Spring Data JPA - Find Records From MySQL

Last Updated : 23 Jul, 2025

Spring Data JPA simplifies database interactions in Spring Boot applications by providing a seamless way to work with relational databases like **MySQL**. It eliminates boilerplate code and allows developers to perform CRUD operations efficiently using **JPA repositories**. With Spring Boot and Spring Data JPA, fetching records from a MySQL database becomes straightforward, making it an essential approach for building scalable applications. In this article, we will explore how to **retrieve a record from a MySQL table using Spring Data JPA's findById() method.**

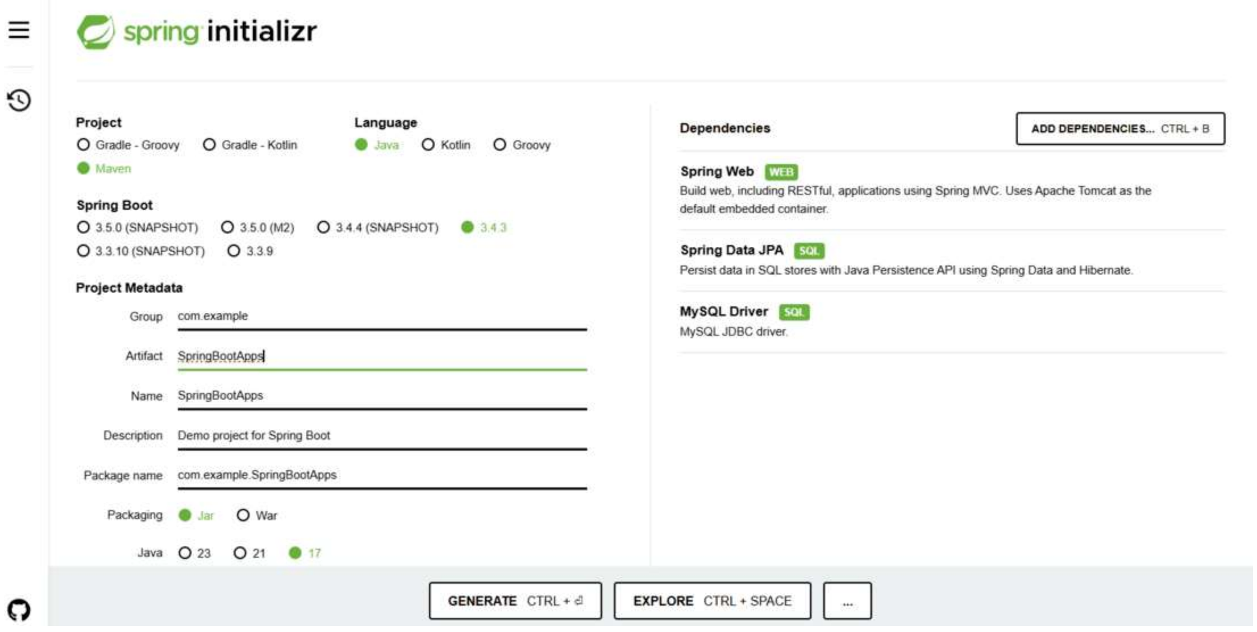
> ***Note***: *findById() method of JpaRepository<> is used to retrieve a record by its primary key from the MySQL database.*

Step-by-Step Implementation

Step 1: Create a Spring Boot Project

Go to Spring Initializr and generate a Spring Boot project with the following dependencies:

- Spring Web
- Spring Data JPA
- MySQL Driver



Project: Gradle - Groovy, Gradle - Kotlin, [✓ Maven]

Language: [✓ Java], Kotlin, Groovy

Spring Boot: 3.5.0 (SNAPSHOT), 3.5.0 (M2), 3.4.4 (SNAPSHOT), [✓ 3.4.3], 3.3.10 (SNAPSHOT), 3.3.9

Project Metadata

Group	com.example
Artifact	SpringBootApps
Name	SpringBootApps
Description	Demo project for Spring Boot
Package name	com.example.SpringBootApps
Packaging	[✓ Jar] War
Java	23 21 [✓ 17]

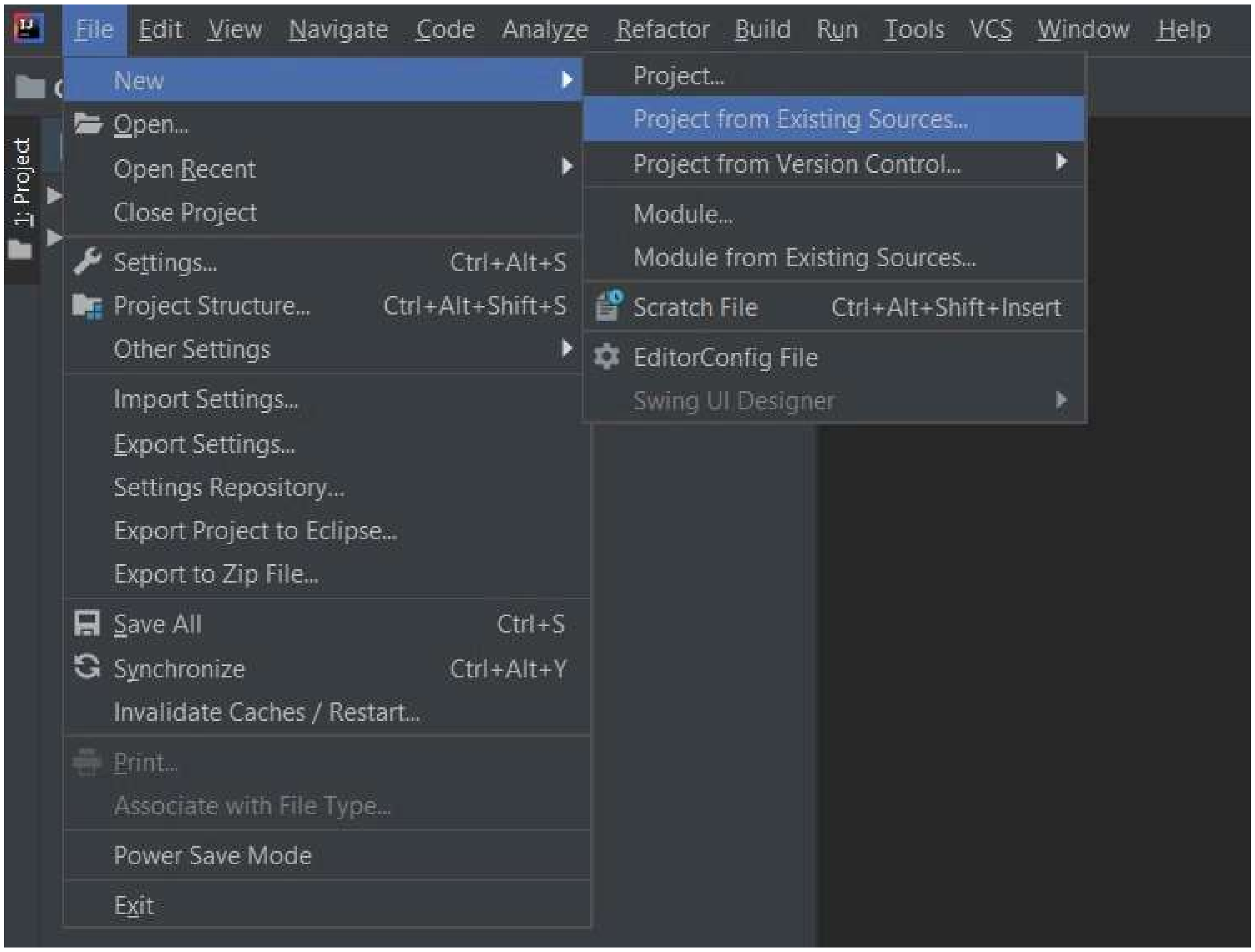
Dependencies

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

MySQL Driver SQL
MySQL JDBC driver.

Extract the zip file. Now open a suitable IDE and then go to **File -> New -> Project from Existing Sources** and select pom.xml. Click on import changes on prompt and wait for the project to sync.


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
New
Open...
Open Recent
Close Project
Settings... Ctrl+Alt+S
Project Structure... Ctrl+Alt+Shift+S
Other Settings
Import Settings...
Export Settings...
Settings Repository...
Export Project to Eclipse...
Export to Zip File...
Save All Ctrl+S
Synchronize Ctrl+Alt+Y
Invalidate Caches / Restart...
Print...
Associate with File Type...
Power Save Mode
Exit
Project...
Project from Existing Sources...
Project from Version Control...
Module...
Module from Existing Sources...
Scratch File Ctrl+Alt+Shift+Insert
EditorConfig File
Swing UI Designer
1: Project

Note: In the Import Project for Maven window, make sure you choose the same version of JDK which you selected while creating the project. For database operation we have to configure the Spring application with the database also it is required to add configuration of the database before executing the Spring project. All the configuration is added in the file application.properties of the Spring project.

Step 2: Add Required Dependencies

Modify the pom.xml file to include the necessary dependencies:

```xml
<dependencies>
    <!-- Spring Boot Data JPA -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <!-- Spring Boot Web Starter -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- MySQL Connector -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <scope>runtime</scope>
    </dependency>
</dependencies>
```

Step 3: Configure Database Connection

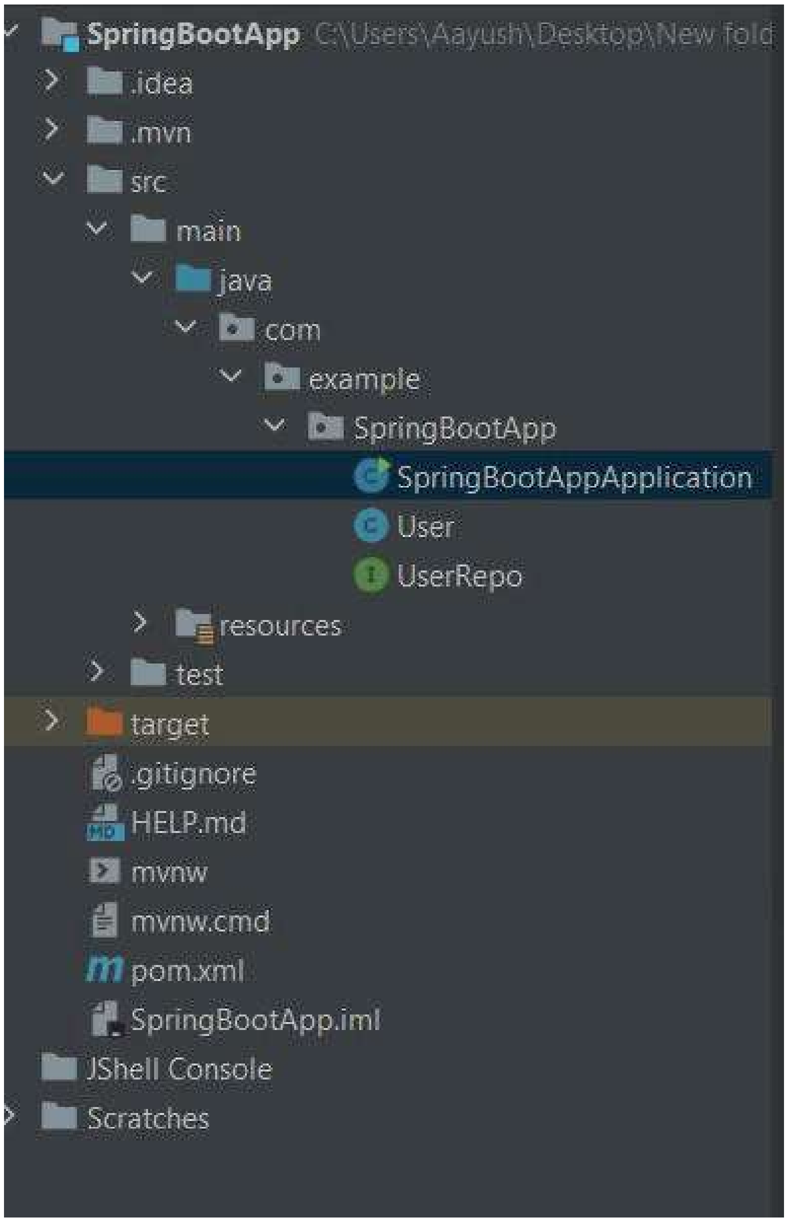
For database operations, the Spring application must be properly configured with the database. This includes adding the necessary database configurations before executing the Spring project. All required configurations should be specified in the application.properties file of the Spring project.

Add the following properties in **src/main/resources/application.properties:**

```
spring.datasource.url=jdbc:mysql://localhost:3306/userdb
spring.datasource.username=root
spring.datasource.password=yourpassword
spring.jpa.hibernate.ddl-auto=update
```

Replace yourpassword with the actual MySQL password.

Project Structure:



```
SpringBootApp C:\Users\Aayush\Desktop\New fold
  .idea
  .mvn
  src
    main
      java
        com
          example
            SpringBootApp
              SpringBootAppApplication
              User
              UserRepo
      resources
    test
  target
  .gitignore
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
  SpringBootApp.iml
JShell Console
Scratches
```

Step 4: Create the Entity Class

Create a User entity in **src/main/java/com/example/model/User.java:**

```
import jakarta.persistence.*;
```

```
@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String name;

    // Constructors
    public User() {}

    public User(String name) {
        this.name = name;
    }

    // Getters and Setters
    public int getId() {
        return id;

    }
    public String getName() {
        return name;

    }
    public void setName(String name) {
        this.name = name;

    }
}
```

Step 5: Create Repository Interface

Define a repository interface in

src/main/java/com/example/repository/UserRepository.java:

```
import org.springframework.data.jpa.repository.JpaRepository;
import com.example.model.User;

public interface UserRepository extends JpaRepository<User, Integer> {}
```

Step 6: Implement Service Class

Create a service to interact with the database in

src/main/java/com/example/service/UserService.java:

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.example.model.User;
import com.example.repository.UserRepository;
import java.util.Optional;

@Service
public class UserService {
    @Autowired
    private UserRepository userRepository;

    public Optional<User> findUserById(int id) {
        return userRepository.findById(id);
    }
}
```

Step 7: Create Controller for API

Implement a controller to handle HTTP requests in **src/main/java/com/example/controller/UserController.java**:

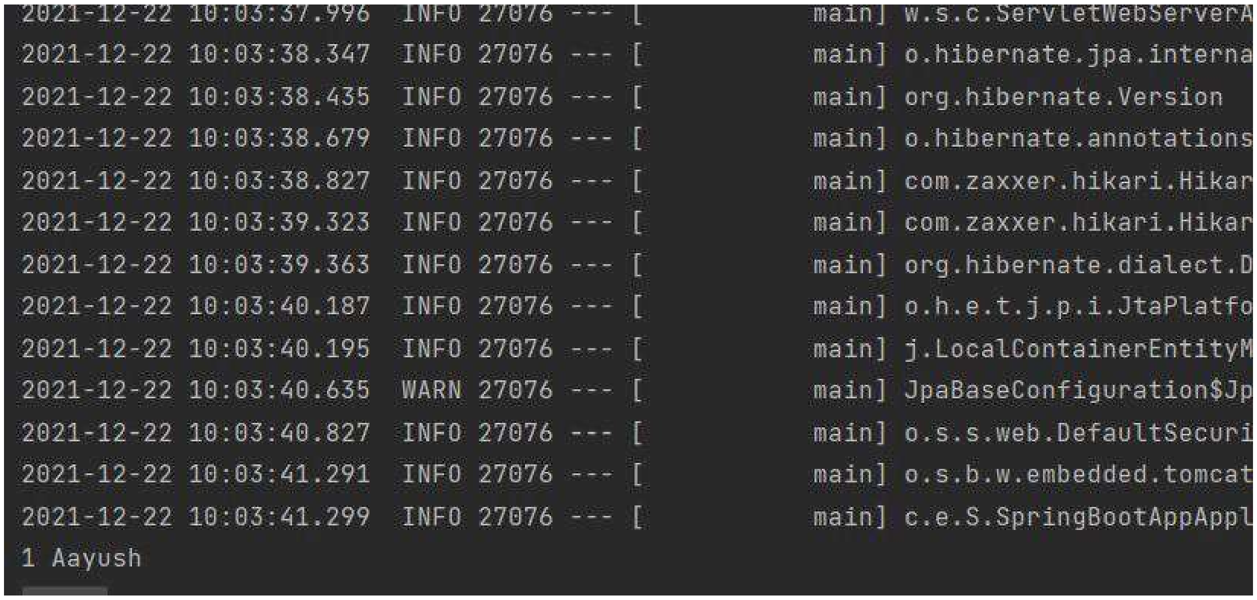
```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import com.example.model.User;
import com.example.service.UserService;
import java.util.Optional;

@RestController
@RequestMapping("/users")
public class UserController {
    @Autowired
    private UserService userService;

    @GetMapping("/{id}")
    public Optional<User> getUserById(@PathVariable int id) {
        return userService.findUserById(id);
    }
}
```

Step 8: Run the Application

Start the Spring Boot application using:

> *mvn spring-boot:run*

Output:



```
2021-12-22 10:03:37.996  INFO 27076 --- [           main] w.s.c.ServletWebServerA
2021-12-22 10:03:38.347  INFO 27076 --- [           main] o.hibernate.jpa.interna
2021-12-22 10:03:38.435  INFO 27076 --- [           main] org.hibernate.Version
2021-12-22 10:03:38.679  INFO 27076 --- [           main] o.hibernate.annotations
2021-12-22 10:03:38.827  INFO 27076 --- [           main] com.zaxxer.hikari.Hikar
2021-12-22 10:03:39.323  INFO 27076 --- [           main] com.zaxxer.hikari.Hikar
2021-12-22 10:03:39.363  INFO 27076 --- [           main] org.hibernate.dialect.D
2021-12-22 10:03:40.187  INFO 27076 --- [           main] o.h.e.t.j.p.i.JtaPlatfo
2021-12-22 10:03:40.195  INFO 27076 --- [           main] j.LocalContainerEntityM
2021-12-22 10:03:40.635  WARN 27076 --- [           main] JpaBaseConfiguration$Jp
2021-12-22 10:03:40.827  INFO 27076 --- [           main] o.s.s.web.DefaultSecuri
2021-12-22 10:03:41.291  INFO 27076 --- [           main] o.s.b.w.embedded.tomcat
2021-12-22 10:03:41.299  INFO 27076 --- [           main] c.e.S.SpringBootAppAppl
1 Aayush
```

Once the application is running, **test the API using Postman or a browser:**

GET http://localhost:8080/users/1

Expected Output (JSON Response):

```
{
  "id": 1,
  "name": "Aayush"
}
```