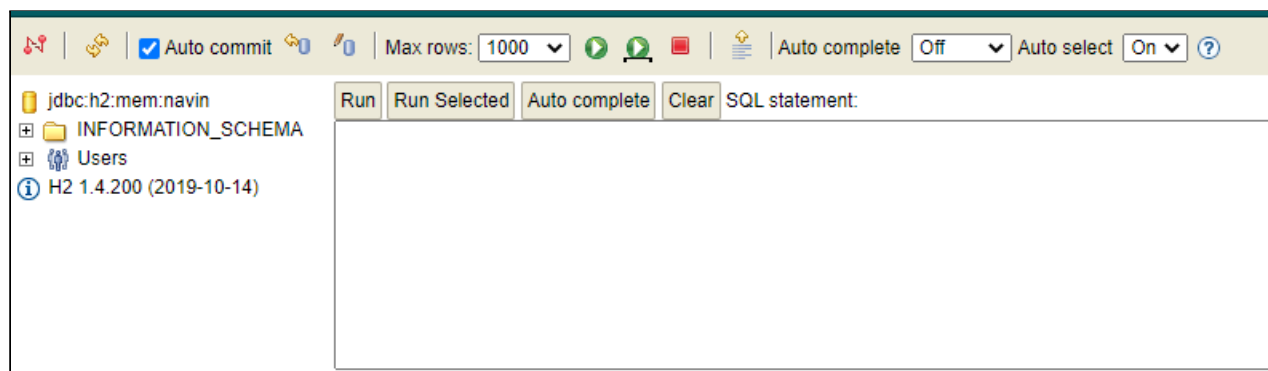
**show-sql:** imprime en la consola las instrucciones hechas en la base de datos.

**ddl-auto:** indica que cuando se ejecuta la aplicación por primera vez, crea en la base de datos todas las tablas automáticamente. Si existieran, las elimina.

3) Corremos la aplicación y verificamos si establece la conexión en el navegador.



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



The screenshot shows the H2 console web interface. The top toolbar includes icons for undo, redo, and other actions, along with settings like 'Auto commit' (checked), 'Max rows: 1000', 'Auto complete' (Off), and 'Auto select' (On). The left sidebar shows the database 'jdbc:h2:mem:navin' with a tree view containing 'INFORMATION\_SCHEMA', 'Users', and 'H2 1.4.200 (2019-10-14)'. The main area has buttons for 'Run', 'Run Selected', 'Auto complete', and 'Clear', followed by an 'SQL statement:' input field.

4) Creamos la entidad Student:

```
@Entity
public class Student {

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

```
private String lastName;  
}
```

5) Para el acceso a datos con Spring Data, solo debemos crear los repositorios. Por ejemplo, para crear el repositorio para la clase Student, solo definimos la interfaz, **IStudentRepository** que extienda de **JpaRepository**:

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

Lo que hicimos fue crear una interfaz que extiende de **JpaRepository<T, ID>** donde:

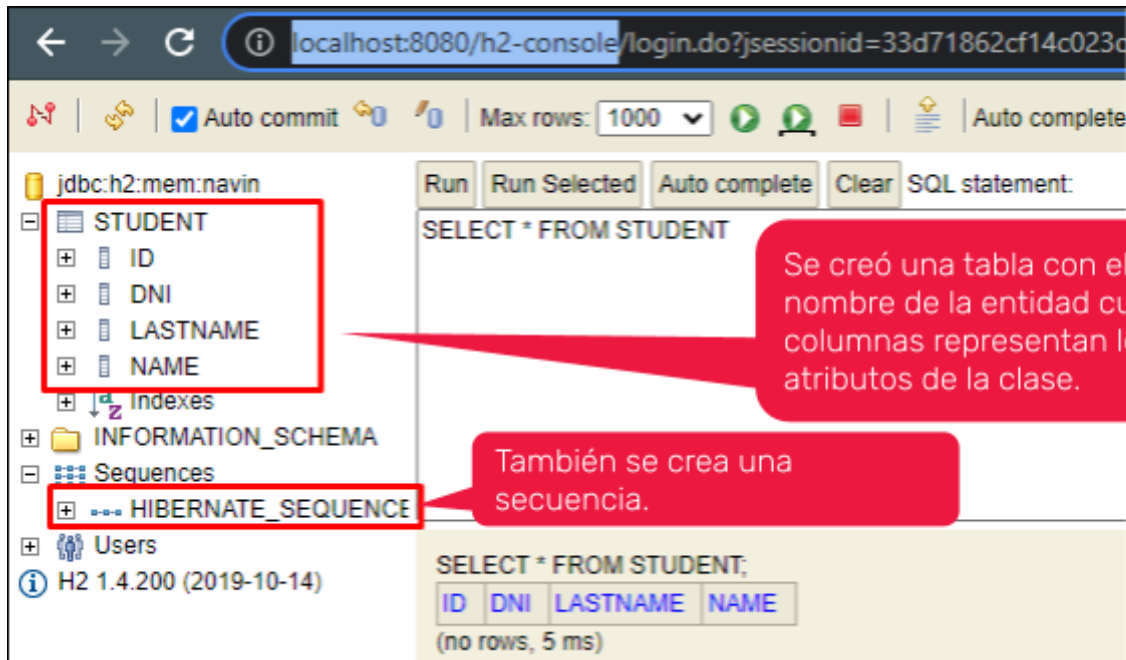
- ▶ **T**: debe ser la clase de la cual crearemos el repositorio (en nuestro ejemplo sería Student).
- ▶ **ID**: el tipo usado como identificador o clave primaria en la base de datos (en nuestro caso, Long).

Con esto Spring Data creará las operaciones CRUD para la entidad Student. Esto quiere decir que ya podríamos: crear, leer, actualizar, y eliminar un Student de la BD.

6) Creamos un servicio al cual se le inyecta el repositorio a través del constructor:

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

7) Ejecutamos la aplicación y volvemos a ingresar a la base de datos  
(<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 with 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 show the columns 'ID', 'DNI', 'LASTNAME', and 'NAME' with the message '(no rows, 5 ms)'. Two red callout boxes provide Spanish annotations: one pointing to the 'STUDENT' table and another pointing to the 'HIBERNATE\_SEQUENCE' sequence.

Se creó una tabla con el nombre de la entidad cuyas columnas representan los atributos de la clase.

También se crea una secuencia.