

Session: Spring Data JPA with Hibernate Part 1

Assignment

(1) Create an Employee Entity which contains following fields

Name, Id, Age, Location

```
@Entity
public class Employee {

    @Id
    @Column(name = "empid")
    private int id;
    @Column(name = "empname")
    private String name;
    private int age;
    private String location;

    public Employee() {
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

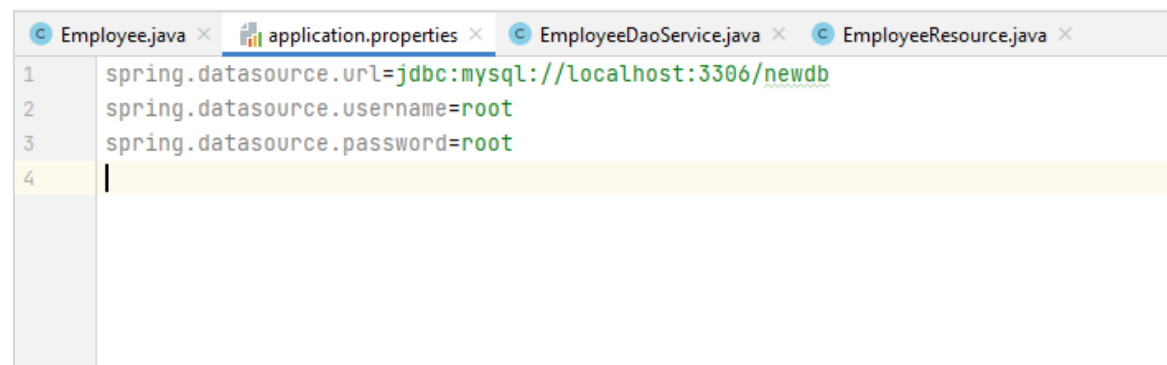
    public void setAge(int age) {
        this.age = age;
    }

    public String getLocation() {
        return location;
    }

    public void setLocation(String location) {
        this.location = location;
    }
}
```

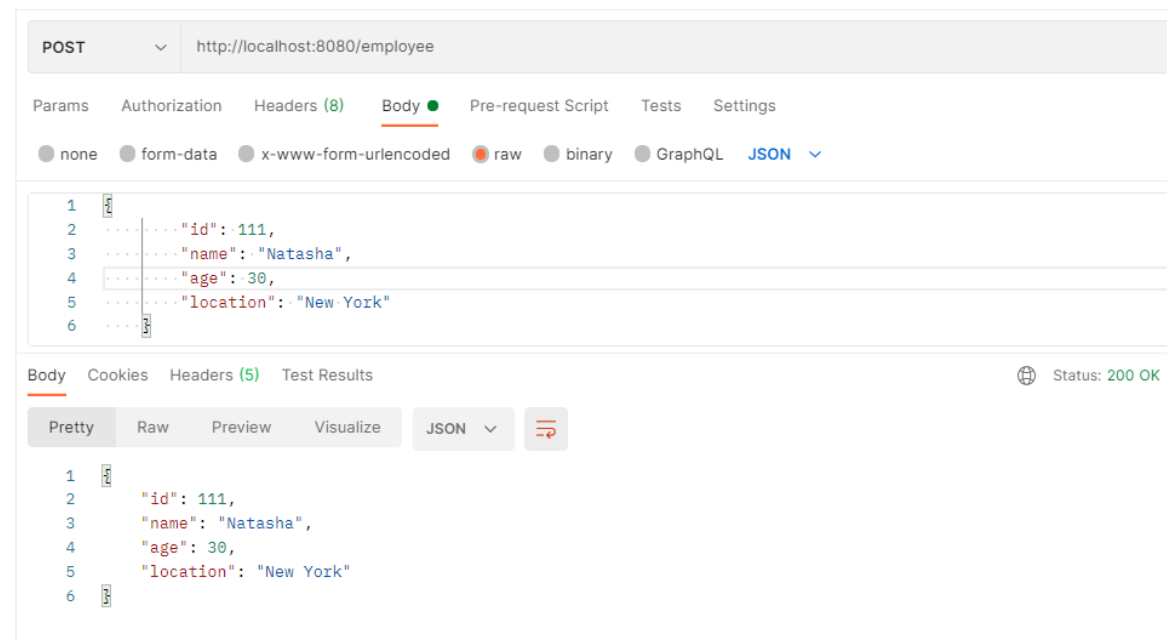
(2) Set up Employee Repository with Spring Data JPA

```
}      <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
}    </dependency>
```

A screenshot of an IDE with several tabs open: Employee.java, application.properties, EmployeeDaoService.java, and EmployeeResource.java. The 'application.properties' tab is active, showing the following configuration:

```
1 spring.datasource.url=jdbc:mysql://localhost:3306/newdb
2 spring.datasource.username=root
3 spring.datasource.password=root
4
```

(3) Perform Create Operation on Entity using Spring Data JPA

A screenshot of a REST client interface. The top bar shows a POST request to 'http://localhost:8080/employee'. Below the bar, the 'Body' tab is selected, showing a JSON payload:

```
1 {
2   "id": 111,
3   "name": "Natasha",
4   "age": 30,
5   "location": "New York"
6 }
```

The bottom section shows the response, with the 'Body' tab selected, displaying the same JSON payload in a pretty-printed format. The status bar at the bottom right indicates 'Status: 200 OK'.

(4) Perform Update Operation on Entity using Spring Data JPA

The screenshot displays a REST client interface for a PUT request. The URL is `http://localhost:8080/employee/111`. The request body is a JSON object representing an employee with the following fields: `id` (111), `name` (Natasha), `age` (32), and `location` (New York). The status bar at the bottom right indicates a successful response with `Status: 200 OK`.

```
1 PUT http://localhost:8080/employee/111
2
3 {
4   "id": 111,
5   "name": "Natasha",
6   "age": 32,
7   "location": "New York"
8 }
```

Body Cookies Headers (5) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 111,
3   "name": "Natasha",
4   "age": 32,
5   "location": "New York"
6 }
```

(5) Perform Delete Operation on Entity using Spring Data JPA

The screenshot displays a REST client interface for a DELETE request. The URL is `http://localhost:8080/employee/111`. The status bar at the bottom right indicates a successful response with `Status: 200 OK`.

DELETE http://localhost:8080/employee/111

Params Authorization Headers (4) Body Pre-request Script Tests Settings

Body Cookies Headers (4) Test Results Status: 200 OK

Pretty Raw Preview Visualize Text

```
1
```

(6) Perform Read Operation on Entity using Spring Data JPA



```
⏪ ⏩ ↺ ⓘ localhost:8080/employee
▼ [
  ▼ {
    "id": 101,
    "name": "John",
    "age": 32,
    "location": "New York"
  },
  ▼ {
    "id": 102,
    "name": "Alex",
    "age": 31,
    "location": "Dallas"
  },
  ▼ {
    "id": 103,
    "name": "Clark",
    "age": 22,
    "location": "Chicago"
  },
  ▼ {
    "id": 104,
    "name": "Scott",
    "age": 25,
    "location": "Boston"
  },
  ▼ {
    "id": 105,
    "name": "Adams",
    "age": 29,
    "location": "London"
  },
  ▼ {
    "id": 106,
    "name": "Justin",
    "age": 30,
    "location": "Paris"
  },
],
```

(7) Get the total count of the number of Employees

← → ↻ ⓘ localhost:8080/employee/count

10

(8) Implement Pagination and Sorting on the bases of Employee Age

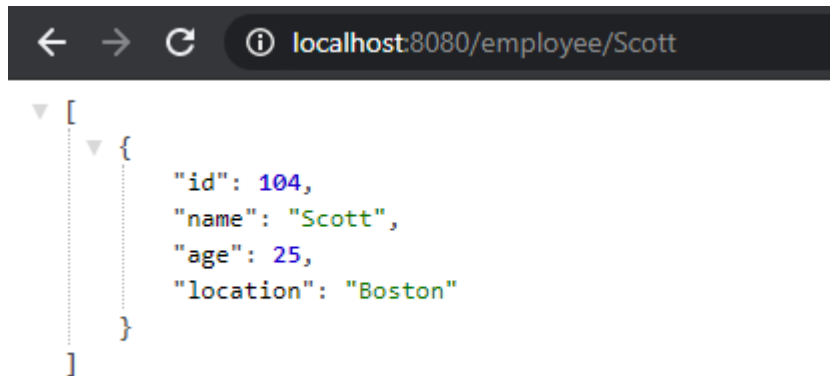
← → ↻ ⓘ localhost:8080/employee/paging?no=0&size=2

```
▼ [
  ▼ {
    "id": 103,
    "name": "Clark",
    "age": 22,
    "location": "Chicago"
  },
  ▼ {
    "id": 104,
    "name": "Scott",
    "age": 25,
    "location": "Boston"
  }
]
```

← → ↻ ⓘ localhost:8080/employee/paging?no=1&size=2

```
▼ [
  ▼ {
    "id": 110,
    "name": "Johnny",
    "age": 27,
    "location": "Barcelona"
  },
  ▼ {
    "id": 105,
    "name": "Adams",
    "age": 29,
    "location": "London"
  }
]
```

(9) Create and use finder to find Employee by Name



```
localhost:8080/employee/Scott
[
  {
    "id": 104,
    "name": "Scott",
    "age": 25,
    "location": "Boston"
  }
]
```

(10) Create and use finder to find Employees starting with A character



```
localhost:8080/employee/likeA
[
  {
    "id": 102,
    "name": "Alex",
    "age": 31,
    "location": "Dallas"
  },
  {
    "id": 105,
    "name": "Adams",
    "age": 29,
    "location": "London"
  },
  {
    "id": 107,
    "name": "Andrew",
    "age": 31,
    "location": "Los Angeles"
  }
]
```

(11) Create and use finder to find Employees Between the age of 28 to 32



The screenshot shows a web browser window with the address bar displaying `localhost:8080/employee/age`. Below the address bar, a REST client interface displays a JSON array of employee objects. The array contains six objects, each with the following fields: `id`, `name`, `age`, and `location`. The objects are expanded to show their details.

```
[
  {
    "id": 101,
    "name": "John",
    "age": 32,
    "location": "New York"
  },
  {
    "id": 102,
    "name": "Alex",
    "age": 31,
    "location": "Dallas"
  },
  {
    "id": 105,
    "name": "Adams",
    "age": 29,
    "location": "London"
  },
  {
    "id": 106,
    "name": "Justin",
    "age": 30,
    "location": "Paris"
  },
  {
    "id": 107,
    "name": "Andrew",
    "age": 31,
    "location": "Los Angeles"
  },
  {
    "id": 108,
    "name": "Smith",
    "age": 30,
    "location": "Madrid"
  }
]
```