**Giới thiệu**

[Trong bài trước](https://loda.me/spring-boot-11-huong-dan-spring-boot-jpa-my-sql-loda1558687596060), mình đã giới thiệu với các bạn Spring JPA, với cách cài đặt và sử dụng hết sức dễ dàng.

[🎎](https://loda.me/articles/sb11-huong-dan-spring-boot-jpa-mysql)

[[SB11] Hướng dẫn Spring Boot JPA + MySQL](https://loda.me/articles/sb11-huong-dan-spring-boot-jpa-mysql)

Nhưng trong thực tế, sẽ có một số yêu cầu nghiệp vụ nằm ngoài các method là JPA hỗ trợ sẵn, lúc này bạn phải tự tạo ra câu query của riêng mình.

Trong phần này, chúng ta sẽ tìm hiểu cách để tự tạo ra các câu truy vấn.

**Query Creation**

Trong **Spring JPA**, có một cơ chế giúp chúng ta tạo ra các câu Query mà không cần viết thêm code.

Cơ chế này xây dựng Query từ tên của method.

Ví dụ:

Chúng ta có đối tượng User.

*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;

}

Khi chúng ta đặt tên method là: findByAtk(int atk)

Thì **Spring JPA** sẽ tự định nghĩa câu Query cho method này, bằng cách xử lý tên method. Vậy là chúng ta đã có thể truy vấn dữ liệu mà chỉ mất thêm 1 dòng code.

Cơ chế xây dựng Query từ tên method này giúp chúng ta tiết kiệm thời gian với những query có logic đơn giản, và cũng đặc biệt hữu ích là nó giống ngôn ngữ con người thường nói hơn là SQL. (human-readable)

**Quy tắc đặt tên method trong Spring JPA**

Trong **Spring JPA**, cơ chế xây dựng truy vấn thông qua tên của method được quy định bởi các tiền tố sau:

find…By, read…By, query…By, count…By, và get…By.

phần còn lại sẽ được parse theo tên của thuộc tính (viết hoa chữ cái đầu). Nếu chúng ta có nhiều điều kiện, thì các thuộc tính có thể kết hợp với nhau bằng chữ And hoặc Or.

Ví dụ:

interface PersonRepository extends JpaRepository<User, Long> {

// Dễ

// version rút gọn

Person findByLastname(String lastname);

// verson đầy đủ

Person findPersonByLastname(String lastname);

Person findAllByLastname(String lastname);

// Trung bình

List<Person> findByEmailAddressAndLastname(EmailAddress emailAddress, String lastname);

// findDistinct là tìm kiếm và loại bỏ đi các đối tượng trùng nhau

List<Person> findDistinctPeopleByLastnameOrFirstname(String lastname, String firstname);

List<Person> findPeopleDistinctByLastnameOrFirstname(String lastname, String firstname);

// IgnoreCase là tìm kiếm không phân biệt hoa thường, ví dụ ở đây tìm kiếm lastname

// lastname sẽ không phân biệt hoa thường

List<Person> findByLastnameIgnoreCase(String lastname);

// AllIgnoreCase là không phân biệt hoa thường cho tất cả thuộc tính

List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstname);

// OrderBy là cách sắp xếp thứ tự trả về

// Sắp xếp theo Firstname ASC

List<Person> findByLastnameOrderByFirstnameAsc(String lastname);

// Sắp xếp theo Firstname Desc

List<Person> findByLastnameOrderByFirstnameDesc(String lastname);

}

Các thuộc tính trong tên method phải thuộc về Class đó, nếu không sẽ gây ra lỗi. Tuy nhiên, trong một số trường hợp bạn có thể query bằng thuộc tính con.

Ví dụ:

Đói tượng Person có thuộc tính là Address và trong Address lại có ZipCode

// person.address.zipCode

List<Person> findByAddressZipCode(ZipCode zipCode);

**@Query**

nếu bạn thực sự thấy khó với cách tiếp cận ở phía trên, thì **Spring JPA** còn hỗ trợ chúng ta một cách nguyên thủy khác.

Với cách sử dụng @Query, bạn sẽ có thể sử dụng câu truy vấn JPQL (Hibernate) hoặc raw SQL.

Nếu bạn chưa biết JPQL thì có thể [xem tại đây](https://loda.me/huong-dan-su-dung-criteria-api-trong-hibernate-loda1552815848300) Ví dụ:

public interface UserRepository extends JpaRepository<User, Long> {

// Khi được gắn @Query, thì tên của method không còn tác dụng nữa

// Đây là JPQL

@Query("select u from User u where u.emailAddress = ?1")

User myCustomQuery(String emailAddress);

// Đây là Native SQL

@Query(value = "select \* from User u where u.email\_address = ?1", nativeQuery = true)

User myCustomQuery2(String emailAddress);

}

Cách truyền tham số là gọi theo thứ tự các tham số của method bên dưới ?1, ?2

Nếu bạn không thích sử dụng ?{number} thì có thể đặt tên cho tham số.

public interface UserRepository extends JpaRepository<User, Long> {

// JPQL

@Query("SELECT u FROM User u WHERE u.status = :status and u.name = :name")

User findUserByNamedParams(@Param("status") Integer status, @Param("name") String name);

// Native SQL

@Query(value = "SELECT \* FROM Users u WHERE u.status = :status and u.name = :name", nativeQuery = true)

User findUserByNamedParamsNative(@Param("status") Integer status, @Param("name") String name);

}

**Demo**

Chúng ta sẽ thử hết các cách ở trên bằng demo dưới đây:

*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>

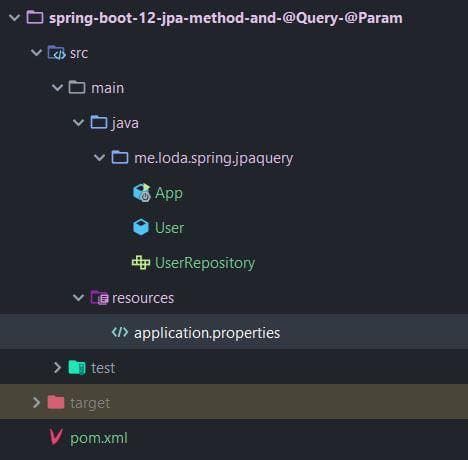
</plugin>

</plugins>

</build>

</project>

Cấu trúc thư mục:



**Tạo Database**

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();

**Cấu hình Spring**

*application.properties*

server.port=8081

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

**Tạo Model và Repository**

*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;

}

*UserRepository.java*

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

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

import org.springframework.data.repository.query.Param;

import org.springframework.stereotype.Repository;

@Repository

public interface UserRepository extends JpaRepository<User, Long> {

List<User> findAllByAtk(int atk);

List<User> findAllByAgiBetween(int start, int end);

@Query("SELECT u FROM User u WHERE u.def = :def")

User findUserByDefQuery(@Param("def") Integer def);

List<User> findAllByAgiGreaterThan(int agiThreshold);

}

**Chạy thử chương trình**

*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);

System.out.println("Tìm user với Agi trong khoảng (25 - 30)");

System.out.println("findAllByAgiBetween: ");

userRepository.findAllByAgiBetween(25, 30)

.forEach(System.out::println);

System.out.println("===========================================");

System.out.println("Tìm user với Agi trong lớn hơn 97");

System.out.println("findAllByAgiGreaterThan: ");

userRepository.findAllByAgiGreaterThan(97)

.forEach(System.out::println);

// Tìm kiếm tất cả theo Atk = 74

System.out.println("===========================================");

System.out.println("Tìm user với Atk = 74");

System.out.println("findAllByAtk: ");

userRepository.findAllByAtk(74)

.forEach(System.out::println);

System.out.println("===========================================");

System.out.println("Tìm User với def = 49");

System.out.println("SELECT u FROM User u WHERE u.def = :def");

User user = userRepository.findUserByDefQuery(49);

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

}

}

**OUTPUT chương trình:**

Tìm user với Agi trong khoảng (25 - 30)

findAllByAgiBetween:

User(id=26, hp=25, stamina=25, atk=25, def=25, agi=25)

User(id=27, hp=26, stamina=26, atk=26, def=26, agi=26)

User(id=28, hp=27, stamina=27, atk=27, def=27, agi=27)

User(id=29, hp=28, stamina=28, atk=28, def=28, agi=28)

User(id=30, hp=29, stamina=29, atk=29, def=29, agi=29)

User(id=31, hp=30, stamina=30, atk=30, def=30, agi=30)

===========================================

Tìm user với Agi trong lớn hơn 97

findAllByAgiGreaterThan:

User(id=99, hp=98, stamina=98, atk=98, def=98, agi=98)

User(id=100, hp=99, stamina=99, atk=99, def=99, agi=99)

===========================================

Tìm user với Atk = 74

findAllByAtk:

User(id=75, hp=74, stamina=74, atk=74, def=74, agi=74)

===========================================

Tìm User với def = 49

SELECT u FROM User u WHERE u.def = :def

User: User(id=50, hp=49, stamina=49, atk=49, def=49, agi=49)

Kết quả hoàn hảo tới mức Perfect :) Chúc bạn thành công