# Spring Boot JPA - Named Queries

Some time case arises, where we need a custom query to fulfil one test case. We can use @NamedQuery annotation to specify a named query within an entity class and then declare that method in repository. Following is an example.

We've added custom methods in Repository in JPA Custom Methods chapter. Now let's add another method using @NamedQuery and test it.

## Entity - Entity.java

Following is the default code of Employee. It represents a Employee table with id, name, age and email columns.

```
package com.tutorialspoint.entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

@Entity
@Table
@NamedQuery(name = "Employee.findByEmail",
query = "select e from Employee e where e.email = ?1")
public class Employee {
   @Id
   @Column
   private int id;

   @Column
   private String name;

   @Column
   private int age;

   @Column
   private String email;

   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 getEmail() {
       return email;
    }
    public void setEmail(String email) {
       this.email = email;
    }
}
```

## Repository - EmployeeRepository.java

Add a method to find an employee by its name and age.

```
package com.tutorialspoint.repository;

import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;
import com.tutorialspoint.entity.Employee;

@Repository
public interface EmployeeRepository extends CrudRepository<Employee, Intege
   public List<Employee> findByName(String name);
   public List<Employee> findByAge(int age);
   public Employee findByEmail(String email);
}
```

Now Spring JPA will create the implementation of above methods automatically using the query provided in named query. Let's test the methods added by adding their test cases in test file. Last two methods of below file tests the named query method added.

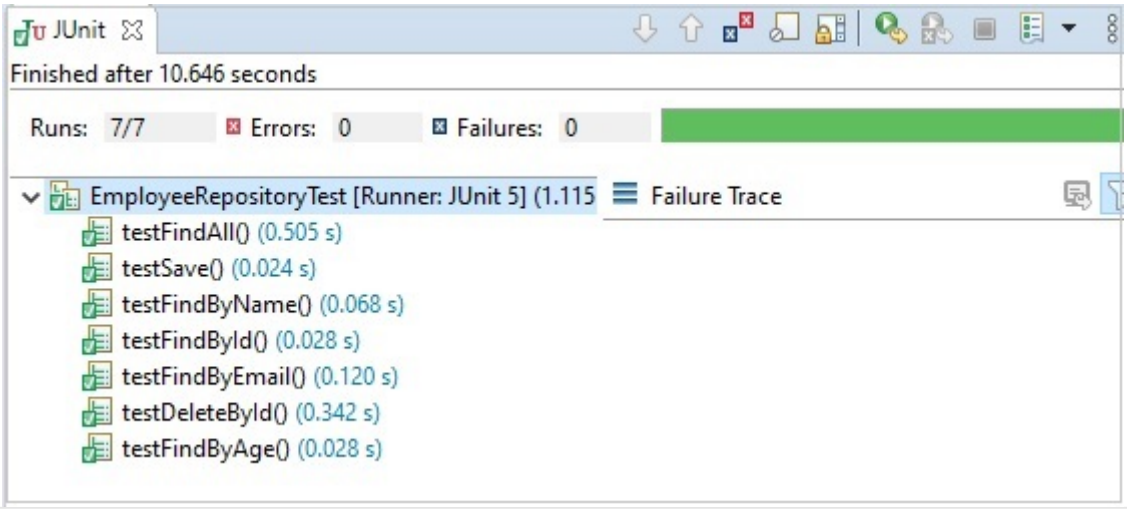
Following is the complete code of EmployeeRepositoryTest.

```
package com.tutorialspoint.repository;

import static org.junit.jupiter.api.Assertions.assertEquals;
import java.util.ArrayList;
import java.util.List;
import javax.transaction.Transactional;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit.jupiter.SpringExtension;
import com.tutorialspoint.entity.Employee;
import com.tutorialspoint.sprintbooth2.SprintBootH2Application;

@ExtendWith(SpringExtension.class)
@Transactional
@SpringBootTest(classes = SprintBootH2Application.class)
public class EmployeeRepositoryTest {
   @Autowired
   private EmployeeRepository employeeRepository;

   @Test
   public void testFindById() {
      Employee employee = getEmployee();
      employeeRepository.save(employee);
      Employee result = employeeRepository.findById(employee.getId()).get()
      assertEquals(employee.getId(), result.getId());
   }
   @Test
   public void testFindAll() {
      Employee employee = getEmployee();
      employeeRepository.save(employee);
      List<Employee> result = new ArrayList<>();
      employeeRepository.findAll().forEach(e -> result.add(e));
      assertEquals(result.size(), 1);
   }
   @Test
   public void testSave() {
      Employee employee = getEmployee();
      employeeRepository.save(employee);
      Employee found = employeeRepository.findById(employee.getId()).get()
      assertEquals(employee.getId(), found.getId());
   }
   @Test
```

```
    public void testDeleteById() {
       Employee employee = getEmployee();
       employeeRepository.save(employee);
       employeeRepository.deleteById(employee.getId());
       List<Employee> result = new ArrayList<>();
       employeeRepository.findAll().forEach(e -> result.add(e));
       assertEquals(result.size(), 0);
    }
    private Employee getEmployee() {
       Employee employee = new Employee();
       employee.setId(1);
       employee.setName("Mahesh");
       employee.setAge(30);
       employee.setEmail("mahesh@test.com");
       return employee;
    }
    @Test
    public void testFindByName() {
       Employee employee = getEmployee();
       employeeRepository.save(employee);
       List<Employee> result = new ArrayList<>();
       employeeRepository.findByName(employee.getName()).forEach(e -> resul
       assertEquals(result.size(), 1);
    }
    @Test
    public void testFindByAge() {
       Employee employee = getEmployee();
       employeeRepository.save(employee);
       List<Employee> result = new ArrayList<>();
       employeeRepository.findByAge(employee.getAge()).forEach(e -> result.a
       assertEquals(result.size(), 1);
    }
    @Test
    public void testFindByEmail() {
       Employee employee = getEmployee();
       employeeRepository.save(employee);
       Employee result = employeeRepository.findByEmail(employee.getEmail()
       assertNotNull(result);
    }
}
```

## Run the test cases

Right Click on the file in eclipse and select **Run a JUnit Test** and verify the result.


JUnit
Finished after 10.646 seconds
Runs: 7/7
Errors: 0
Failures: 0
EmployeeRepositoryTest [Runner: JUnit 5] (1.115
Failure Trace
testFindAll() (0.505 s)
testSave() (0.024 s)
testFindByName() (0.068 s)
testFindById() (0.028 s)
testFindByEmail() (0.120 s)
testDeleteById() (0.342 s)
testFindByAge() (0.028 s)