

Spring Data JPA part 1 Assignment

(1) Create an Employee Entity which contains following fields

Name

Id

Age

Location

CODE

Employee.java

```
@Entity
public class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    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;
}
}

```

OUTPUT

```

Database changed
mysql> use SpringDataJPA1;
Database changed
mysql> show tables;
+-----+
| Tables_in_SpringDataJPA1 |
+-----+
| employee                  |
+-----+
1 row in set (0.00 sec)

mysql> desc employee;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra           |
+-----+-----+-----+-----+-----+-----+
| id     | int(11)       | NO   | PRI | NULL    | auto_increment |
| age    | int(11)       | NO   |     | NULL    |                 |
| location | varchar(255) | YES  |     | NULL    |                 |
| name   | varchar(255) | YES  |     | NULL    |                 |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

```

(2) Set up EmployeeRepository with Spring Data JPA

CODE

@Repository

```
public interface EmployeeRepository extends CrudRepository<Employee,Integer> {  
  
}
```

OUTPUT



```
EmployeeController.java x EmployeeRepository.java x Employee.java x SpringDataJpaA  
package com.SpringData.JPAwithHibernate.SpringDataJPA_Assignment1.repository;  
  
import com.SpringData.JPAwithHibernate.SpringDataJPA_Assignment1.entity.Employee;  
import org.springframework.data.repository.CrudRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface EmployeeRepository extends CrudRepository<Employee,Integer> {  
  
}
```

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

CODE

@RestController

@RequestMapping("/Employees")

```
public class EmployeeController {
```

@Autowired

```
EmployeeRepository employeeRepository;
```

@GetMapping("/create")

```
public String addEmployee(){  
    Employee employee = new Employee();  
    employee.setName("Aayushi");  
    employee.setAge(24);  
    employee.setLocation("Delhi");  
    employeeRepository.save(employee);  
    return "Employee Added";  
}
```

}

OUTPUT

The screenshot shows a REST client interface. The top bar indicates a GET request to `http://localhost:8080/Employees/create`. Below the bar, tabs for Params, Authorization, Headers (7), Body, Pre-request Script, Tests, and Settings are visible. The 'Query Params' section is expanded, showing a table with columns KEY, VALUE, and DESCRIPTION. The table contains one entry: Key, Value, Description. Below this, the 'Body' tab is selected, showing 'Employee Added'. The status bar at the bottom right indicates 'Status: 200 OK' and 'Time: 210'.

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (5) Test Results Status: 200 OK Time: 210

Pretty Raw Preview Visualize Text

1 Employee Added

```
mysql> select * from employee;
+----+-----+-----+-----+
| id | age | location | name |
+----+-----+-----+-----+
| 1  | 24  | Delhi    | Aayushi |
+----+-----+-----+-----+
1 row in set (0.00 sec)
```

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

CODE

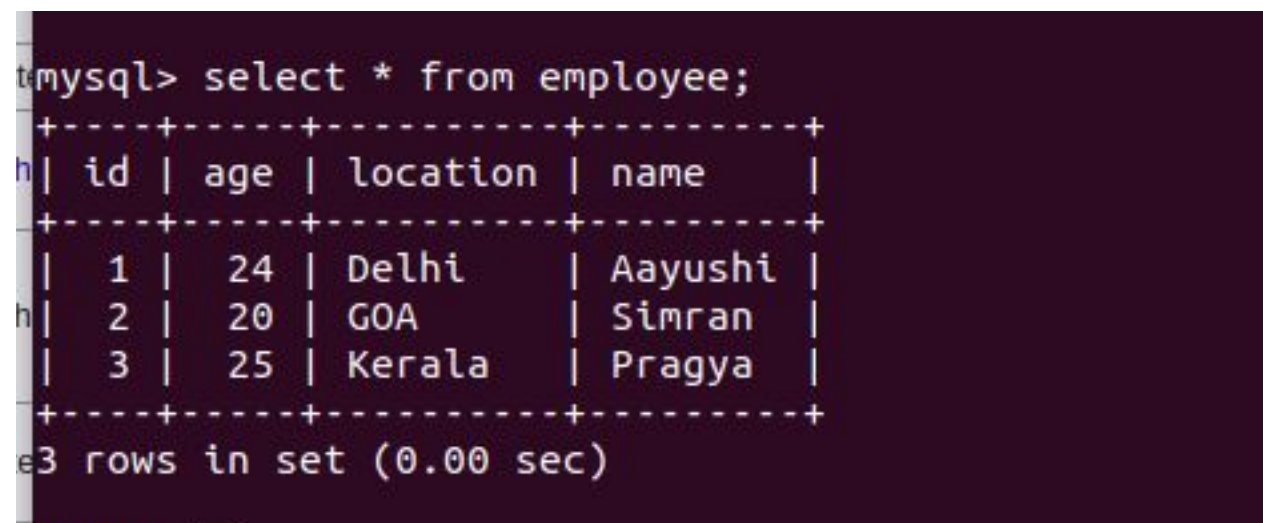
```
//UPDATE OPERATION
@GetMapping("/update")
```

```

public String updateEmployee(){
    Optional<Employee> employee = employeeRepository.findById(3);
    if (employee.isPresent())
    {
        Employee employee1 = employee.get();
        employee1.setLocation("Chandigarh");
        employee1.setAge(22);
        employeeRepository.save(employee1);
        return "Record Updated";
    }
    else
        return "Record Not found";
}

```

OUTPUT



```

mysql> select * from employee;
+----+-----+-----+-----+
| id | age | location | name |
+----+-----+-----+-----+
| 1 | 24 | Delhi | Aayushi |
| 2 | 20 | GOA | Simran |
| 3 | 25 | Kerala | Pragya |
+----+-----+-----+-----+
3 rows in set (0.00 sec)

```

GET http://localhost:8080/Employees/update

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
Key	Value

Body Cookies Headers (5) Test Results Status

Pretty Raw Preview Visualize Text

1 Record Updated

```
mysql> select * from employee;
+----+-----+-----+-----+
| id | age | location | name |
+----+-----+-----+-----+
| 1  | 24  | Delhi    | Aayushi |
| 2  | 20  | GOA      | Simran |
| 3  | 25  | Kerala   | Pragya |
+----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> select * from employee;
+----+-----+-----+-----+
| id | age | location | name |
+----+-----+-----+-----+
| 1  | 24  | Delhi    | Aayushi |
| 2  | 20  | GOA      | Simran |
| 3  | 22  | Chandigarh | Pragya |
+----+-----+-----+-----+
3 rows in set (0.00 sec)
```

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

CODE

//DELETE OPERATION

@GetMapping("/delete/{id}")

public String deleteEmployee(@PathVariable Integer id){

Optional<Employee> employee = employeeRepository.findById(id);

if (employee.isPresent())

{

employeeRepository.deleteById(id);

return "Record Deleted";

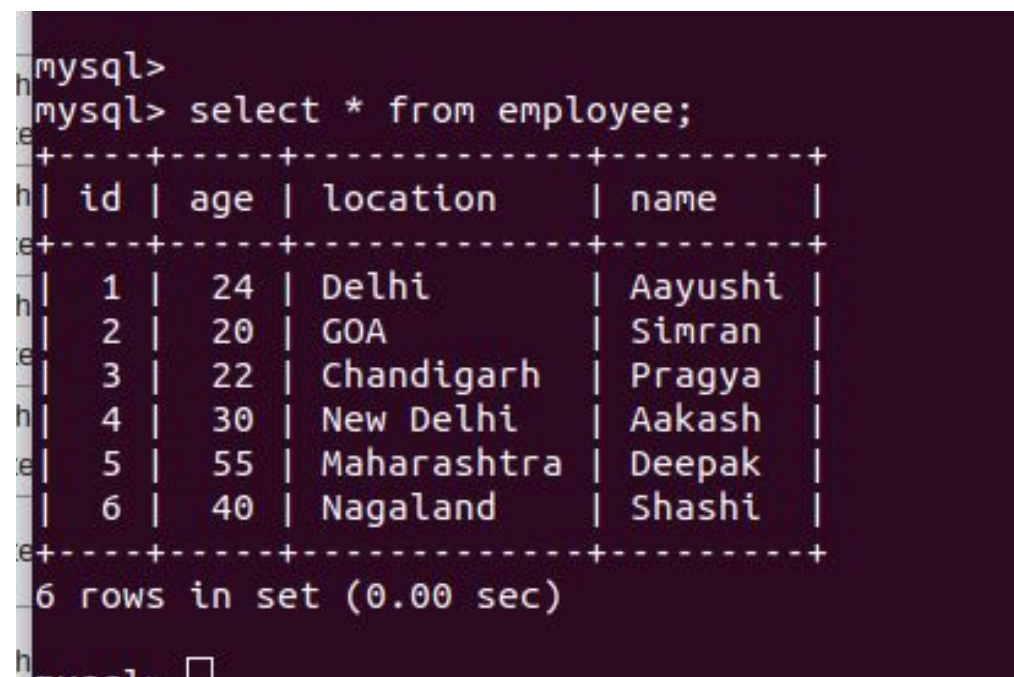
}

else

return "Record Not found";

}

OUTPUT



The screenshot shows a MySQL terminal session. The user enters the command 'mysql> select * from employee;' and the database returns a table with 6 rows. The columns are 'id', 'age', 'location', and 'name'. The data is as follows:

id	age	location	name
1	24	Delhi	Aayushi
2	20	GOA	Simran
3	22	Chandigarh	Pragya
4	30	New Delhi	Aakash
5	55	Maharashtra	Deepak
6	40	Nagaland	Shashi

Below the table, the terminal shows '6 rows in set (0.00 sec)'. The prompt 'mysql>' is visible at the bottom left.

Untitled Request

GET



http://localhost:8080/Employees/delete/2

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Query Params

	KEY	VALUE
	Key	Value

Body

Cookies

Headers (5)

Test Results

Pretty

Raw

Preview

Visualize

Text



1 Record Deleted


```
mysql>
```

```
mysql> select * from employee;
```

id	age	location	name
1	24	Delhi	Aayushi
2	20	GOA	Simran
3	22	Chandigarh	Pragya
4	30	New Delhi	Aakash
5	55	Maharashtra	Deepak
6	40	Nagaland	Shashi

```
6 rows in set (0.00 sec)
```

```
mysql> select * from employee;
```

id	age	location	name
1	24	Delhi	Aayushi
3	22	Chandigarh	Pragya
4	30	New Delhi	Aakash
5	55	Maharashtra	Deepak
6	40	Nagaland	Shashi

```
5 rows in set (0.00 sec)
```

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

CODE

//READ OPERATION

@GetMapping("/read")

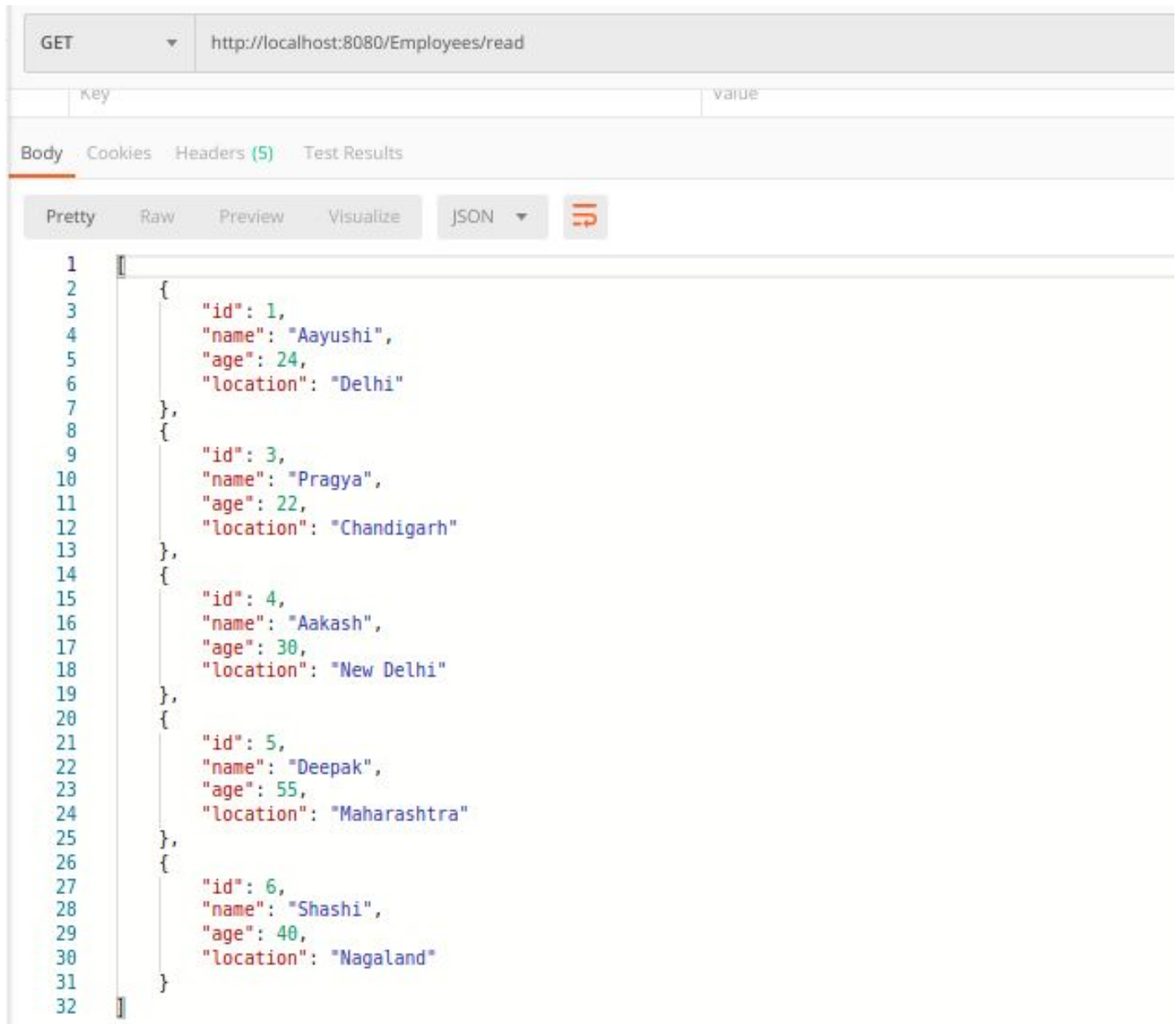
public List<Employee> readEmployee(){

 List<Employee> employeeList = (List<Employee>) employeeRepository.findAll();

 return employeeList;

}

OUTPUT



The screenshot shows a web browser interface with a GET request to `http://localhost:8080/Employees/read`. The response is displayed in the 'Body' tab, formatted as JSON. The JSON array contains six employee objects, each with fields for id, name, age, and location.

```
1 {
2   {
3     "id": 1,
4     "name": "Aayushi",
5     "age": 24,
6     "location": "Delhi"
7   },
8   {
9     "id": 3,
10    "name": "Pragya",
11    "age": 22,
12    "location": "Chandigarh"
13  },
14  {
15    "id": 4,
16    "name": "Aakash",
17    "age": 30,
18    "location": "New Delhi"
19  },
20  {
21    "id": 5,
22    "name": "Deepak",
23    "age": 55,
24    "location": "Maharashtra"
25  },
26  {
27    "id": 6,
28    "name": "Shashi",
29    "age": 40,
30    "location": "Nagaland"
31  }
32 }
```

(6) Get the total count of the number of Employees.

CODE

```
@GetMapping("/count")
public String countEmployee(){
    Long count = employeeRepository.count();
    return "Total No. Of Records Are:" +count;
}
```

OUTPUT

The screenshot shows a REST client interface. At the top, a GET request is configured for the URL `http://localhost:8080/Employees/count`. Below the URL bar, there are tabs for Params, Authorization, Headers (7), Body, Pre-request Script, Tests, and Settings. The Params tab is active, showing a table for Query Params with two columns: KEY and VALUE. The first row contains 'Key' and 'Value'. Below the Params tab, there are tabs for Body, Cookies, Headers (5), and Test Results. The Body tab is active, showing a text response: `1 Total No. Of Records Are:5`. The response is displayed in a 'Text' format, as indicated by the 'Text' dropdown menu.

KEY	VALUE
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize Text ▼

1 Total No. Of Records Are:5

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

CODE

EmployeeRepository.java

```
@Repository
public interface EmployeeRepository extends CrudRepository<Employee,Integer> {

    List<Employee> findByName(String name);

    List<Employee> findByNameLike(String desc);

    List<Employee> findByAgeBetween(Integer age1,Integer age2);
}
```

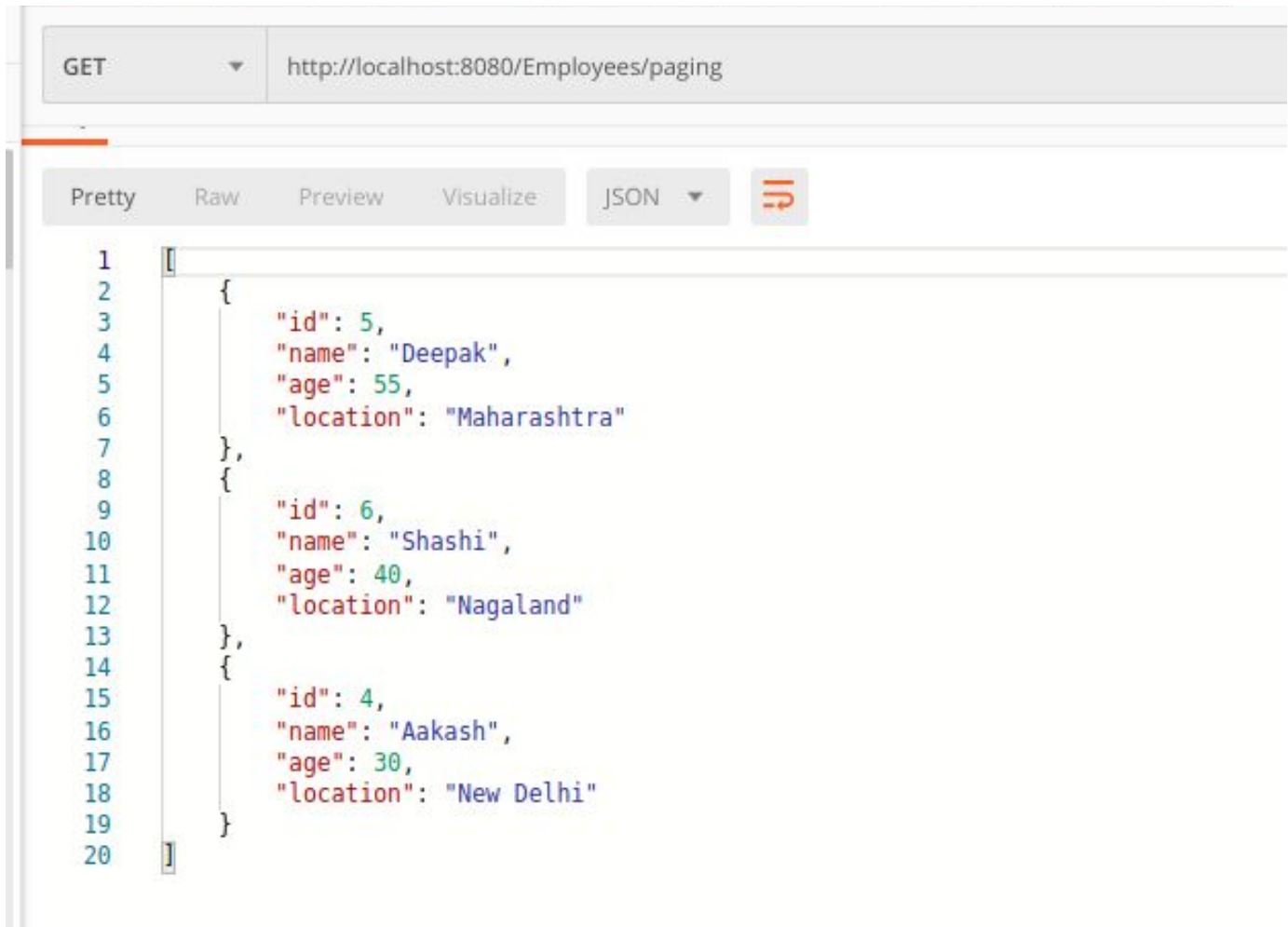
```
List<Employee> findAll(Pageable pageable);
}
```

EmployeeController.java

```
//PAGING AND SORTING
@GetMapping("/paging")
public List<Employee> employeeAgePagable(){
    Pageable pageable = PageRequest.of(0,3, Sort.Direction.DESC,"age");
    List<Employee> employeeList = (List<Employee>) employeeRepository.findAll(pageable);
    return employeeList;
}
```

OUTPUT

```
mysql> select * from employee;
+----+-----+-----+-----+
| id | age | location      | name    |
+----+-----+-----+-----+
| 1  | 24  | Delhi        | Aayushi |
| 3  | 22  | Chandigarh   | Pragya  |
| 4  | 30  | New Delhi    | Aakash  |
| 5  | 55  | Maharashtra  | Deepak  |
| 6  | 40  | Nagaland     | Shashi  |
+----+-----+-----+-----+
5 rows in set (0.00 sec)
```



(8) Create and use finder to find Employee by Name.

CODE

EmployeeRepository.java

```
@Repository
public interface EmployeeRepository extends CrudRepository<Employee,Integer> {

    List<Employee> findByName(String name);
}
```

EmployeeController.java

```
//FIND BY NAME OPERATION
@GetMapping("/findByName/{name}")
public List<Employee> findEmployeeByName(@PathVariable String name){
    List<Employee> employeeList = (List<Employee>) employeeRepository.findByName(name);
    return employeeList;
}
```

OUTPUT

The screenshot shows a REST client interface with a GET request to `http://localhost:8080/Employees/findByName/Deepak`. The response body is displayed in JSON format, showing a single employee record.

KEY	VALUE
Key	Value

Body	Cookies	Headers (5)	Test Results
<pre>1 [2 { 3 "id": 5, 4 "name": "Deepak", 5 "age": 55, 6 "location": "Maharashtra" 7 } 8]</pre>			

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

CODE

EmployeeRepository.java

```
@Repository
public interface EmployeeRepository extends CrudRepository<Employee,Integer> {

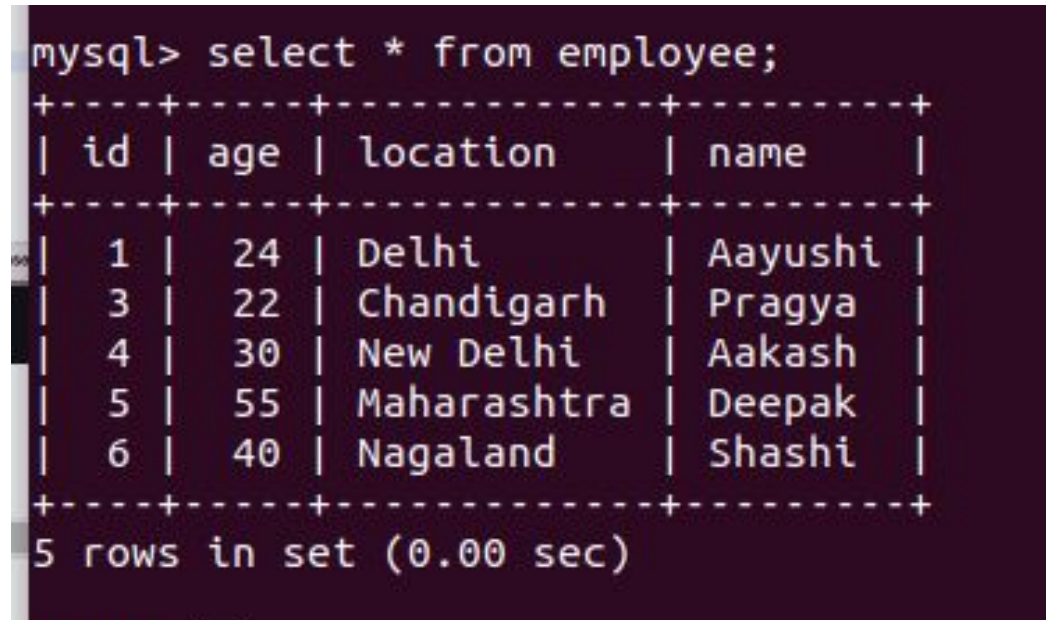
    List<Employee> findByNameLike(String desc);
}
```

EmployeeController.java

```
//FIND BY NAME STARTING WITH 'A' OPERATION
```

```
@GetMapping("/findByNameLike")
public List<Employee> findEmployeeByNameLike(){
    List<Employee> employeeList = (List<Employee>)
employeeRepository.findByNameLike("A%");
    return employeeList;
}
```

OUTPUT



The screenshot shows a MySQL terminal window with a dark background. The command `mysql> select * from employee;` has been executed. The output is a table with 5 rows and 4 columns: `id`, `age`, `location`, and `name`. The rows contain the following data: (1, 24, Delhi, Aayushi), (3, 22, Chandigarh, Pragya), (4, 30, New Delhi, Aakash), (5, 55, Maharashtra, Deepak), and (6, 40, Nagaland, Shashi). The terminal also displays `5 rows in set (0.00 sec)` at the bottom.

id	age	location	name
1	24	Delhi	Aayushi
3	22	Chandigarh	Pragya
4	30	New Delhi	Aakash
5	55	Maharashtra	Deepak
6	40	Nagaland	Shashi

5 rows in set (0.00 sec)

GET http://localhost:8080/Employees/findByAgeLike

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1  [
2    {
3      "id": 1,
4      "name": "Aayushi",
5      "age": 24,
6      "location": "Delhi"
7    },
8    {
9      "id": 4,
10     "name": "Aakash",
11     "age": 30,
12     "location": "New Delhi"
13   }
14 ]
```

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

CODE

```
//FIND BY AGE BETWEEN 28 TO 32
@GetMapping("/findByAgeBetween")
public List<Employee> findEmployeeByAgeBetween(){
    List<Employee> employeeList = (List<Employee>)
employeeRepository.findByAgeBetween(28,32);
    return employeeList;
}
```


OUTPUT

The screenshot displays a REST client interface with the following components:

- Request Bar:** Method: GET, URL: `http://localhost:8080/Employees/findByAgeBetween`
- Request Tabs:** Params, Authorization, Headers (7), Body, Pre-request Script, Tests, Settings. The 'Params' tab is selected.
- Query Params Table:**

KEY	VALUE
Key	Value
- Response Tabs:** Body, Cookies, Headers (5), Test Results. The 'Body' tab is selected.
- Response Format:** Pretty, Raw, Preview, Visualize. The format is set to JSON.
- Response Body:**

```
1  [
2    {
3      "id": 4,
4      "name": "Aakash",
5      "age": 30,
6      "location": "New Delhi"
7    }
8  ]
```