**Giới thiệu**

Để đi tiếp trong series Spring Boot này, tôi không thể bỏ qua một phần quan trọng đó là giao tiếp với Database.

Nếu bạn cần xem lại các bài trước, thì nó ở đây:

[📞](https://loda.me/articles/sb8-tao-web-helloworld-voi-controller-thymeleaf)

[[SB8] Tạo Web Helloworld với @Controller + Thymeleaf](https://loda.me/articles/sb8-tao-web-helloworld-voi-controller-thymeleaf)

[🧵](https://loda.me/articles/sb9-gii-thch-cch-thymeleaf-vn-hnh-expression-demo-full)

[[SB9] Giải thích cách Thymeleaf vận hành + Expression + Demo Full](https://loda.me/articles/sb9-gii-thch-cch-thymeleaf-vn-hnh-expression-demo-full)

[🎲](https://loda.me/articles/sb10-requestmapping-postmapping-modelattribute-requestparam-web-to-do-voi-thymeleaf)

[[SB10] @RequestMapping + @PostMapping + @ModelAttribute + @RequestParam + Web To-Do với Thymeleaf](https://loda.me/articles/sb10-requestmapping-postmapping-modelattribute-requestparam-web-to-do-voi-thymeleaf)

Vì thiếu phần kết nối với Database nên chúng ta chưa thể hoàn thiện được trang Web của mình, trong bài này chúng ta sẽ tìm hiểu **Spring Boot JPA**.

Trong bài có đề cập kiến thức:

1. [Hibernate](https://loda.me/hibernate-la-gi-loda1554623701594)
2. [Lombok](https://loda.me/general-huong-dan-su-dung-lombok-giup-code-java-nhanh-hon-69-loda1552789752787)

**Spring Boot JPA**

**Spring Boot JPA** là một phần trong hệ sinh thái Spring Data, nó tạo ra một layer ở giữa tầng service và database, giúp chúng ta thao tác với database một cách dễ dàng hơn, tự động config và giảm thiểu code thừa thãi.

**Spring Boot JPA** đã wrapper Hibernate và tạo ra một interface mạnh mẽ. Nếu như bạn gặp khó khăn khi làm việc với Hibernate thì đừng lo, bạn hãy để **Spring JPA** làm hộ.

**Cài đặt**

Để thêm Spring JPA vào project, bạn cần thêm dependency spring-boot-starter-data-jpa.

Ngoài ra, để connect tới MySql, chúng ta cần driver tương ứng, vì vậy phải bổ sung thêm cả dependency mysql-connector-java vào *pom.xml*.

*pom.xml*

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<packaging>pom</packaging>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.0.5.RELEASE</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<groupId>me.loda.spring</groupId>

<artifactId>spring-boot-learning</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>spring-boot-learning</name>

<description>Everything about Spring Boot</description>

<properties>

<java.version>1.8</java.version>

</properties>

<dependencies>

<!--spring mvc, rest-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!--spring jpa-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<!-- mysql connector -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

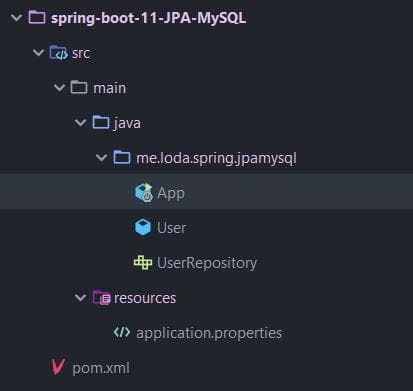
</plugin>

</plugins>

</build>

</project>

Cấu trúc thư mục:



**Tạo Table và dữ liệu**

Trước khi bắt đầu, chúng ta cần tạo ra dữ liệu trong Database. Ở đây tôi chọn MySQL.

Dưới đây là SQL Script để tạo DATABASE micro\_db. Chứa một TABLE duy nhất là User.

Khi chạy script này, nó sẽ tự động insert vào db 100 User.

CREATE DATABASE micro\_db;

use micro\_db;

CREATE TABLE `user`

(

`id` bigint(20) NOT NULL AUTO\_INCREMENT,

`hp` int NULL DEFAULT NULL,

`stamina` int DEFAULT NULL,

`atk` int DEFAULT NULL,

`def` int DEFAULT NULL,

`agi` int DEFAULT NULL,

PRIMARY KEY (`id`)

);

DELIMITER $$

CREATE PROCEDURE generate\_data()

BEGIN

DECLARE i INT DEFAULT 0;

WHILE i < 100 DO

INSERT INTO `user` (`hp`,`stamina`,`atk`,`def`,`agi`) VALUES (i,i,i,i,i);

SET i = i + 1;

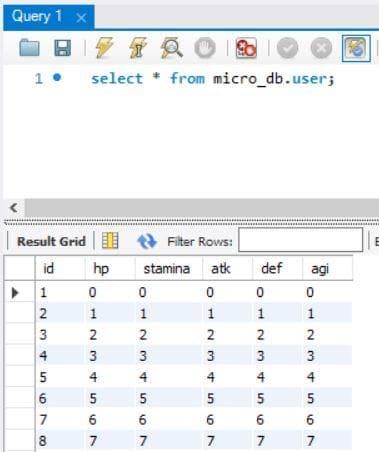
END WHILE;

END$$

DELIMITER ;

CALL generate\_data();

Sau khi chạy xong script trên, chúng ta kiểm tra database đã có dữ liệu chưa.



**Tạo Model User**

Khi đã có dữ liệu trong Database. Chúng ta sẽ tạo một Class trong Java để mapping thông tin.

Phần này chúng ta cần có một chút kiến thức về Hibernate. Nếu bạn chưa biết những Annotation ở dưới đây để làm gì thì hãy tạm dừng và [tìm hiểu Hibernate tại đây](https://loda.me/hibernate-la-gi-loda1554623701594).

*User.java*

@Entity

@Table(name = "user")

@Data

public class User implements Serializable {

private static final long serialVersionUID = -297553281792804396L;

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

// Mapping thông tin biến với tên cột trong Database

@Column(name = "hp")

private int hp;

@Column(name = "stamina")

private int stamina;

// Nếu không đánh dấu @Column thì sẽ mapping tự động theo tên biến

private int atk;

private int def;

private int agi;

}

Tới đây là chúng ta làm được nửa đường rồi.

**Vấn đề của Hibernate truyền thống**

Thông thường, khi bạn đã định nghĩa Entity tương ứng với Table trong DB thông qua Hibernate. Thì nhiệm vụ tiếp theo sẽ là tạo ra các class thao tác với DB.

Ví dụ muốn query lấy tất cả User bằng Hibernate truyền thống sẽ như sau:

// Giả sử đã có đối tượng session rồi

Session session = getSession();

try {

// Tất cả các lệnh hành động với DB thông qua Hibernate

// đều phải nằm trong 1 giao dịch (Transaction)

// Bắt đầu giao dịch

session.getTransaction().begin();

// Tạo một query

String sql = "Select u from " + User.class.getName() + " u ";

// Tạo đối tượng Query.

Query<User> query = session.createQuery(sql);

// Thực hiện truy vấn và lấy ra dữ liệu.

List<User> users = query.getResultList();

// In ra màn hình

for (User user : users) {

System.out.println(user);

}

// Commit dữ liệu và kết thúc session.

session.getTransaction().commit();

} catch (Exception e) {

e.printStackTrace();

// Rollback trong trường hợp có lỗi xẩy ra.

session.getTransaction().rollback();

}

Mặc dù Hibernate đã làm rất tốt và giảm thiểu code cho việc thao tác với Database xuống rồi, những nó vẫn chưa hẳn là dễ dàng :(

Mục đích ban đầu của Hibernate là giúp người lập trình dễ sử dụng, tuy nhiên, trên thực tế, nhiều người gặp khó khăn trong việc sử dụng với Hibernate hơn cả jdbc.

Nắm được vấn đề này, **Spring Data** đã wrapper lên Hibernate một lớp nữa gọi là **Spring JPA**, giúp cho mọi thao tác với DB của chúng ta rút ngắn xuống còn 1 dòng và tất nhiên là làm mờ Hibernate xuống đáng kể để tránh rắc rối cho người lập trình.

**JpaRepository**

Để sử dụng **Spring JPA**, bạn cần sử dụng interface JpaRepository.

Yêu cầu của interface này đó là bạn phải cung cấp 2 thông tin:

1. Entity (Đối tượng tương ứng với Table trong DB)
2. Kiểu dữ liệu của khóa chính (primary key)

Ví dụ: Tôi muốn lấy thông tin của bảng User thì làm như sau:

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface UserRepository extends JpaRepository<User, Long> {

}

Vậy thôi, @Repository đánh dấu UserRepository là một Bean và chịu trách nhiệm giao tiếp với DB.

**Spring Boot** sẽ tự tìm thấy và khởi tạo ra đối tượng UserRepository trong Context. Việc tạo ra UserRepository hoàn toàn tự động và tự config, vì chúng ta đã kế thừa JpaRepository.

Bây giờ, việc lấy ra toàn bộ User sẽ như sau:

@Autowired

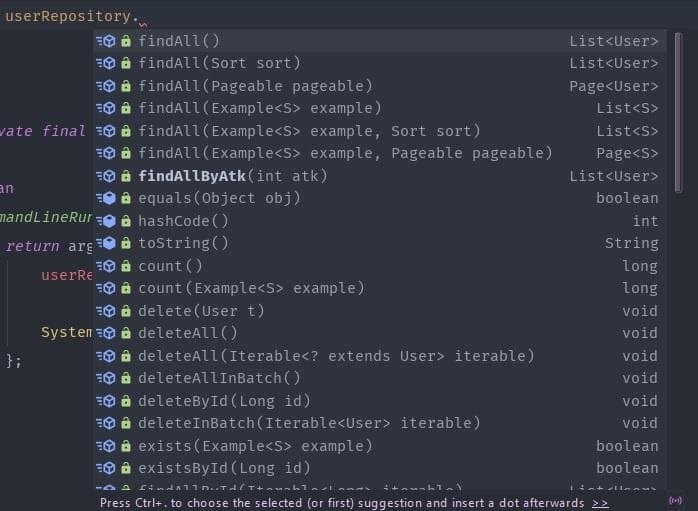
UserRepository userRepository;

userRepository.findAll()

.forEach(System.out::println);

Đơn giản và ngắn gọn hơn rất nhiều.

Nếu bạn tìm kiếm thì sẽ thấy UserRepository có hàng chục method mà chúng ta không cần viết lại nữa. Vì nó kế thừa JpaRepository rồi.



**Demo**

Bây giờ chúng ta sẽ làm ứng dụng Demo các tính năng cơ bản với JpaRepository

Bước đầu tiên là config thông tin về MySQL trong application.properties

*application.properties*

spring.datasource.url=jdbc:mysql://localhost:3306/micro\_db?useSSL=false

spring.datasource.username=root

spring.datasource.password=root

## Hibernate Properties

# The SQL dialect makes Hibernate generate better SQL for the chosen database

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect

# Hibernate ddl auto (create, create-drop, validate, update)

spring.jpa.hibernate.ddl-auto = update

logging.level.org.hibernate = ERROR

**Spring JPA** sẽ tự kết nối cho chúng ta, mà không cần thêm một đoạn code nào cả.

*User.java*

@Entity

@Table(name = "user")

@Data

public class User implements Serializable {

private static final long serialVersionUID = -297553281792804396L;

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

// Mapping thông tin biến với tên cột trong Database

@Column(name = "hp")

private int hp;

@Column(name = "stamina")

private int stamina;

// Nếu không đánh dấu @Column thì sẽ mapping tự động theo tên biến

private int atk;

private int def;

private int agi;

}

*App.java*

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import lombok.RequiredArgsConstructor;

@SpringBootApplication

@RequiredArgsConstructor

public class App {

public static void main(String[] args) {

ApplicationContext context = SpringApplication.run(App.class, args);

UserRepository userRepository = context.getBean(UserRepository.class);

// Lấy ra toàn bộ user trong db

userRepository.findAll()

.forEach(System.out::println);

// Lưu user xuống database

User user = userRepository.save(new User());

// Khi lưu xong, nó trả về User đã lưu kèm theo Id.

System.out.println("User vừa lưu có ID: " + user.getId());

Long userId = user.getId();

// Cập nhật user.

user.setAgi(100);

// Update user

// Lưu ý, lúc này đối tượng user đã có Id.

// Nên nó sẽ update vào đối tượng có Id này

// chứ không insert một bản ghi mới

userRepository.save(user);

// Query lấy ra user vừa xong để kiểm tra xem.

User user2 = userRepository.findById(userId).get();

System.out.println("User: " + user);

System.out.println("User2: " + user2);

// Xóa User khỏi DB

userRepository.delete(user);

// In ra kiểm tra xem userId còn tồn tại trong DB không

User user3 = userRepository.findById(userId).orElse(null);

System.out.println("User3: " + user2);

}

}

OUTPUT chương trình:

User vừa lưu có ID: 104

// sau khi update, cả 2 đối tượng user đều có giá trị agi mới

User: User(id=104, hp=0, stamina=0, atk=0, def=0, agi=100)

User2: User(id=104, hp=0, stamina=0, atk=0, def=0, agi=100)

// Sau khi xóa, user không còn tồn tại

User3: null