Spring Boot – Spring Data JPA Query or Finder Methods

Description

1.Introduction

In this post we will see how to fetch data from the database using Spring Data JPA finder methods. We can fetch data using any column name or using any other criteria we use in SQL queries like AND, OR, LIKE etc. We will use the in memory database i.e. h2 and try to write finder methods with some variations in it. If you plan to use some other database then the only thing you have to do is change the database specific connector dependency and the DB connection details.

2. Project Download

For creating the Spring Boot project, head over to https://start.spring.io and add â€œSpring Webâ€• , Spring Data JPA and H2 dependency. Once you download the project, import it as a Maven project into your favorite IDE.

3. Database Connectivity

```
server.port=9011
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.h2.console.enabled=true
```

In your application.properties file add the above configuration. The first line is self explanatory that our application will run on 9011 port and the rest are database connectivity details. If you want to see the SQL that gets executed on the eclipse console then add the last line.

4. Creating the Model class

We will create an Employee POJO class and play around with the data we insert against the table. From the annotation perspective, since Employee class will be mapped to Employee table, lets annotate the class with @Entity and annotate the primary key or the id field with @Id and @GeneratedValue annotation.

```
package com.springmicroservices.model;
```

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="Employee")
public class Employee {

 @Id
 @GeneratedValue
 public Integer empId;
 public String empName;
 public String empCity;
 public Integer empSalary;

 public Integer getEmpId() {
  return empId;
 }
 public void setEmpId(Integer empId) {
  this.empId = empId;
 }
 public String getEmpName() {
  return empName;
 }
 public void setEmpName(String empName) {
  this.empName = empName;
 }
 public String getEmpCity() {
  return empCity;
 }
 public void setEmpCity(String empCity) {
  this.empCity = empCity;
 }
 public Integer getEmpSalary() {
  return empSalary;
 }
 public void setEmpSalary(Integer empSalary) {
  this.empSalary = empSalary;
 }
 @Override
 public String toString() {
  return "Employee [empId=" + empId + ", empName=" + empName + ", empCity=" +
    + "]";
 }
}
```


5. Creating the Repository Interface

Lets create the EmployeeRepository interface using which we will be performing the find operations against the database. To enable this capability we will extend the EmployeeRepository with the CrudRepository interface.

If you see the below code, in the angular brackets we have given the Employee class name followed by Integer. We have given Employee class name because this repository is responsible to perform operation against the Employee table and the Integer is the datatype of the primary key.

```
package com.springmicroservices.repository;
import java.util.List;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;
import com.springmicroservices.model.Employee;

@Repository
public interface EmployeeRepository extends CrudRepository<Employee, Integer>
 List<Employee> findByEmpName(String name);
 List<Employee> findByEmpCity(String city);
 List<Employee> findByEmpSalaryGreaterThan(Integer salary);
}
```

The way to write a finder method is by appending the domain class attribute name after “findBy”. If you see the Employee POJO class, one of the attribute name we have is empName. If you want to write a method which will fetch the records based on the employee name then the method name will be findByEmpName, and pass in the parameter which you want to use as a criteria.

The equivalent SQL query that will get generated out of finByEmpName will be **“select * from employee where empname=’Ravi’“**

Apart from writing simple finder methods, we can write complex finder methods. For example if you want to find all the employees who’s salary is greater than a specific value then the finder method will be **findByEmpSalaryGreaterThan**. Below are few of the sample implementations.

Logical keyword	Keyword expressions
BETWEEN	Between, IsBetween
GREATER_THAN	GreaterThan, IsGreaterThan
STARTING_WITH	StartingWith, IsStartingWith, StartsWith
IS_NOT_NULL	NotNull, IsNotNull

If you want to explore more on how to write method names by which you can fetch the data, please visit the link Finder methods reference guide

6. Creating the Controller Class

Lets create EmployeeController class which will contain endpoints that can be invoked through POSTMAN. These controller methods will invoke the methods of EmployeeRepository. We will have to autowire EmployeeRepository class inside the controller class.

Method 1 :- getEmployeebyId controller method will fetch the employee data based on the passed

employee Id.

Method 2 :- getEmployeebyName controller method will fetch the employee data based on the passed employee name.

Method 3 :- getEmployeebyCity controller method will fetch the employee data based on the passed employee City.

Method 4 :- getEmployeesBasedOnSalary controller method will fetch the employee data whose salary is greater than the passed salary.

```
package com.springmicroservices.controller;
import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
import com.springmicroservices.model.Employee;
import com.springmicroservices.repository.EmployeeRepository;

@RestController
public class EmployeeController {

 @Autowired
 EmployeeRepository employeeRepository;

 @GetMapping("/employees/employeeid/{empId}")
 public Employee getEmployeebyId(@PathVariable("empId") Integer empId) {

  //If the ID you are trying to look for is not present then
  //you will get "java.util.NoSuchElementException: No value present" exceptio

  System.out.println(employeeRepository.findById(empId));
  Optional<Employee> empListByName = employeeRepository.findById(empId);
  return empListByName.get();
 }


 @GetMapping("/employees/employeename/{empName}")
 public List<Employee> getEmployeebyName(@PathVariable("empName") String empNa

  System.out.println(employeeRepository.findByEmpName(empName));
  List<Employee> empListByName = employeeRepository.findByEmpName(empName);
  return empListByName;
 }

 @GetMapping("/employees/employeecity/{empCity}")
 public List<Employee> getEmployeebyCity(@PathVariable("empCity") String empCi

  System.out.println(employeeRepository.findByEmpCity(empCity));
  List<Employee> empListByCity = employeeRepository.findByEmpCity(empCity);
```

```
  return empListByCity;
 }

 @GetMapping("/employees/salary/{minSalary}")
 public List<Employee> getEmployeesBasedOnSalary(@PathVariable("minSalary") In

  List<Employee> empListByMinSalary = employeeRepository.findByEmpSalaryGreate
  System.out.println(empListByMinSalary);
  return empListByMinSalary;
 }

}
```

You can download the project from the below GIT repository. Detailed documentation is provided on how to execute the project.

Git Repository – https://github.com/RavikiranKada/H2-FinderMethodsSpringJPA

Category

1. Hands on
2. Spring Boot

Date Created
April 29, 2021
Author
kk-ravi144gmail-com