**Giới thiệu**

Cách biểu thị quan hệ n-n trong cơ sở dữ liệu là rất phổ biến, ví dụ một địa chỉ có thể có nhiều người ở (gia đình). và một người có thể có nhiều hơn một địa chỉ.

Bình thường, khi các bạn tạo table trong csdl để biểu thị mối quan hệ này, chúng ta sẽ tạo ra một bảng mới, tham chiếu tới cả bảng này.



Thể hiện mỗi quan hệ này một cách đầy đủ trong code bằng Hibernate thì chúng ta sẽ dùng @ManyToMany

Trong bài sử dụng các kiến thức:

1. [Hibernate là gì?](https://loda.me/hibernate-la-gi-loda1554623701594)
2. [Cách sử dụng Lombok để tiết kiệm thời gian code](https://loda.me/general-huong-dan-su-dung-lombok-giup-code-java-nhanh-hon-69-loda1552789752787)

**Tạo project**

Toàn bộ bài viết được up tại Github: [github.com/loda-kun/java-all](https://loda.me/github.com/loda-kun/java-all)

Chúng ta sẽ sử dụng Gradle để tạo một project có khai báo Spring Boot và Jpa để hỗ trợ cho việc demo @ManyToMany.

Các bạn có thể tự tạo 1 project Spring-boot với gradle đơn giản tại: [https://start.spring.io](https://start.spring.io/)

plugins {

id 'org.springframework.boot' version '2.1.4.RELEASE'

id 'java'

}

apply plugin: 'io.spring.dependency-management'

group 'me.loda.java'

version '1.0-SNAPSHOT'

sourceCompatibility = 1.8

configurations {

compileOnly {

extendsFrom annotationProcessor

}

}

repositories {

mavenCentral()

}

dependencies {

implementation 'org.springframework.boot:spring-boot-starter-data-jpa'

implementation 'org.springframework.boot:spring-boot-starter-web'

compileOnly 'org.projectlombok:lombok'

runtimeOnly 'com.h2database:h2'

annotationProcessor 'org.projectlombok:lombok'

testImplementation 'org.springframework.boot:spring-boot-starter-test'

}

Trong ứng dụng trên bạn sẽ thấy có com.h2database:h2. Đây là một **database**, tuy nhiên nó chỉ tồn tại trong bộ nhớ. Tức làm mỗi khi chạy chương trình này, nó sẽ tạo database trong RAM, và tắt chương trình đi nó sẽ mất.

Chúng ta sẽ sử dụng H2 thay cho MySql để cho.. tiện!

Khi tạo xong project, sẽ có thư mục như sau:



**Tạo Table**

Để tạo table, chúng ta tạo ra các Class tương ứng.

import java.util.Collection;

import javax.persistence.CascadeType;

import javax.persistence.Entity;

import javax.persistence.FetchType;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.JoinTable;

import javax.persistence.ManyToMany;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.EqualsAndHashCode;

import lombok.NoArgsConstructor;

import lombok.ToString;

@Entity // Đánh dấu đây là table trong db

@Data // lombok giúp generate các hàm constructor, get, set v.v.

@AllArgsConstructor

@NoArgsConstructor

@Builder

public class Address {

@Id //Đánh dấu là primary key

@GeneratedValue // Giúp tự động tăng

private Long id;

private String city;

private String province;

@ManyToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY)

// Quan hệ n-n với đối tượng ở dưới (Person) (1 địa điểm có nhiều người ở)

@EqualsAndHashCode.Exclude // không sử dụng trường này trong equals và hashcode

@ToString.Exclude // Khoonhg sử dụng trong toString()

@JoinTable(name = "address\_person", //Tạo ra một join Table tên là "address\_person"

joinColumns = @JoinColumn(name = "address\_id"), // TRong đó, khóa ngoại chính là address\_id trỏ tới class hiện tại (Address)

inverseJoinColumns = @JoinColumn(name = "person\_id") //Khóa ngoại thứ 2 trỏ tới thuộc tính ở dưới (Person)

)

private Collection<Person> persons;

}

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class Person {

@Id

@GeneratedValue

private Long id;

private String name;

// mappedBy trỏ tới tên biến persons ở trong Address.

@ManyToMany(mappedBy = "persons")

// LAZY để tránh việc truy xuất dữ liệu không cần thiết. Lúc nào cần thì mới query

@EqualsAndHashCode.Exclude

@Exclude

private Collection<Address> addresses;

}

Nếu chúng ta chưa tạo ra các table trong cơ sở dữ liệu, thì mặc định Hibernate sẽ bind dữ liệu từ class xuống và tạo table cho chúng ta.

Bạn phải tạo file config src\main\resources\application.properties như sau để kết nối tới H2 database nhé:

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

// Không có password, vào thẳng luôn

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

# Cho phép vào xem db thông qua web

spring.h2.console.enabled=true

**Chạy thử**

Bạn tạo file ManyToManyExampleApplication và cấu hình Spring Boot và khởi chạy chương trình.

@SpringBootApplication

@RequiredArgsConstructor

public class ManyToManyExampleApplication {

public static void main(String[] args) {

SpringApplication.run(ManyToManyExampleApplication.class, args);

}

}

Sau khi chạy xong, hãy truy cập vào <http://localhost:8080/h2-console/> để vào xem database có gì nhé.



Bạn sẽ thấy nó tạo table giống với mô tả ở đầu bài. Gồm có hai bảng chính là address và person. Ngoài ra, sẽ tạo ra một bảng trung gian ở giữa liên kết hai bảng là address\_person.

**Thêm dữ liệu**

Để thêm dữ liệu vào database, chúng ta sẽ dùng tới Spring JPA .

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

public interface AddressRepository extends JpaRepository<Address,Long> {

}

public interface PersonRepository extends JpaRepository<Person, Long> {

}

Chúng ta sẽ tạo một chương trình Spring Boot đơn giản bằng cách sử dụng CommandLineRunner để chạy code ngay khi khởi động.

import javax.transaction.Transactional;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import com.google.common.collect.Lists;

import lombok.RequiredArgsConstructor;

@SpringBootApplication

@RequiredArgsConstructor

public class ManyToManyExampleApplication implements CommandLineRunner {

public static void main(String[] args) {

SpringApplication.run(ManyToManyExampleApplication.class, args);

}

// Sử dụng @RequiredArgsConstructor và final để thay cho @Autowired

private final PersonRepository personRepository;

private final AddressRepository addressRepository;

@Override

@Transactional

public void run(String... args) throws Exception {

// Tạo ra đối tượng Address

Address hanoi = Address.builder()

.city("hanoi")

.build();

Address hatay = Address.builder()

.city("hatay")

.build();

// Tạo ra đối tượng person

Person person1 = Person.builder()

.name("loda1")

.build();

Person person2 = Person.builder()

.name("loda2")

.build();

// set Persons vào address

hanoi.setPersons(Lists.newArrayList(person1, person2));

hatay.setPersons(Lists.newArrayList(person1));

// Lưu vào db

// Chúng ta chỉ cần lưu address, vì cascade = CascadeType.ALL nên nó sẽ lưu luôn Person.

addressRepository.saveAndFlush(hanoi);

addressRepository.saveAndFlush(hatay);

// Vào:http://localhost:8080/h2-console/ để xem dữ liệu đã insert

Address queryResult = addressRepository.findById(1L).get();

System.out.println(queryResult.getCity());

System.out.println(queryResult.getPersons());

}

}

// Output:

// hanoi

// [Person(id=2, name=loda1), Person(id=3, name=loda2)]

Lưu ý ở đây chúng ta dùng @Transactional. Đê khiến toàn bộ code chạy trong hàm đều nằm trong Session quản lý của Hibernate.

Nếu không có @Transactional thì việc bạn gọi address.getPersons() sẽ bị lỗi, vì nó không thể query xuống database để lấy dữ liệu person lên được. Bạn ghi nhớ chỗ này nhé.

Kết quả trong database lúc này:

Address



Person



Address\_Person

