Theory: H2 database

22 minutes 0 / 5 problems solved

Skip this topic Start practicing

758 users solved this topic. Latest completion was about 1 hour ago.

§1. Why use H2 as a database

H2 is an open-source lightweight relational database for the Java platform. It is provided by Spring Boot by default and is easy and quick to set up in comparison with other databases. Using H2 as a database in training projects can help you to concentrate on learning Spring Boot and java features instead of database-specific setup.

However, it also requires a minimal configuration to be done. The config you will create in this topic can then be simply reused in other studying projects with slight changes.

There are two main ways to store data in the H2 database. You can use it as an **in-memory database** and keep your data in memory while H2 is running. After you stop using H2 or the application, all the data will be lost.

The other way is to **store the data on your drive** at the file system and reuse it when you need it. We will use this type of database here to learn different options available for it.

§2. The basic project

First, we need to create a basic Spring Boot project. Visit the Spring Initializr site and create a basic project topic. We need to add some additional dependencies:

- spring-boot-starter-data-jpa
- spring-boot-starter-jdbc
- spring-boot-starter-web
- com.h2database:h2.

Your **build.gradle** with the dependencies should look like the one below.

```
dependencies {
    implementation 'org.springframework.boot:sweb'
    implementation 'org.springframework.boot:spring-boot-starter-jdbc'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    runtimeOnly 'com.h2database:h2'
    testImplementation('org.springframework.boot:spring-boot-starter-
test') {
        exclude group: 'org.junit.vintage', module: 'junit-vintage-engine'
    }

}
```

Now we are ready to set up the H2 database.

§3. Datasource parameters

The h2 parameters are configured in the **application.properties** file. First, we set the datasource parameters, which define where our database is stored, what type it is, and how the application can connect to it.

Let's start with the following string in the **application.properties** file:

```
spring.datasource.driverClassName=org.h2.Driver
```

This parameter sets the class of the datasource driver. If you tap on the value, you will be able to see that it is a regular Java class in the `org.h2` package:

```
public class Driver implements java.sql.Driver, JdbcDriverBackwardsCompat
```

The next parameter sets the place where your db files would be stored. As H2 commonly is used to manage an in-memory database, all the files will be placed on your drive. For example, the following string creates a database named "sampleDB" in the current user's directory:

```
spring.datasource.url=jdbc:h2:file:~/sampleDB
```

The last two options describe the database username and password.

```
spring.datasource.username=sa
spring.datasource.password=abc123
```

The user "sa" is the default one. The default password is an empty string, so this parameter is not necessary.

You can also set the database type you are about to use. Otherwise, it will be set automatically.

```
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

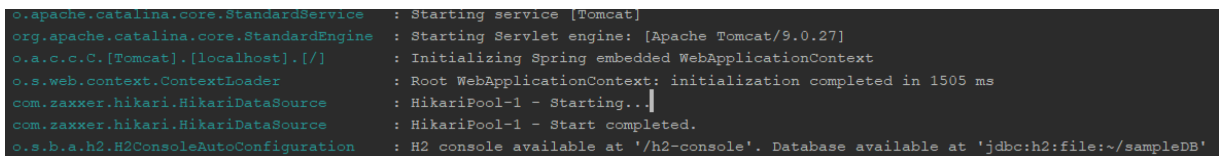
§4. H2 console

Now we enable the h2 console so it can run our application and use the console functionality:

```
spring.h2.console.enabled=true
spring.h2.console.settings.trace=false
```

The `trace` parameter value above hides all the H2 console trace in the application log. You can set it to `true` and look at how console actions are traced.

If you now run the application, you will see how the sampleDB is created and the console is enabled:



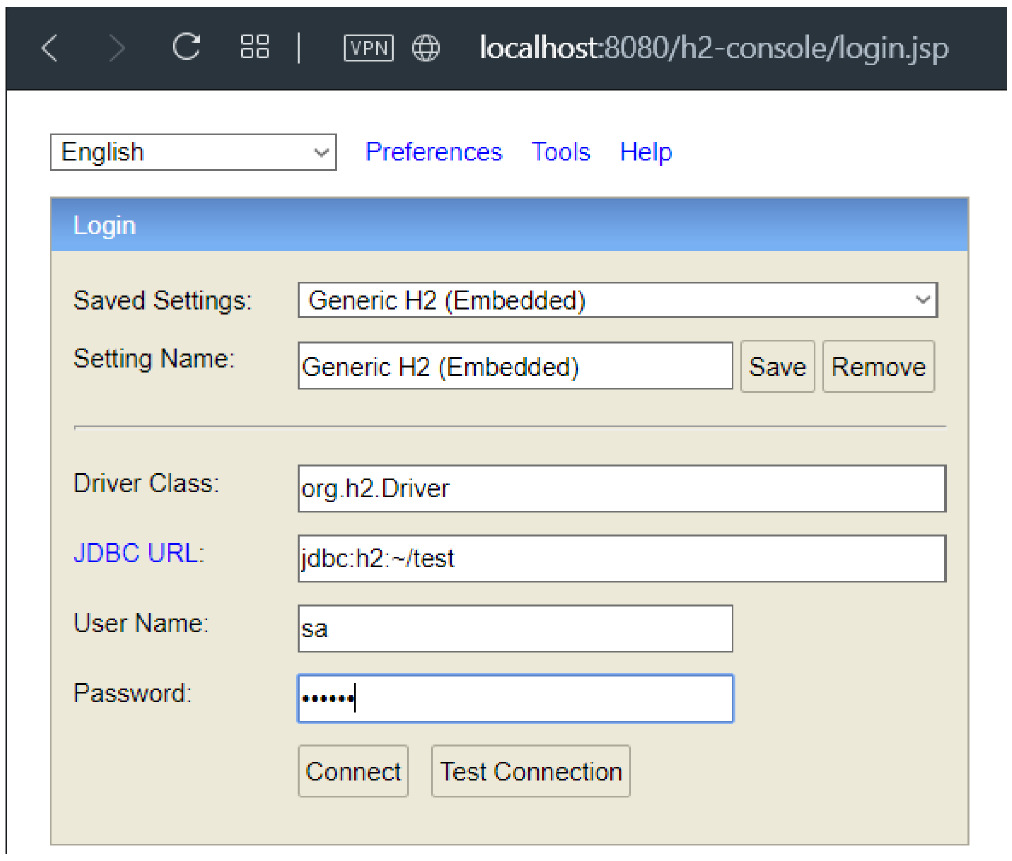
```
o.apache.catalina.core.StandardService   : Starting service [Tomcat]
org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.27]
o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization completed in 1505 ms
com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
o.s.b.a.h2.H2ConsoleAutoConfiguration    : H2 console available at '/h2-console'. Database available at 'jdbc:h2:file:~/sampleDB'
```

h2 console startup message

As the log says, the database is available at the user's directory. You can open it and find the sampleDB.mv file there. It also says that now you can get to the h2 console at the `localhost:8080/h2-console` URL with your browser.

You can also change the `/h2-console` path in the **application.properties** file:

```
spring.h2.console.path=/h2
```



localhost:8080/h2-console/login.jsp

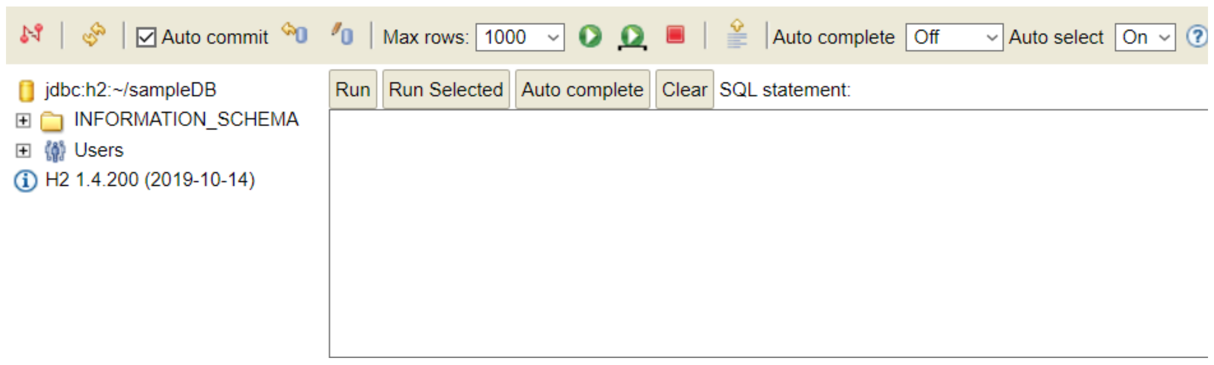
English Preferences Tools Help

Login

Saved Settings:	Generic H2 (Embedded)
Setting Name:	Generic H2 (Embedded)
Driver Class:	org.h2.Driver
JDBC URL:	jdbc:h2:~/test
User Name:	sa
Password:	••••••

Replace the JDBC URL value with the one you set at `spring.datasource.url` parameter and connect with the username and password you set at `spring.datasource.username` and `spring.datasource.password` parameters.

The main console page should look like the one in the following picture.



You can see the window for your SQL queries and the sampleDB schema. Now we can create the table to this schema.

§5. Data parameters

A simple example of an entity model class is an object with specific annotations on it and its fields. For example, our table will be called "User" and will contain fields:

- id
- username
- password
- enabled

Explore the model class for this table:

```
@Entity(name = "user")
public class User {

    @Id
    @Column
    private long userId;

    @Column
    private String username;

    @Column
    private String password;

    @Column
    private boolean enabled;

    //getters and setters

    //equals() and hashCode() methods
}
```

Hibernate can create the table according to this model. To enable updating schema by models, we set some properties at the **application.properties** file again:

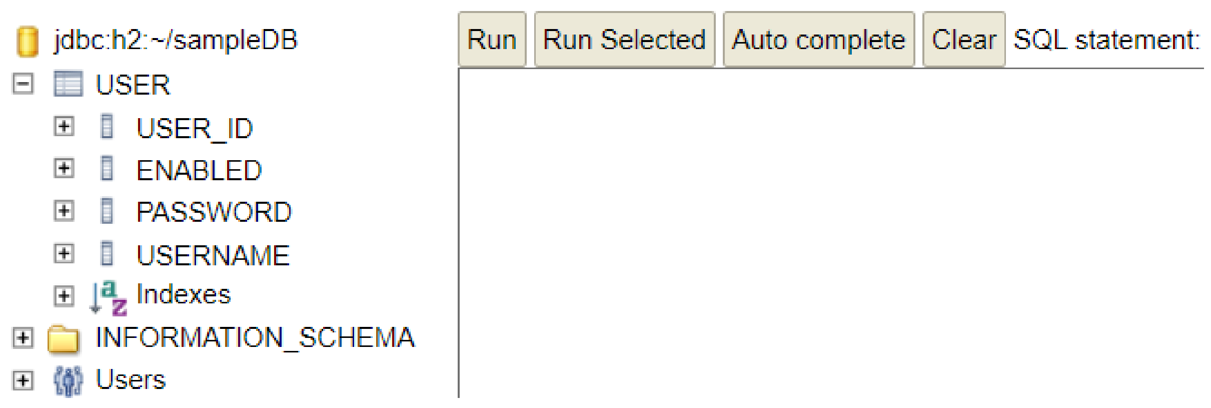
```
spring.jpa.hibernate.ddl-auto=update
```

This parameter can get different values to change the schema creation behavior on the application startup:

- update: updating schema if some changes were made to entity classes;
- create: always drops the previous schema and generates a new one;

- create-drop: creates the new schema and drops the previous when an application is stopped;
- validate: just validating the schema and making no changes to the existing one.

Now let's restart the application and check the changes at the h2 console.



You can see that the USER table was created with all of the fields from the `User` class.

§6. Logging SQL queries

We did not insert any information to the table here, but usually, the application runs different SQL queries at runtime. To see these queries in the application log you can enable show-sql in the application-properties file :

```
spring.jpa.show-sql=true
```

It will not show the queries made in the H2 console, only the queries initiated in the application.

§7. Conclusion

In this topic, you've created the Spring Boot application with H2 in-memory integration and set up the H2 console by writing the **application.properties** file.

At the end of the application setting, this file can look like this:

```
#datasource settings
spring.datasource.url=jdbc:h2:file:~/sampleDB
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=abc123

#data settings
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update

#console settings
spring.h2.console.enabled=true
spring.h2.console.path=/h2
spring.h2.console.settings.trace=false
spring.jpa.show-sql=true
```

Report a typo

71 users liked this theory. 5 didn't like it. **What about you?**

Start practicing

Comments (30) Hints (2) Useful links (0) Show discussion