**Spring Boot MyBatis With MySQL**

1. **What's MyBatis?**

MyBatis là một persistence framework được hỗ trợ sẵn trong Spring Boot

Mybatis không truy vấn trên java object như các framework khác, mà truy vấn trưc tiếp vào table thông qua SQL query, và kết quả trả về được mapping vào java object. Do đó MyBatis chỉ có các tính năng như Create session, query data. còn Transaction thì không cung cấp mà dùng của SQL. Ngoài ra MyBatis cũng không lưu trữ cache như Hibernate hoặc JPA.

Việc sử dụng SQL Native ở Hibernate và JPA cũng có nhưng Mybatis đã tận dụng và phát huy sức mạnh kết hợp với Spring Boot làm giảm bớt phần cấu hình, giúp cho framework đơn giản, dễ học và phát triển ứng dụng nhanh và tốc độ truy vấn cao (do nó truy vấn trực tiếp vào table, bỏ qua bước ánh xạ các database table vào các Entity object trước khi truy vấn)

MyBatis khá đơn giản, dễ học và có kích thước nhỏ, số lượng các package ít hơn nhiều so với các persistence framework khác. Vì thế nếu xây dựng ứng dụng mà yêu cầu thời gian phát triển ngắn, cấu trúc database không quá phức tạp, thì Mybatis là lựa chọn thích hợp

1. **Chuẩn bị MySQL**

* **Install Xampp on Mac & Start MySQL Database https: //www.webucator.com/how-to/how-install-start-test-xampp-on-mac-osx.cfm**
* **Access MySql phpmyadmin:** <http://localhost:80/phpmyadminhttp>
* **Create Database:** <https://github.com/java-spring-training/spring-boot-api/blob/api-processing-flow/MySQL%20Script/product.sql>

1. **Implement MyBatis with Spring Boot**

**Vì MyBatis là 1 thành phần trong Spring boot, nên hầu như các config đều được xử lý tự động. Việc cần làm chỉ là khai báo thông tin connection và sử dụng**

**Thư viện**

|  |
| --- |
| **compile("org.mybatis.spring.boot:mybatis-spring-boot-starter:{version}")** |

* 1. ***Database Connection***

Application.yml

|  |
| --- |
| spring:  datasource:  url: jdbc:mysql://localhost:3306/product\_management?serverTimezone=UTC  driver: com.mysql.cj.jdbc.Driver  username: root  password: |

**Server database url: jdbc:mysql://{host}:{post}/{database name}**

driver: mỗi loại database có driver khác nhau

username: default username của local MySQL là “root”

password: default password của local MySQL là “”

* 1. ***Implement MyBatis***
* Chỉ có 1 step duy nhất là define Mapper interface để chứa các SQL query. Khi đó MyBatis sẽ execute SQL query và trả data về cho method tương ứng với SQL query được thực thi
* **Không cần phải define các implement cho các interface mapper, vì MyBatis sẽ tự làm việc đó**
* **Có 2 cách để tạo 1 Mapper:**
  + **Annotation-driven mapper (cách đơn giản nhất)**
  + **XML mapper (Cách này chỉ thích hợp cho các câu query phức tạp, hoặc query động)**
* **VD 1 interface Mapper**

|  |
| --- |
| **@Mapper**  **public interface ClientMapper {**  **@Select("SELECT id, first\_name, last\_name FROM customer WHERE id = #{id}")**  **Customer getById(int id);**  **@Insert("INSERT INTO customer (id, first\_name, last\_name) VALUES (#{id}, #{firstName}, #{lastName})")**  **void insertClient(Client client);**  **}** |

* **@Mapper: Để MyBatis nhận biết đây là 1 mapper interface**
* **@Select, @Insert: được dùng tùy vào câu SQL query**
* **@Select: return data**
* **@Insert: return số row đã insert, hoặc không return khi dùng void() như ví dụ trên**
* **Để biết thêm về các MyBatis Annotation:** <https://www.baeldung.com/mybatis> (internet)

1. **MyBatis @Select**

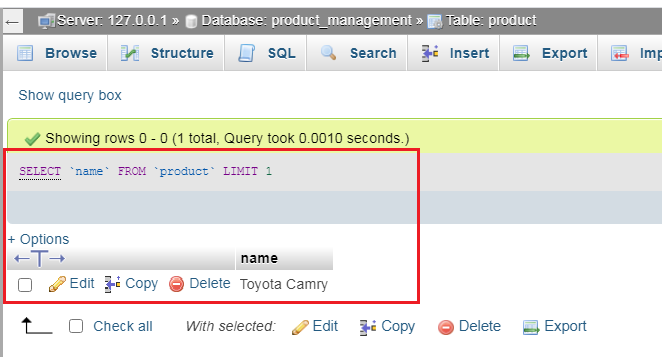
**VD1: SQL query đơn giản, return String**

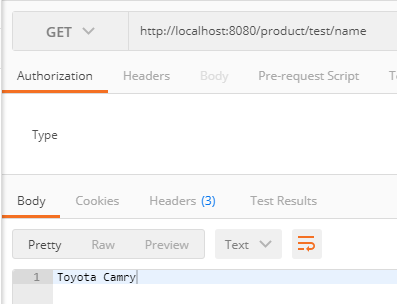
* **Step 1: ProductMapper.java**

|  |
| --- |
| **@Mapper**  **public interface ProductMapper {**  **@Select("SELECT `name` FROM `product` LIMIT 1")**  **String getFirstProductName();**  **}** |

* **Step 2: Call ProductMapper. getFirstProductName()**

|  |
| --- |
| **@Repository**  **public class ProductDbRepositoryImpl implements ProductDbRepository {**  **private ProductMapper mapper;**  **@Autowired**  **public ProductDbRepositoryImpl(final ProductMapper mapper) {**  **this.mapper = mapper;**  **}**  **@Override**  **public String getFirstProductName() {**  **return mapper.getFirstProductName();**  **}**  **}** |

****

****

**VD2: SQL query Return List object**

**ProductMapper.java**

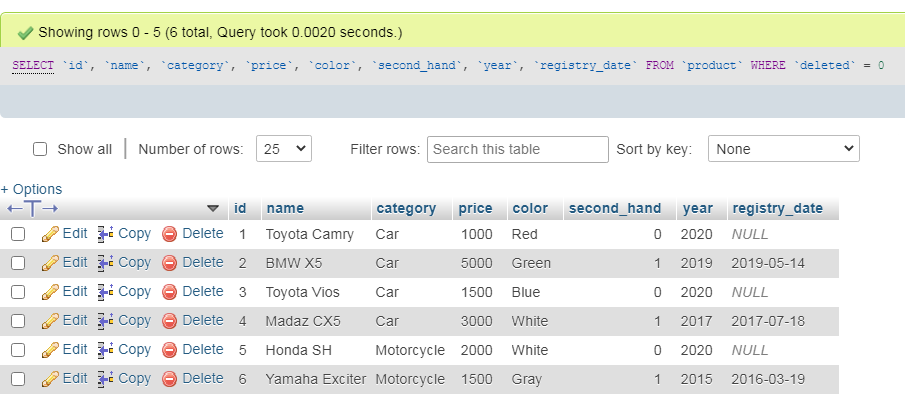
|  |
| --- |
| **@Select("SELECT `id`, `name`, `category`, `price`, `color`, `second\_hand`, `year`, `registry\_date` " +**  **"FROM `product` " +**  **"WHERE `deleted` = 0")**  **List<Product> getListAll();** |

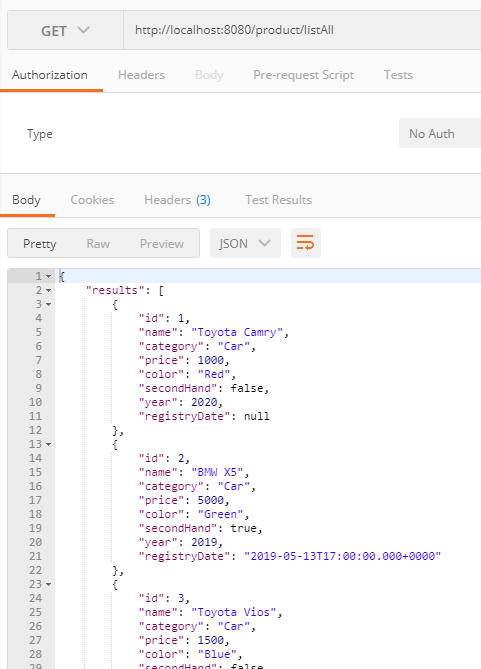
**Product.java**

|  |
| --- |
| **public Product(int id, String name, String category, int price, String color, boolean secondHand, int year, Date registryDate) {**  **// Validate output base on Database constrain. Ex: check require not null**  **Preconditions.checkArgument(!StringUtils.isEmpty(name), "name must not be empty");**  **Preconditions.checkArgument(price >= 100, "price must be greater than or equal 100");**  **// Others checking ...**  **this.id = id;**  **this.name = name;**  **this.category = category;**  **this.price = price;**  **this.color = color;**  **this.secondHand = secondHand;**  **this.year = year;**  **this.registryDate = registryDate;**  **}** |

**Lưu ý:**

* **Các item name trong object Product không cần phải giống với column name**
* **MyBatis sẽ tự động tìm Constructor có arguments tương ứng với dữ liệu mà nó select được**
* **Hoặc cũng có thể dùng Annotation chỉ định column name cho từng item name của object (tham khảo cách này trong tài liệu về MyBatis Annotation ở trên)**

****

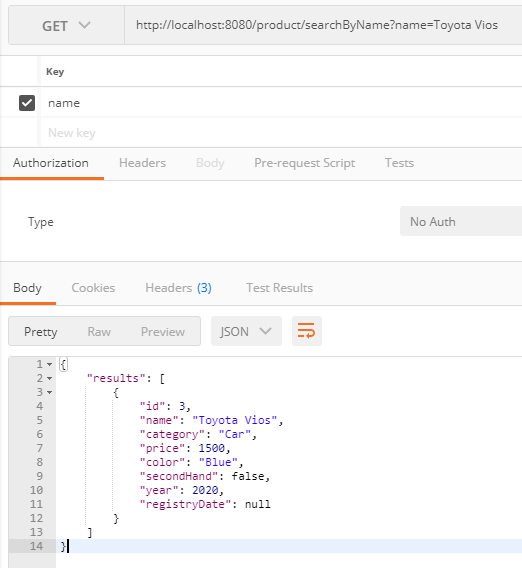
****

**VD3: Truyền parameter vào SQL query**

**ProductMapper.java**

|  |
| --- |
| **@Mapper**  **public interface ProductMapper {**  **@Select("SELECT `id`, `name`, `category`, `price`, `color`, `second\_hand`, `year`, `registry\_date` " +**  **"FROM `product` " +**  **"WHERE `deleted` = 0 AND `name` = #{name}")**  **List<Product> searchByName(String name);**  **}** |

****

****

**Lưu ý:**

* **Mapper không hiểu các ký tự đặc biệt được truyền vào SQL Query như: ‘, %, &, …**
* **Do đó nếu cần sử dụng các ký tự đặc biệt thì phải gắn sẵn các ký tự đặc biệt này vào parameter.**
* **Ví dụ như mệnh để LIKE trong SQL cần dùng %. Khi đó không được để ký tự % trong interface mapper, mà phải gắn % vào parameter trước khi attach vào câu SQL**

**ProductDbRepositoryImpl.java**

|  |
| --- |
| **@Repository**  **public class ProductDbRepositoryImpl implements ProductDbRepository {**  **private ProductMapper mapper;**  **@Autowired**  **public ProductDbRepositoryImpl(final ProductMapper mapper) {**  **this.mapper = mapper;**  **}**  **@Override**  **public List<Product> searchByName(String name) {**  **return mapper.searchByName("%" + name + "%");**  **//return mapper.searchByName(name);**  **}**  **}** |

**ProductMapper.java**

|  |
| --- |
| **@Select("SELECT `id`, `name`, `category`, `price`, `color`, `second\_hand`, `year`, `registry\_date` " +**  **"FROM `product` " +**  **"WHERE `deleted` = 0 AND `name` LIKE #{name}")**  **List<Product> searchByName(String name);** |