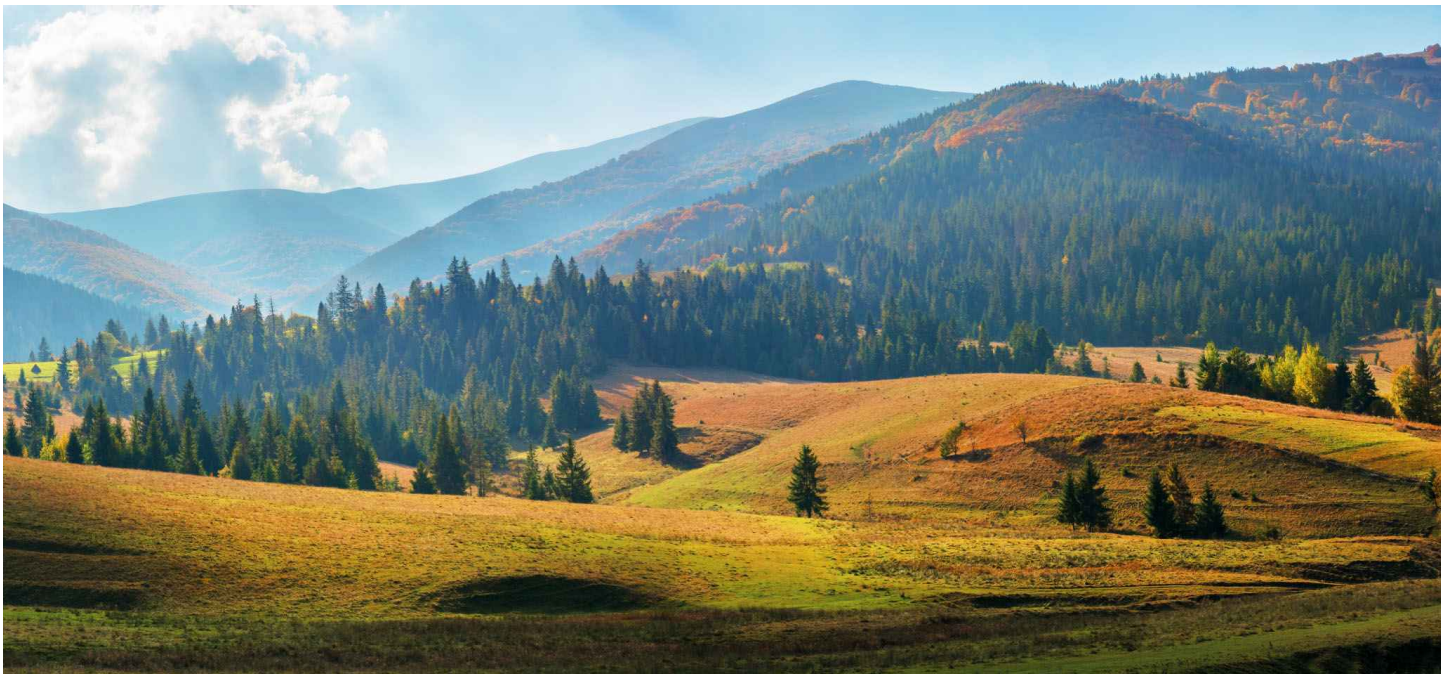


14-Database, Schema & Data Configuration

Author: Vincent Lau

Learning Objectives

> Include the correct dependencies to set up MySQL/ PostgreSQL database connection.
>
> Include the correct application yml configuration to setup MySQL/ PostgreSQL.
>
> Ways to initialize the database, schema & data.

Configuring MySQL Database

Maven Dependencies

- When initializing the project with Spring Initializr, select **Spring Web**, **Spring Data JPA**, and **MySQL Driver**.
- The resulting *pom.xml* should include these dependenies:

```xml
<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>
```

MySQL

```xml
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <!-- use the latest -->
    <version>8.0.28</version>
</dependency>
```

PostgreSQL

```xml
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <!-- use the latest -->
    <version>42.2.23</version>
</dependency>
```

application.yml

- Spring Boot has *H2* as the default database. Consequently, when you want to use any other database, you must define the connection attributes in the `application.yml` file.

MySQL

```yaml
spring.datasource:
  url: jdbc:mysql://localhost:3306/database_name
  username: ${DB_USER}
  password: ${DB_PASSWORD}
  driver-class-name: com.mysql.cj.jdbc.Driver
```

```
spring.jpa:
  properties.hibernate.dialect: org.hibernate.dialect.MySQL8Dialect
  hibernate.ddl-auto: none
  show-sql: true
```

PostgreSQL

```
spring.datasource:
    url: jdbc:postgresql://localhost:5432/database_name
    username: your_username
    password: your_password
    driver-class-name: org.postgresql.Driver
spring.jpa:
  properties.hibernate.dialect: org.hibernate.dialect.PostgreSQLDialect
  hibernate.ddl-auto: none
  show-sql: true
```

Schema Config Using Hibernate

Spring provides a JPA-specific **property that Hibernate uses for DDL generation: *spring.jpa.hibernate.ddl-auto*.**

- `spring.jpa.hibernate.ddl-auto` can be ***none, update, create, or create-drop***
  - `none` : The default for MySQL. No change is made to the database structure.
  - `update` : Hibernate changes the database according to the given entity structures.
  - `create` : Creates the database every time but does not drop it on close.
  - `create-drop` : Creates the database and drops it when SessionFactory closes.
  - `validate` : Performs a validation check to ensure that your database schema is in sync with your entity mappings, but it does not make any modifications to the database itself. It is a **read-only operation for schema validation** and is useful for catching schema issues during development and testing without altering the database.
- For the first time you interact with the database, it must be with either `create` or `update` , because you do not yet have the database structure. After the first run, you can switch it to `validate` , `update` or `none` , according to program requirements. Use `update` when you want to make some changes to the database structure.
- The default for *H2* and other embedded databases is `create-drop` . For *MySQL*, the default is `none` .

> It is a good security practice to, after your database is in a production state, set this to `none`, revoke all privileges from the MySQL user connected to the Spring application, and give the MySQL user only `SELECT`, `UPDATE`, `INSERT`, and `DELETE`. You can read more about this at the end of this guide.

Customizing Database Creation

- By default, Spring Boot automatically creates the schema of an embedded *DataSource*.
- If we need to control or customize this behavior, we can use the property *spring.sql.init.mode*.
- The `spring.sql.init.mode` property takes one of three values:
  - ***always*** - always initialize the database
  - ***embedded*** - always initialize if an embedded database (e.g. H2 Database) is in use. This is the default if the property value is not specified.
  - ***never*** - never initialize the database
- If we are using a non-embedded database, let's say MySQL or PostGreSQL, and want to initialize its schema, we'll have to set this property to ***always*** **(for the first run at least)**.

Ways to define & initialize Schema & Data

Creating the @Entity Model and Database Table

- By creating the entity model annotated with `@Entity` in Java and setting `spring.jpa.hibernate.ddl-auto` to `create` or `create-drop`, Spring will search for entities in our packages and create the corresponding tables automatically.

```
1  @Entity
2  public class Book {
3      @Id
4      @GeneratedValue(strategy = GenerationType.IDENTITY)
5      private Long id;
6
7      @Column(nullable = false)
8      private String title;
9
10     @Column(nullable = false)
11     private String author;
12
13     @Column(nullable = false)
```

```
14      private Integer page;
15
16      // constructors, getters and setters
17  }
```

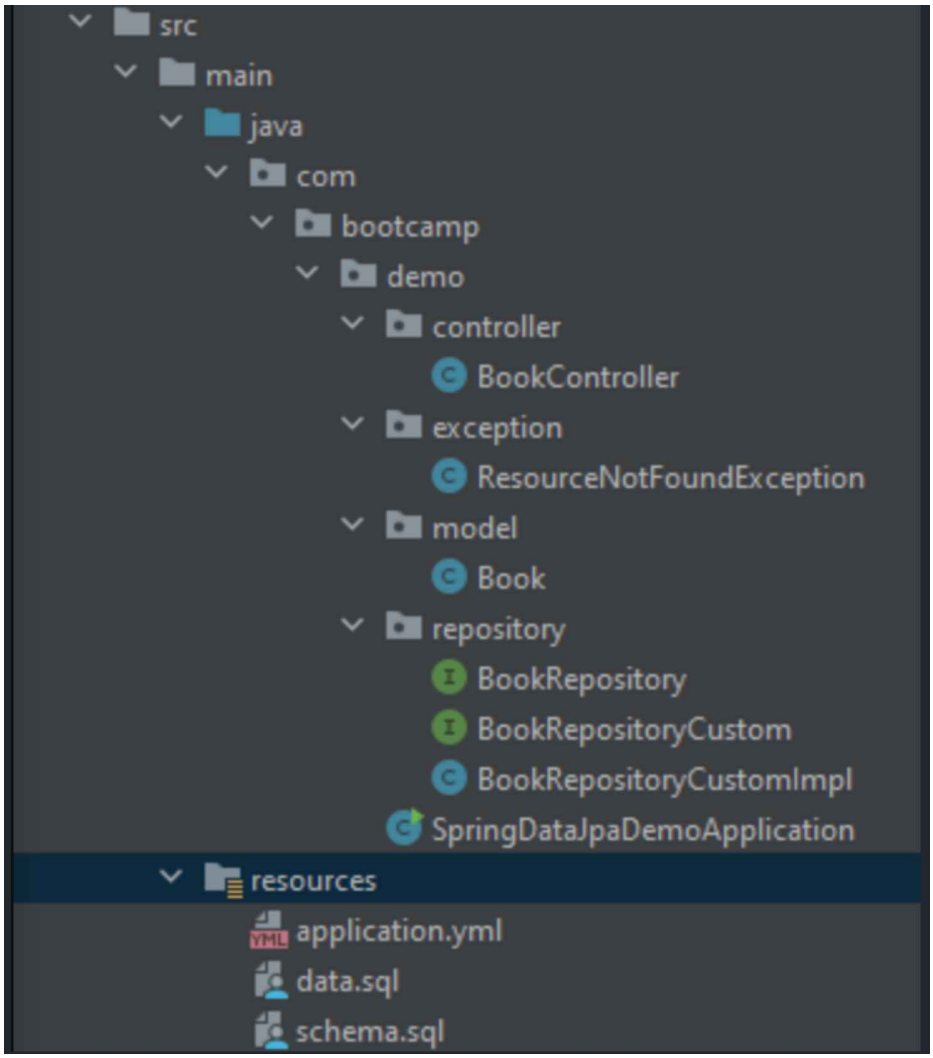
- If we run our application, **Spring Boot will create an empty table for us but won't populate it with anything.**
- Sometimes, we need more fine-grained control over the database alterations. And that's when we can use the *data.sql* and *schema.sql* files in Spring.

The data.sql File

- An easy way to populate an empty table is to create a file named *data.sql*:

```
1 INSERT INTO book (id, title, author, page) VALUES (1, 'Spring in Action',
  'Craig Walls', 521);
2 INSERT INTO book (id, title, author, page) VALUES (2, 'Spring Boot in Action',
  'Craig Walls', 266);
3 INSERT INTO book (id, title, author, page) VALUES (3, 'Thinking in Java',
  'Bruce Eckel', 1079);
4 INSERT INTO book (id, title, author, page) VALUES (4, 'On Java 8', 'Bruce
  Eckel', 1778);
5 INSERT INTO book (id, title, author, page) VALUES (5, 'Effective Java',
  'Joshua Bloch', 413);
```

- When we run the project with this file on the classpath or in the *resources* folder, Spring will pick it up and use it for populating the database.



The schema.sql File

- Sometimes, we don't want to rely on the default schema creation mechanism.
- In such cases, we can create a custom *schema.sql* file:

```
CREATE TABLE book (
        id BIGINT NOT NULL AUTO_INCREMENT, -- database can handle the increment by "thread safe" approach
        author VARCHAR(255) NOT NULL,
        page INTEGER NOT NULL,
        title VARCHAR(255) NOT NULL,
        PRIMARY KEY (id)
);
```

Properties for Script-Based Initialization

- In the `application.yml` file, we should also set the `spring.sql.init.mode` and `spring.jpa.hibernate.ddl-auto` properties properly.

```
# properties
spring.sql.init.mode: always
spring.jpa.hibernate.ddl-auto: none
```

- This will turn off automatic DDL generation by Hibernate, and allow non-embedded databases such as MySQL to initialize its schema.
- After the first run where the schema has been created, we should set `spring.sql.init.mode` back to ***never***, so that the application will not attempt to create the schema twice (and throw exceptions) in the MySQL database.