Spring Data JPA – practical example

1. **Class diagram:**

Course

corseId : Long

name : String

credit : Integer

Student

studentId : Long

firstName : String

lastName : String

emailId : String

1..\*

students

1

1

1

1

courseMaterial course teacher guardian

1

Teacher

teacherId : Long

firstNmae: String

lastName :String

1

1

Guardian

name :String

email : String

mobile : String

CourseMaterial

courseMaterialId : Long

url : String

1. **Create Db schema schooldb in mysql**
2. **In package ‘entity’ create entity classes:**

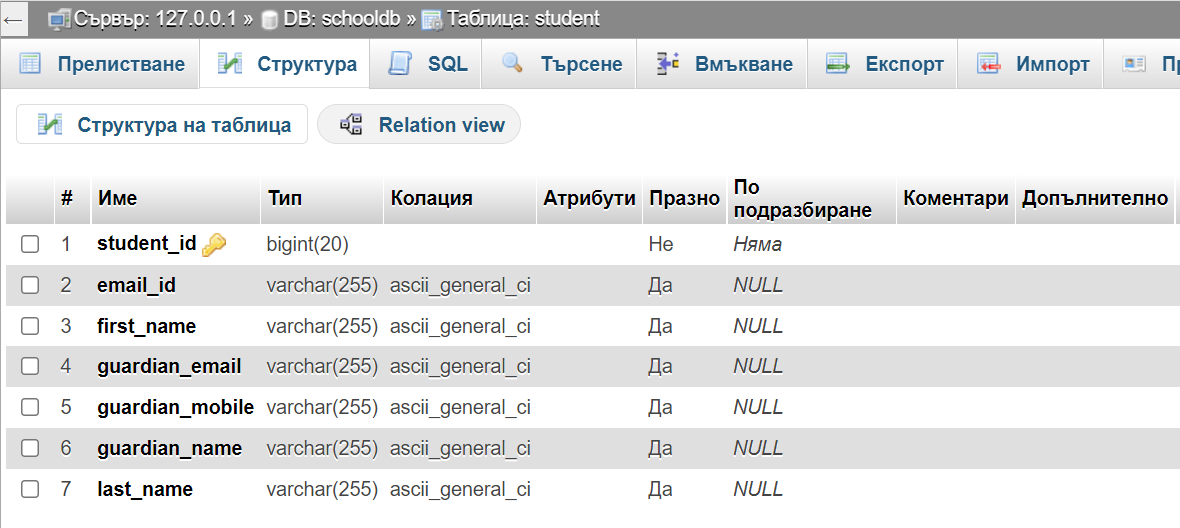
Student

package com.soar.spring.data.jpa.entity;  
  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import javax.persistence.\*;  
  
@Entity  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
@Builder  
public class Student {  
  
 private Long studentId;  
 private String firstName;  
 private String lastName;  
 private String emailId;  
 private String guardianName;  
 private String guardianEmail;  
 private String guardianMobile;  
}

* 1. **Run application and see creation of corresponding table ‘student’**

2021-07-06 12:07:26.622 INFO 18048 --- [ main] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.MySQL55Dialect

Hibernate: create table student (student\_id bigint not null, email\_id varchar(255), first\_name varchar(255), guardian\_email varchar(255), guardian\_mobile varchar(255), guardian\_name varchar(255), last\_name varchar(255), primary key (student\_id)) engine=InnoDB



* 1. Add some new code to Student class:

3.2.1

@Table(name="tbl\_student",  
 uniqueConstraints = @UniqueConstraint(  
 name = "emailid\_unique",  
 columnNames = "email\_address"  
 )  
)

3.2.2

@Id  
@SequenceGenerator(  
 name="student\_sequence",  
 sequenceName = "student\_sequence",  
 allocationSize=1  
)  
@GeneratedValue(  
 strategy = GenerationType.*SEQUENCE*,  
 generator="student\_sequence"  
)  
private Long studentId;

3.2.3

@Column(  
 name="email\_address",  
 nullable = false  
)  
private String emailId;

And the class Students looks like:

package com.soar.spring.data.jpa.entity;  
  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import javax.persistence.\*;  
  
@Entity  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
@Builder  
@Table(name="tbl\_student",  
 uniqueConstraints = @UniqueConstraint(  
 name = "emailid\_unique",  
 columnNames = "email\_address"  
 )  
)  
public class Student {  
  
 @Id  
 @SequenceGenerator(  
 name="student\_sequence",  
 sequenceName = "student\_sequence",  
 allocationSize=1  
 )  
 @GeneratedValue(  
 strategy = GenerationType.*SEQUENCE*,  
 generator="student\_sequence"  
 )  
 private Long studentId;  
 private String firstName;  
 private String lastName;  
  
 @Column(  
 name="email\_address",  
 nullable = false  
 )  
 private String emailId;   
}

After running the application, we have two new tables created into db:

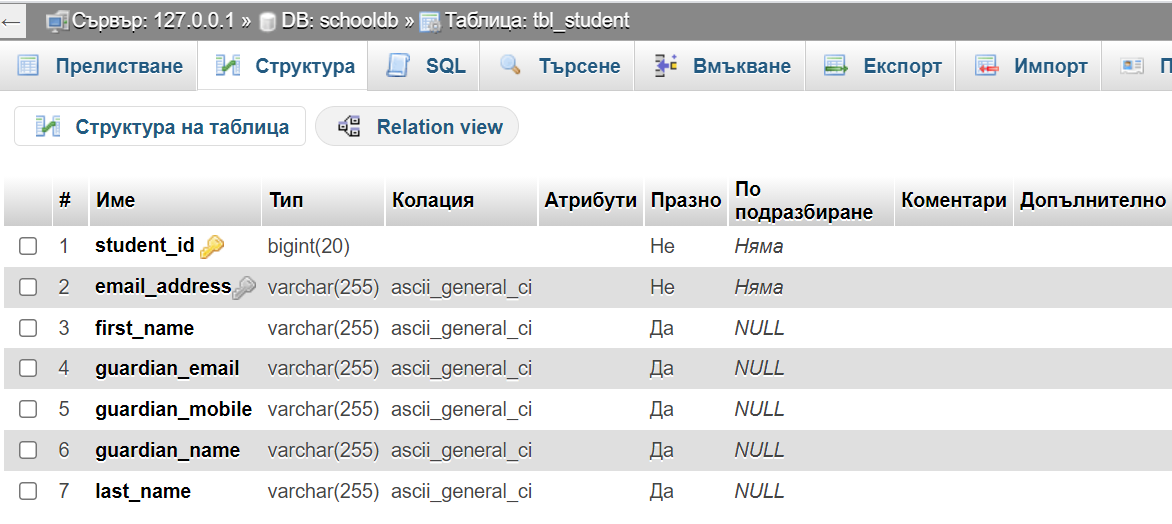
Hibernate: create **table student\_sequence** (next\_val bigint) engine=InnoDB

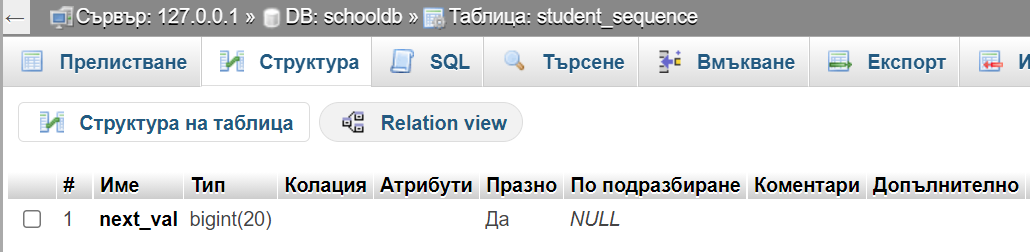
Hibernate: insert into student\_sequence values ( 1 )

Hibernate: create **table tbl\_student** (student\_id bigint not null, email\_address varchar(255) not null, first\_name varchar(255), guardian\_email varchar(255), guardian\_mobile varchar(255), guardian\_name varchar(255), last\_name varchar(255), primary key (student\_id)) engine=InnoDB

Hibernate: alter table tbl\_student drop index emailid\_unique

Hibernate: alter table tbl\_student add constraint emailid\_unique unique (email\_address)





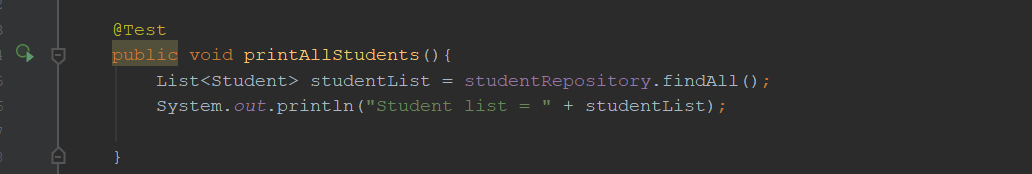
1. **Create repository interface for Student entity:**

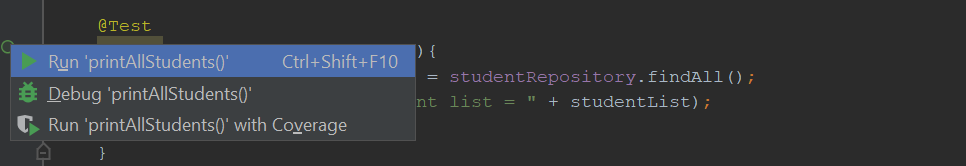
package com.soar.spring.data.jpa.repository;  
  
import com.soar.spring.data.jpa.entity.Student;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface StudentRepository extends JpaRepository<Student, Long> {  
  
}

* 1. Test repository:

package com.soar.spring.data.jpa;  
  
import com.soar.spring.data.jpa.entity.Guardian;  
import com.soar.spring.data.jpa.entity.Student;  
import com.soar.spring.data.jpa.repository.StudentRepository;  
import lombok.Builder;  
import org.junit.jupiter.api.Test;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;  
import org.springframework.boot.test.context.SpringBootTest;  
  
import java.util.List;  
  
  
@SpringBootTest  
public class StudentRepositoryTest {  
  
 @Autowired  
 private StudentRepository studentRepository;  
  
 @Test  
 public void saveStudent(){  
 Student student = Student.*builder*()  
 .emailId("soar@yahoo.com")  
 .firstName("Sonia")  
 .lastName("Arnaudova")  
 // .guardianName("ADN")  
 // .guardianEmail("adn@gmail.com")  
 // .guardianMobile("999999999")  
 .build();  
  
 studentRepository.save(student);  
 }  
  
 @Test  
 public void saveStudentWithGuardian(){  
 Guardian guardian = Guardian.*builder*()  
 .name("Peter Stone")  
 .email("ggg@yahoo.com")  
 .mobile("0998877665")  
 .build();  
  
 Student student = Student.*builder*()  
 .firstName("Ivan")  
 .lastName("Adam")  
 .emailId("adam@gmail.com")  
 .guardian(guardian)  
 .build();  
 studentRepository.save(student);  
  
 }  
  
 @Test  
 public void printAllStudents(){  
 List<Student> studentList = studentRepository.findAll();  
 System.*out*.println("Student list = " + studentList);  
  
 }  
}

Attention: Test each method separately! (click on green triangle to see the menu and then on ‘Run ….’)





* 1. Make a code to look presentable:

We don’t need guardian staff(

private String guardianName;  
private String guardianEmail;  
private String guardianMobile;

) inside Student class. So we create new class called Guardian, which is @Embeddable because we don’t want separate table:

package com.soar.spring.data.jpa.entity;  
  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import javax.persistence.\*;  
  
@Embeddable  
@Data  
@Builder  
@AllArgsConstructor  
@NoArgsConstructor  
@AttributeOverrides(  
 {  
 @AttributeOverride(  
 name = "name",  
 column = @Column(name = "guardian\_name")  
 ),  
 @AttributeOverride(  
 name = "email",  
 column = @Column(name = "guardian\_email")  
 ),  
 @AttributeOverride(  
 name = "mobile",  
 column = @Column(name = "guardian\_mobile")  
 )  
 }  
)  
public class Guardian {  
  
 private String name;  
 private String email;  
 private String mobile;  
  
}

Instead this in Student class :

private String guardianName;  
private String guardianEmail;  
private String guardianMobile;

we change it to:

@Embedded  
private Guardian guardian;

4.3 Test:

@Test  
public void saveStudentWithGuardian(){  
 Guardian guardian = Guardian.*builder*()  
 .name("Peter Stone")  
 .email("ggg@yahoo.com")  
 .mobile("0998877665")  
 .build();  
  
 Student student = Student.*builder*()  
 .firstName("Ivan")  
 .lastName("Adam")  
 .emailId("adam@gmail.com")  
 .guardian(guardian)  
 .build();  
 studentRepository.save(student);  
  
}

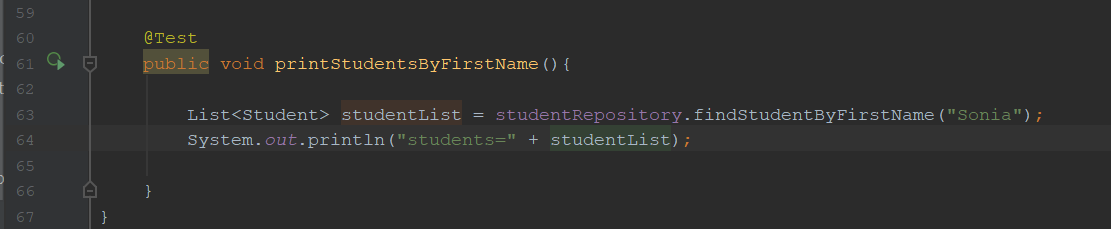
In table student we have new record:



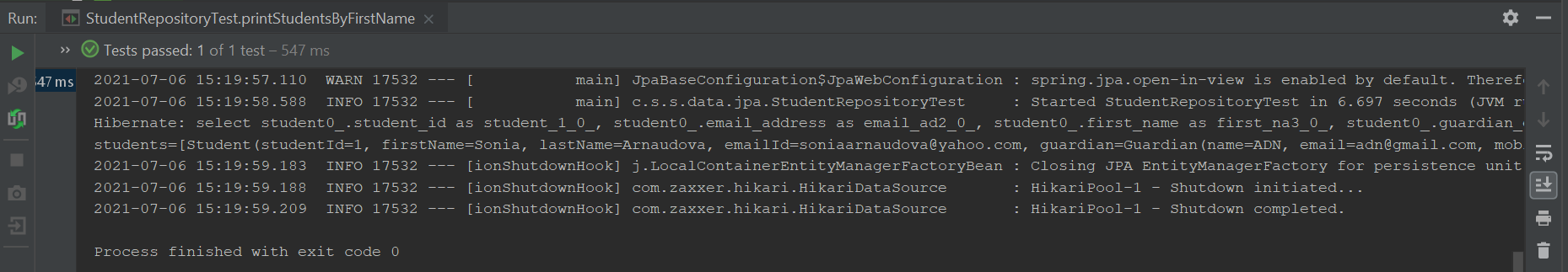
1. **Create custom methods in StudentRepository:**

package com.soar.spring.data.jpa.repository;  
  
import com.soar.spring.data.jpa.entity.Student;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
import java.util.List;  
  
@Repository  
public interface StudentRepository extends JpaRepository<Student, Long> {  
  
 public List<Student> findStudentByFirstName(String name);  
}

5.1 Test:



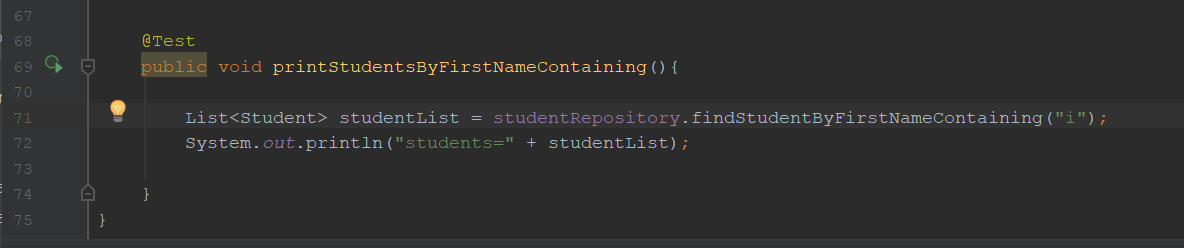
Result:



5.2 Create second method to find out students by first name containing “i”:

package com.soar.spring.data.jpa.repository;  
  
import com.soar.spring.data.jpa.entity.Student;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
import java.util.List;  
  
@Repository  
public interface StudentRepository extends JpaRepository<Student, Long> {  
  
 List<Student> findStudentByFirstName(String name);  
  
 List<Student> findStudentByFirstNameContaining(String name);  
}

Test it:



Result:

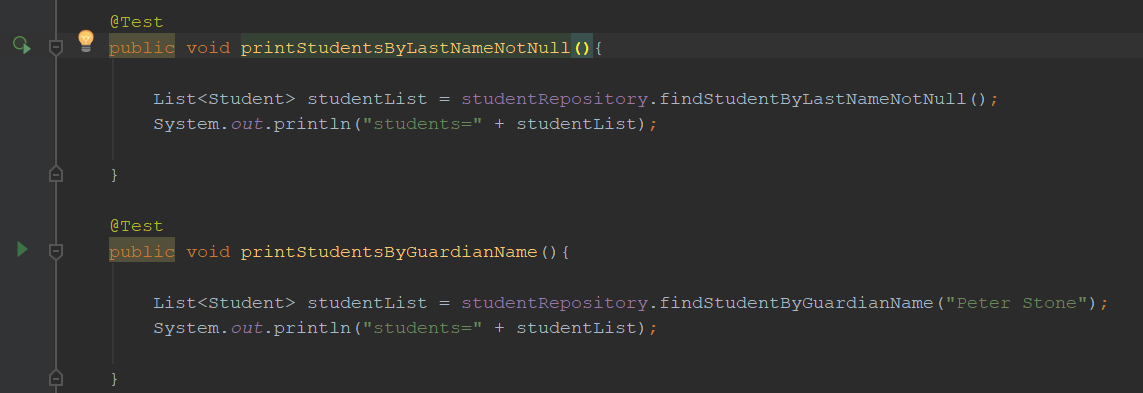
Hibernate: select student0\_.student\_id as student\_1\_0\_, student0\_.email\_address as email\_ad2\_0\_, student0\_.first\_name as first\_na3\_0\_, student0\_.guardian\_email as guardian4\_0\_, student0\_.guardian\_mobile as guardian5\_0\_, student0\_.guardian\_name as guardian6\_0\_, student0\_.last\_name as last\_nam7\_0\_ from student student0\_ where student0\_.first\_name like ? escape ?

students=[Student(studentId=1, firstName=Sonia, lastName=Arnaudova, emailId=soniaarnaudova@yahoo.com, guardian=Guardian(name=ADN, email=adn@gmail.com, mobile=999999999)), Student(studentId=3, firstName=Sonia, lastName=Arnaudova, emailId=soar@yahoo.com, guardian=Guardian(name=ADN, email=adn@gmail.com, mobile=999999999)), Student(studentId=6, firstName=Ivan, lastName=Adam, emailId=adam@gmail.com, guardian=Guardian(name=Peter Stone, email=ggg@yahoo.com, mobile=0998877665))]

5.3 Write two more methods:

package com.soar.spring.data.jpa.repository;  
  
import com.soar.spring.data.jpa.entity.Student;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
import java.util.List;  
  
@Repository  
public interface StudentRepository extends JpaRepository<Student, Long> {  
  
 List<Student> findStudentByFirstName(String name);  
  
 List<Student> findStudentByFirstNameContaining(String name);  
  
 List<Student> findStudentByLastNameNotNull();  
  
 List<Student> findStudentByGuardianName(String name);  
}

Test:



Result:

**findStudentByLastNameNotNull():**

Hibernate: select student0\_.student\_id as student\_1\_0\_, student0\_.email\_address as email\_ad2\_0\_, student0\_.first\_name as first\_na3\_0\_, student0\_.guardian\_email as guardian4\_0\_, student0\_.guardian\_mobile as guardian5\_0\_, student0\_.guardian\_name as guardian6\_0\_, student0\_.last\_name as last\_nam7\_0\_ from student student0\_ where student0\_.last\_name is not null

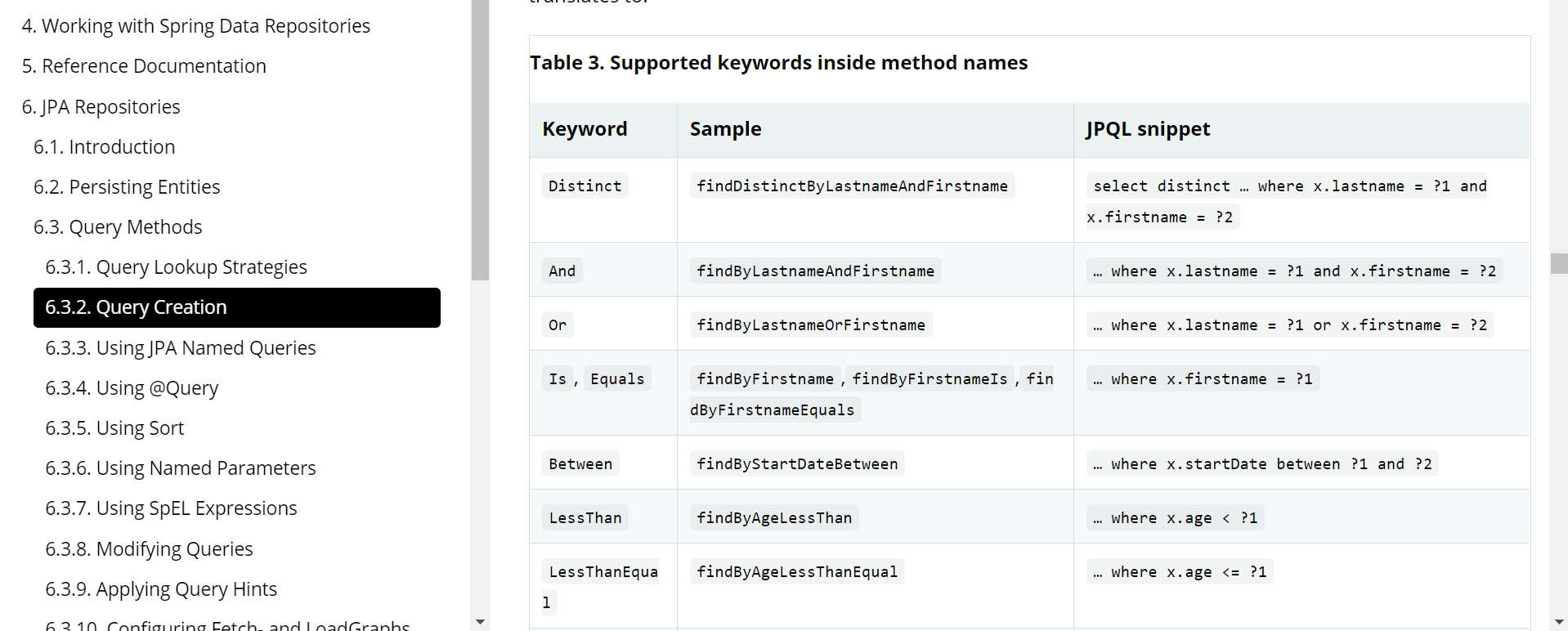
students=[Student(studentId=1, firstName=Sonia, lastName=Arnaudova, emailId=soniaarnaudova@yahoo.com, guardian=Guardian(name=ADN, email=adn@gmail.com, mobile=999999999)), Student(studentId=3, firstName=Sonia, lastName=Arnaudova, emailId=soar@yahoo.com, guardian=Guardian(name=ADN, email=adn@gmail.com, mobile=999999999)), Student(studentId=6, firstName=Ivan, lastName=Adam, emailId=adam@gmail.com, guardian=Guardian(name=Peter Stone, email=ggg@yahoo.com, mobile=0998877665))]

**findStudentByGuardianName(“Peter Stone”):**

Hibernate: select student0\_.student\_id as student\_1\_0\_, student0\_.email\_address as email\_ad2\_0\_, student0\_.first\_name as first\_na3\_0\_, student0\_.guardian\_email as guardian4\_0\_, student0\_.guardian\_mobile as guardian5\_0\_, student0\_.guardian\_name as guardian6\_0\_, student0\_.last\_name as last\_nam7\_0\_ from student student0\_ where student0\_.guardian\_name=?

students=[Student(studentId=6, firstName=Ivan, lastName=Adam, emailId=adam@gmail.com, guardian=Guardian(name=Peter Stone, email=ggg@yahoo.com, mobile=0998877665))]

For more info [Spring Data JPA - Reference Documentation](https://docs.spring.io/spring-data/jpa/docs/2.5.2/reference/html/#jpa.query-methods) :

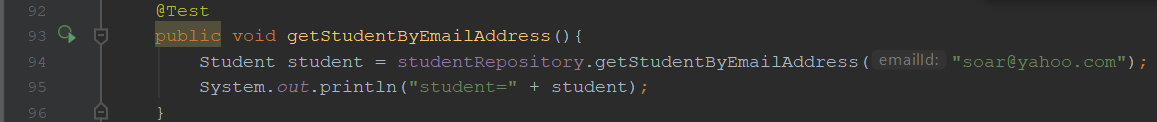


5.4 More examples:

//JPQL  
@Query("select s from Student s where s.emailId = ?1")  
Student getStudentByEmailAddress(String emailId);  
  
//JPQL  
@Query("select s.firstName from Student s where s.emailId = ?1")  
Student getStudentFirstNameByEmailAddress(String emailId);  
  
//Native query  
@Query(  
 value="select \* from student s where s.email\_address = ?1",  
 nativeQuery = true  
)  
Student getStudentNativeByEmailAddress(String emailId);

Test:

5.4.1

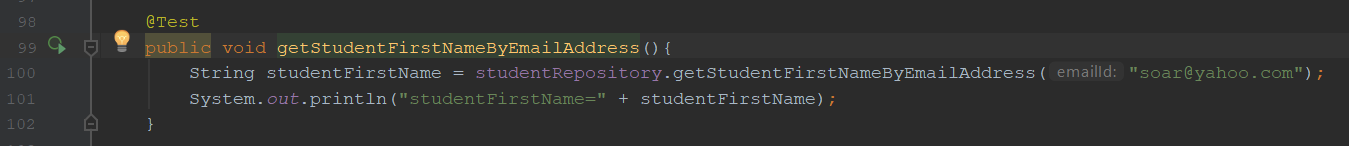


Result:

Hibernate: select student0\_.student\_id as student\_1\_0\_, student0\_.email\_address as email\_ad2\_0\_, student0\_.first\_name as first\_na3\_0\_, student0\_.guardian\_email as guardian4\_0\_, student0\_.guardian\_mobile as guardian5\_0\_, student0\_.guardian\_name as guardian6\_0\_, student0\_.last\_name as last\_nam7\_0\_ from student student0\_ where student0\_.email\_address=?

student=Student(studentId=3, firstName=Sonia, lastName=Arnaudova, emailId=soar@yahoo.com, guardian=Guardian(name=ADN, email=adn@gmail.com, mobile=999999999)

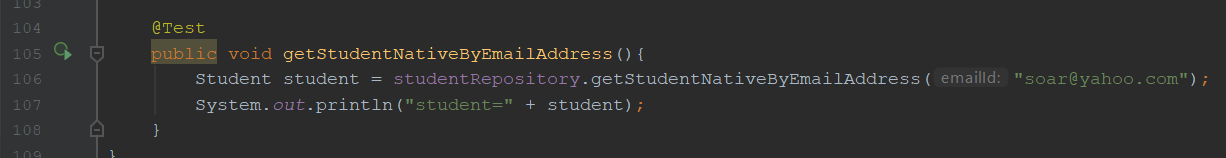
5.4.2



Hibernate: select student0\_.first\_name as col\_0\_0\_ from student student0\_ where student0\_.email\_address=?

studentFirstName=Sonia

5.4.3



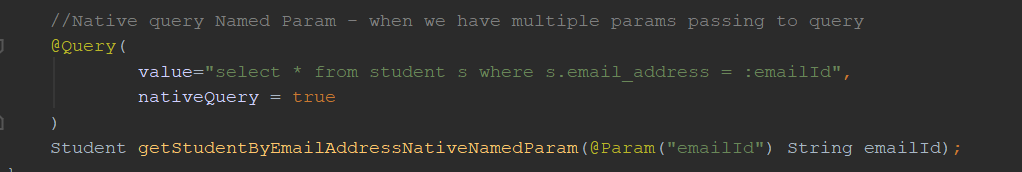
Result:

Hibernate: select \* from student s where s.email\_address = ?

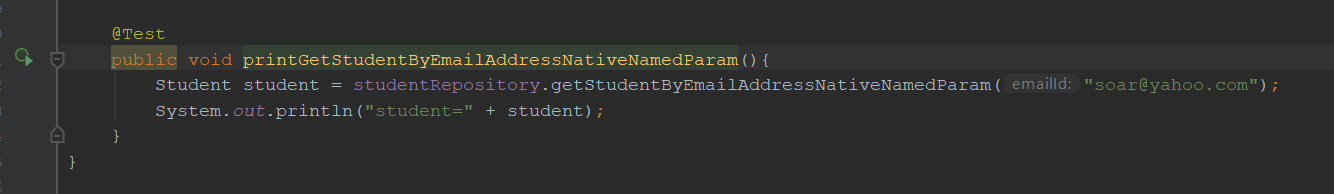
student=Student(studentId=3, firstName=Sonia, lastName=Arnaudova, emailId=soar@yahoo.com, guardian=Guardian(name=ADN, email=adn@gmail.com, mobile=999999999))

5.5 Query Named Param:

In StudentRepository interface:



Test:



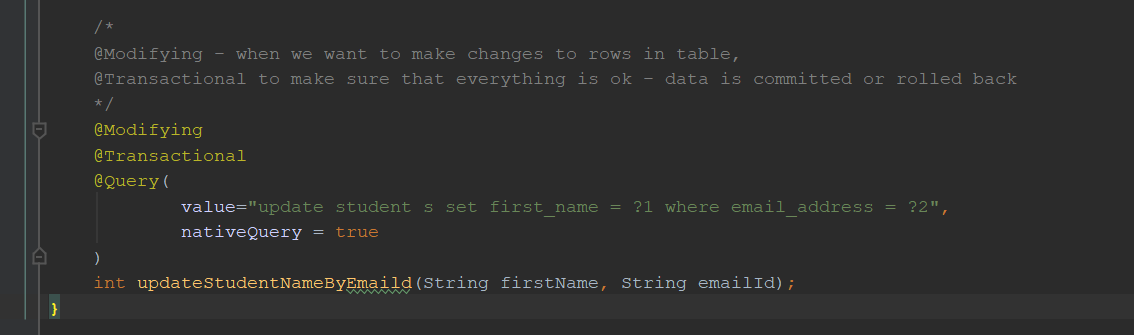
Result:

Hibernate: select \* from student s where s.email\_address = ?

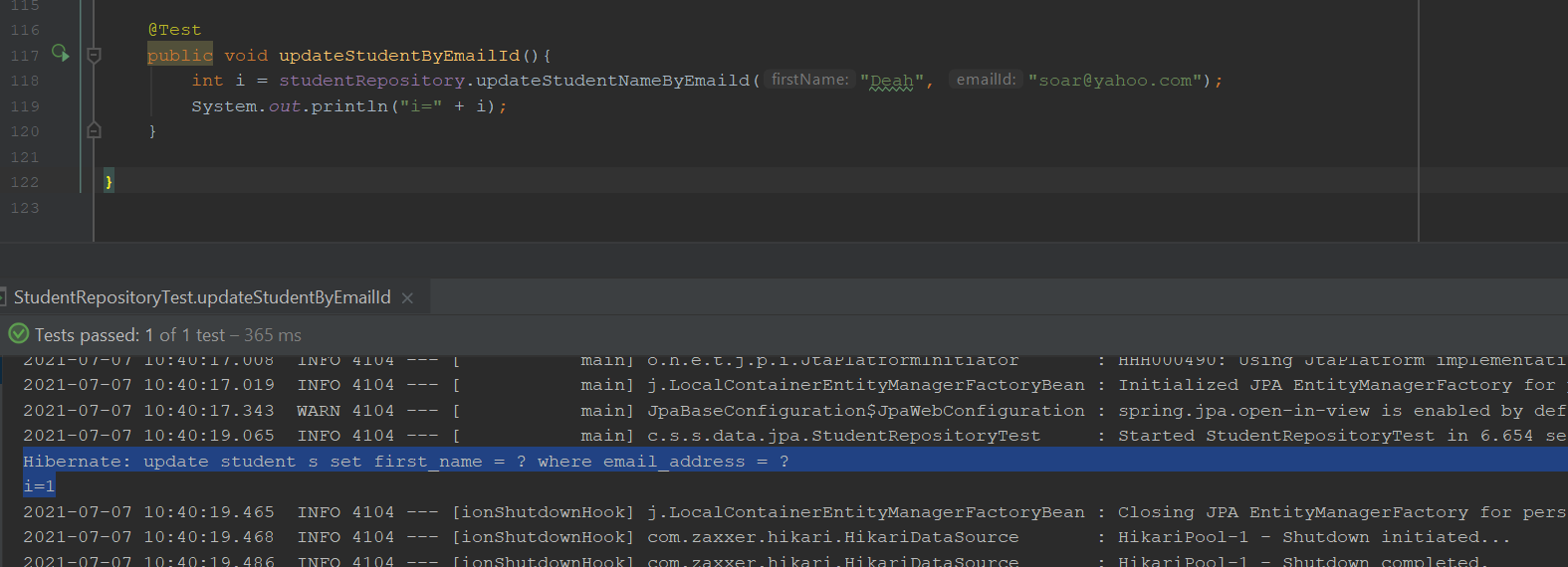
student=Student(studentId=3, firstName=Sonia, lastName=Arnaudova, emailId=soar@yahoo.com, guardian=Guardian(name=ADN, email=adn@gmail.com, mobile=999999999))

* 1. @Transactional and @Modifying Annotation:

In StudentRepository add method:



Test and Result:



6.@OneToOne annotation:

* 1. Create two classes – Course and CourseMaterial. Here we have 1 to 1 relationship:

package com.soar.spring.data.jpa.entity;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import javax.persistence.\*;  
  
@Entity  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
public class Course {  
  
 @Id  
 @SequenceGenerator(  
 name = "course\_sequence",  
 sequenceName = "course\_sequence",  
 allocationSize = 1  
 )  
 @GeneratedValue(  
 strategy = GenerationType.*SEQUENCE*,  
 generator = "course\_sequence"  
 )  
 private Long courseId;  
 private String title;  
 private Integer credit;  
  
}

and

package com.soar.spring.data.jpa.entity;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import javax.persistence.\*;  
  
@Entity  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
public class CourseMaterial {  
  
 @Id  
 @SequenceGenerator(  
 name="corse\_material\_sequence",  
 sequenceName = "corse\_material\_sequence",  
 allocationSize = 1  
 )  
 @GeneratedValue(  
 strategy = GenerationType.*SEQUENCE*,  
 generator = "corse\_material\_sequence"  
 )  
 private Long courseMaterialId;  
 private String url;  
  
 @OneToOne  
 @JoinColumn(  
 name = "course\_Id",  
 referencedColumnName = "courseId"  
  
 )  
 private Course course;  
}

CourseMaterial can’t be without course, so in this class where add @OneToOne relationship.

After Runnig application:

2021-07-07 11:15:51.116 INFO 19064 --- [ main] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.MySQL55Dialect

Hibernate: create table corse\_material\_sequence (next\_val bigint) engine=InnoDB

Hibernate: insert into corse\_material\_sequence values ( 1 )

Hibernate: create table course (course\_id bigint not null, credit integer, title varchar(255), primary key (course\_id)) engine=InnoDB

Hibernate: create table course\_sequence (next\_val bigint) engine=InnoDB

Hibernate: insert into course\_sequence values ( 1 )

Hibernate: create table course\_material (course\_material\_id bigint not null, url varchar(255), course\_id bigint, primary key (course\_material\_id)) engine=InnoDB

Hibernate: alter table course\_material add constraint FK6qgylrot7cxgungunwyrslpeo foreign key (course\_id) references course (course\_id)

6.2 Create CourseRepository interface:

package com.soar.spring.data.jpa.repository;  
  
import com.soar.spring.data.jpa.entity.Course;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface CourseRepository extends JpaRepository<Course, Long> {  
}

,

and CourseMaterialRepository interface:

package com.soar.spring.data.jpa.repository;  
  
import com.soar.spring.data.jpa.entity.CourseMaterial;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface CourseMaterialRepository extends JpaRepository<CourseMaterial, Long> {  
}

6.3 Create CourseMaterialRepositoryTest class:

package com.soar.spring.data.jpa;  
  
  
import com.soar.spring.data.jpa.entity.Course;  
import com.soar.spring.data.jpa.entity.CourseMaterial;  
import com.soar.spring.data.jpa.repository.CourseMaterialRepository;  
import org.junit.jupiter.api.Test;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
  
@SpringBootTest  
public class CourseMaterialRepositoryTest {  
  
 @Autowired  
 private CourseMaterialRepository courseMaterialRepository;  
  
 @Test  
 public void saveCourseMaterial(){  
 Course course =Course.*builder*()  
 .title("COS-1")  
 .credit(6)  
 .build();  
  
 CourseMaterial courseMaterial = CourseMaterial.*builder*()  
 .url("www.google.com")  
 .course(course)  
 .build();  
 courseMaterialRepository.save(courseMaterial);  
  
 }  
}

When we run a test an error appears:

Hibernate: select next\_val as id\_val from corse\_material\_sequence for update

Hibernate: update corse\_material\_sequence set next\_val= ? where next\_val=?

org.springframework.dao.InvalidDataAccessApiUsageException: org.hibernate.TransientPropertyValueException: object references an unsaved transient instance - save the transient instance before flushing : com.soar.spring.data.jpa.entity.CourseMaterial.course -> com.soar.spring.data.jpa.entity.Course; nested exception is java.lang.IllegalStateException: org.hibernate.TransientPropertyValueException: object references an unsaved transient instance - save the transient instance before flushing : com.soar.spring.data.jpa.entity.CourseMaterial.course -> com.soar.spring.data.jpa.entity.Course

Because in table course there isn’t any course corresponding to CourseMaterial. In this case we have to use cascading mechanism – any changes affect all participants. This establishes dependency between related entitie and appears inside @OneToOne, @OneToMany, @ManyToOne, @ManyToMany relationships.

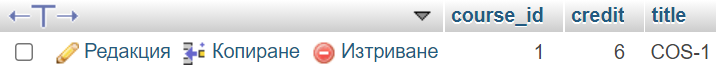
cascade = CascadeType.ALL - all the above cascade operations can be applied to the entities related to parent entities.

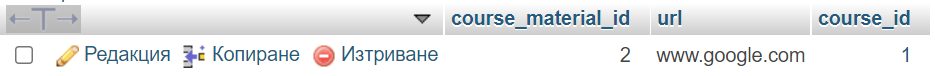
For more info: https://www.javatpoint.com/jpa-cascading-operations

In CourseMaterial we add cascade to @OneToOne annotation”

@OneToOne(  
 cascade = CascadeType.*ALL*)  
@JoinColumn(  
 name = "course\_Id",  
 referencedColumnName = "courseId"  
  
)  
private Course course;

After running the test on method saveCorseMaterial() in CourseMaterialRepositoryTest class, everything is OK. New records are created into course and corse\_material tables:





* 1. Fetch Type – when we want to bring an info about all dependencies, we use EAGER fetching type, but only for courseMaterial data + courceId, and later for course – LASY.

Write test method:

@Test  
public void printAllCourseMaterials(){  
 List<CourseMaterial> courseMaterials =  
 courseMaterialRepository.findAll();  
 System.*out*.println("courseMaterials=" + courseMaterials);  
}

* + 1. EAGER:

@OneToOne(  
 cascade = CascadeType.*ALL*,  
 fetch = FetchType.*EAGER*)

Result of test:

Hibernate: select coursemate0\_.course\_material\_id as course\_m1\_1\_, coursemate0\_.course\_id as course\_i3\_1\_, coursemate0\_.url as url2\_1\_ from course\_material coursemate0\_

Hibernate: select course0\_.course\_id as course\_i1\_0\_0\_, course0\_.credit as credit2\_0\_0\_, course0\_.title as title3\_0\_0\_ from course course0\_ where course0\_.course\_id=?

Hibernate: select course0\_.course\_id as course\_i1\_0\_0\_, course0\_.credit as credit2\_0\_0\_, course0\_.title as title3\_0\_0\_ from course course0\_ where course0\_.course\_id=?

courseMaterials=[CourseMaterial(courseMaterialId=2, url=www.google.com, course=Course(courseId=1, title=COS-1, credit=6)), CourseMaterial(courseMaterialId=3, url=www.yahoo.com, course=Course(courseId=2, title=COS-2, credit=6))]

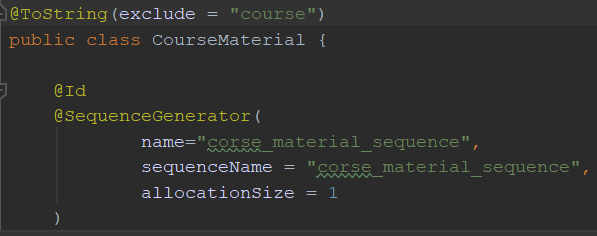
6.4.2

@OneToOne(  
 cascade = CascadeType.*ALL*,  
 fetch = FetchType.*LAZY*)

Result:

Error - org.hibernate.LazyInitializationException: could not initialize proxy [com.soar.spring.data.jpa.entity.Course#1] - no Session

To avoid this, add annotation @ToString(exclude = “course”) / @ToString.Exclude above class CorseMaterial



Result:

Hibernate: select coursemate0\_.course\_material\_id as course\_m1\_1\_, coursemate0\_.course\_id as course\_i3\_1\_, coursemate0\_.url as url2\_1\_ from course\_material coursemate0\_

courseMaterials=[CourseMaterial(courseMaterialId=2, url=www.google.com), CourseMaterial(courseMaterialId=3, url=www.yahoo.com)]

We see that course is excluded.

6.5 Bi directional relationship:

Create test class for CourseRepository and add method:

package com.soar.spring.data.jpa;  
  
import com.soar.spring.data.jpa.entity.Course;  
import com.soar.spring.data.jpa.repository.CourseRepository;  
import org.junit.jupiter.api.Test;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
  
import java.util.List;  
  
@SpringBootTest  
public class CourseRepositoryTest {  
  
 @Autowired  
 private CourseRepository courseRepository;  
  
 @Test  
 public void printAllCources(){  
 List<Course> courses = courseRepository.findAll();  
 System.*out*.println("courses=" + courses);  
 }  
}

Run the test:

Hibernate: select course0\_.course\_id as course\_i1\_0\_, course0\_.credit as credit2\_0\_, course0\_.title as title3\_0\_ from course course0\_

courses=[Course(courseId=1, title=COS-1, credit=6), Course(courseId=2, title=COS-2, credit=6)]

Till now we have only data from courses not from courseMaterial.

In CourseMaterial class we have already this relationship:

@OneToOne(  
 cascade = CascadeType.*ALL*,  
 fetch = FetchType.*EAGER*,  
 optional = false  
)  
@JoinColumn(  
 name = "course\_Id",  
 referencedColumnName = "courseId"  
)  
private Course course;

Hibernate: select course0\_.course\_id as course\_i1\_0\_, course0\_.credit as credit2\_0\_, course0\_.title as title3\_0\_ from course course0\_

Hibernate: select coursemate0\_.course\_material\_id as course\_m1\_1\_0\_, coursemate0\_.course\_id as course\_i3\_1\_0\_, coursemate0\_.url as url2\_1\_0\_ from course\_material coursemate0\_ where coursemate0\_.course\_id=?

Hibernate: select coursemate0\_.course\_material\_id as course\_m1\_1\_0\_, coursemate0\_.course\_id as course\_i3\_1\_0\_, coursemate0\_.url as url2\_1\_0\_ from course\_material coursemate0\_ where coursemate0\_.course\_id=?

courses=[Course(courseId=1, title=COS-1, credit=6, courseMaterial=CourseMaterial(courseMaterialId=2, url=www.google.com)), Course(courseId=2, title=COS-2, credit=6, courseMaterial=CourseMaterial(courseMaterialId=3, url=www.yahoo.com))]

We received data about each course with corresponding courseMaterial’s data !

But, of course we can omit @OneToOne annotation in Course entity class. The result after running the test will be:

courseMaterials=[CourseMaterial(courseMaterialId=2, url=www.google.com), CourseMaterial(courseMaterialId=3, url=www.yahoo.com), CourseMaterial(courseMaterialId=4, url=www.yahoo.com)]

6.6 @OneToMany:

6.6.1 Create class Teacher:

package com.soar.spring.data.jpa.entity;  
  
  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import javax.persistence.\*;  
import java.util.List;  
  
@Entity  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class Teacher {  
  
 @Id  
 @SequenceGenerator(  
 name = "teacher\_sequence",  
 sequenceName = "teacher\_sequence",  
 allocationSize = 1  
 )  
 @GeneratedValue(  
 strategy = GenerationType.*SEQUENCE*,  
 generator = "teacher\_sequence"  
 )  
 private Long teacherId;  
 private String firstName;  
 private String lastName;  
  
@OneToMany(mappedBy = "teacher",  
 cascade = CascadeType.*ALL*,  
 fetch = FetchType.*EAGER*)  
@ToString.Exclude  
private List<Course> courses;

}

One teacher can teach many courses.

When runing the project table teacher will be created:

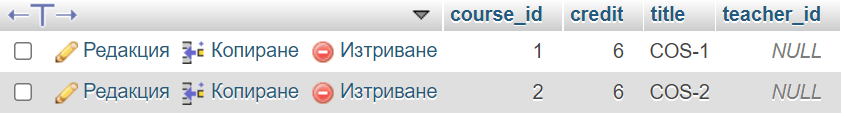
Hibernate: alter table course add column teacher\_id bigint

Hibernate: create table teacher (teacher\_id bigint not null, first\_name varchar(255), last\_name varchar(255), primary key (teacher\_id)) engine=InnoDB

Hibernate: create table teacher\_sequence (next\_val bigint) engine=InnoDB

Hibernate: insert into teacher\_sequence values ( 1 )

Hibernate: alter table course add constraint FKsybhlxoejr4j3teomm5u2bx1n foreign key (teacher\_id) references teacher (teacher\_id)



6.6.2 Create TeacherRepository interface:

package com.soar.spring.data.jpa.repository;  
  
import com.soar.spring.data.jpa.entity.Teacher;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface TeacherRepository extends JpaRepository<Teacher,Long> {  
}

package com.soar.spring.data.jpa;  
  
import com.soar.spring.data.jpa.entity.Course;  
import com.soar.spring.data.jpa.entity.Teacher;  
import com.soar.spring.data.jpa.repository.TeacherRepository;  
import org.junit.jupiter.api.Test;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
  
import java.util.Arrays;  
  
@SpringBootTest  
public class TeacherRepositoryTest {  
  
 @Autowired  
 private TeacherRepository teacherRepository;  
  
 @Test  
 public void saveTeacher(){  
 Course course = Course.*builder*()  
 .title("COS-3")  
 .credit(5)  
 .build();  
 Course course2 = Course.*builder*()  
 .title("COS-4")  
 .credit(3)  
 .build();  
  
 Teacher teacher = Teacher.*builder*()  
 .firstName("Adam")  
 .lastName("Smook")  
 .courses(Arrays.*asList*(course,course2))  
 .build();  
  
 teacherRepository.save(teacher);  
 }  
}

Run a method of test class:

Hibernate: select next\_val as id\_val from teacher\_sequence for update

Hibernate: update teacher\_sequence set next\_val= ? where next\_val=?

Hibernate: select next\_val as id\_val from course\_sequence for update

Hibernate: update course\_sequence set next\_val= ? where next\_val=?

Hibernate: select next\_val as id\_val from course\_sequence for update

Hibernate: update course\_sequence set next\_val= ? where next\_val=?

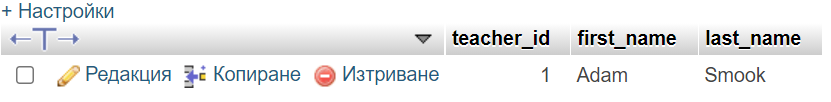
Hibernate: insert into teacher (first\_name, last\_name, teacher\_id) values (?, ?, ?)

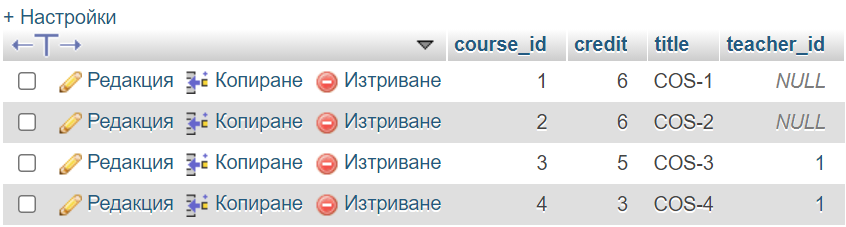
Hibernate: insert into course (credit, title, course\_id) values (?, ?, ?)

Hibernate: insert into course (credit, title, course\_id) values (?, ?, ?)

Hibernate: update course set teacher\_id=? where course\_id=?

Hibernate: update course set teacher\_id=? where course\_id=?





In this case we have two new courses without course materials because by default optional part in @OneToOne in CourseMaterial class is true

@OneToOne(  
 cascade = CascadeType.*ALL*,  
 fetch = FetchType.*LAZY*  
)  
@JoinColumn(  
 name = "course\_Id",  
 referencedColumnName = "courseId"  
)  
private Course course;

We have to change it to: optional = false

@OneToOne(  
 cascade = CascadeType.*ALL*,  
 fetch = FetchType.*LAZY*,  
 optional = false  
)  
@JoinColumn(  
 name = "course\_Id",  
 referencedColumnName = "courseId"  
)  
private Course course;

This forces us to add course when saving coursematerial.

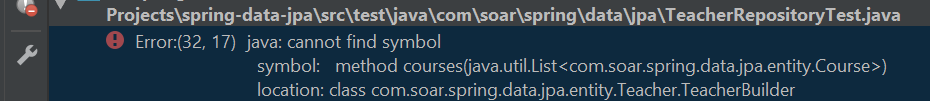
6.7 @ManyToOne relationtionships:

Add this to Course class:

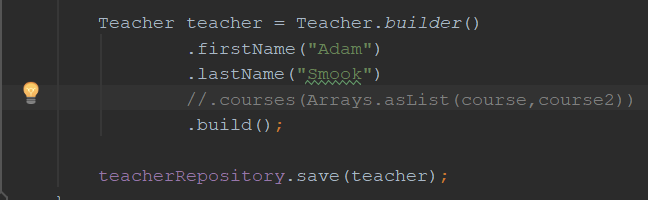
@ManyToOne(  
 cascade = CascadeType.*ALL*,  
 fetch = FetchType.*EAGER*)  
@JoinColumn(  
 name = "teacher\_id",  
 referencedColumnName = "teacherId"  
)  
@ToString.Exclude  
private Teacher teacher;

6.7.1 In CourseRepositoryTest class:

@Test  
public void saveCourseWithTeacher(){  
 Teacher teacher = Teacher.*builder*()  
 .firstName("Simon")  
 .lastName("Walker")  
 .build();  
  
 Course course = Course.*builder*()  
 .title("COS-5")  
 .credit(6)  
 .teacher(teacher)  
 .build();  
  
 courseRepository.save(course);  
}



Comment courses() in TeacherRepositoryTest



Result: New course and teacher are inserted into db.

2021-07-11 17:04:15.517 INFO 19720 --- [ main] c.s.s.data.jpa.CourseRepositoryTest : Started CourseRepositoryTest in 7.163 seconds (JVM running for 8.969)

Hibernate: select next\_val as id\_val from course\_sequence for update

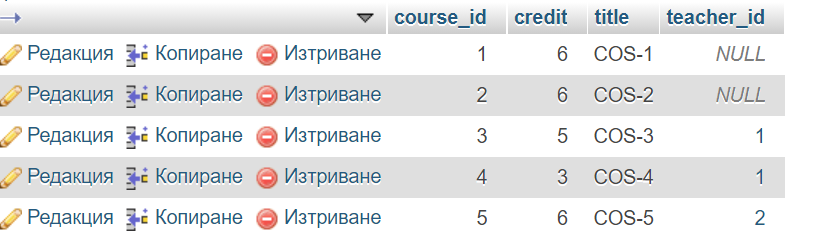
Hibernate: update course\_sequence set next\_val= ? where next\_val=?

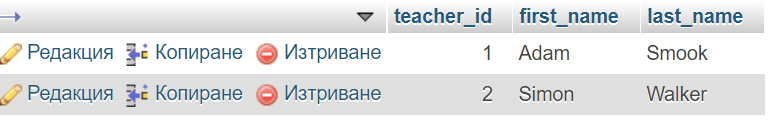
Hibernate: select next\_val as id\_val from teacher\_sequence for update

Hibernate: update teacher\_sequence set next\_val= ? where next\_val=?

Hibernate: insert into teacher (first\_name, last\_name, teacher\_id) values (?, ?, ?)

Hibernate: insert into course (credit, teacher\_id, title, course\_id) values (?, ?, ?, ?)





6.8 Paging and Sorting

All our repository interfaces extend JpaRepository interface which extends PagingAndSortingRepository<T, ID> interface.

package org.springframework.data.repository;  
  
import org.springframework.data.domain.Page;  
import org.springframework.data.domain.Pageable;  
import org.springframework.data.domain.Sort;  
  
@NoRepositoryBean  
public interface PagingAndSortingRepository<T, ID> extends CrudRepository<T, ID> {  
 Iterable<T> findAll(Sort var1);  
  
 Page<T> findAll(Pageable var1);  
}

In CourseRepositoryTest:

@Test  
public void findAllPagination(){  
 Pageable firstPageWithTreeRecords =  
 PageRequest.*of*(0,3);  
 Pageable secondPageWithTreeRecords =  
 PageRequest.*of*(1,2);  
 List<Course> courses = courseRepository.findAll(firstPageWithTreeRecords)  
 .getContent();  
 Long totalElements =   
 courseRepository.findAll(firstPageWithTreeRecords)  
 .getTotalElements();  
 int totalPages =   
 courseRepository.findAll(firstPageWithTreeRecords)  
 .getTotalPages();  
  
 System.*out*.println("courses = " + courses);  
  
 System.*out*.println("totalElements = " +totalElements);  
   
 System.*out*.println("totalPages = " + totalPages);  
}

Result:

courses = [Course(courseId=1, title=COS-1, credit=6, courseMaterial=CourseMaterial(courseMaterialId=2, url=www.google.com), teacher=null), Course(courseId=2, title=COS-2, credit=6, courseMaterial=CourseMaterial(courseMaterialId=3, url=www.yahoo.com), teacher=null), Course(courseId=3, title=COS-3, credit=5, courseMaterial=null, teacher=Teacher(teacherId=1, firstName=Adam, lastName=Smook))]

totalElements = 5

totalPages = 2

6.8.1 Sorting:

@Test  
public void findAllSorting(){  
 Pageable sortByTitle =  
 PageRequest.*of*(0,2, Sort.*by*("title"));  
 Pageable sortByCreditDesc =  
 PageRequest.*of*(0,2,Sort.*by*("credit").descending());  
  
 Pageable sortByTytleAndCreditDesc =  
 PageRequest.*of*(0,2,  
 Sort.*by*("title").descending()  
 .and(Sort.*by*("credit")));  
  
 List<Course> courses = courseRepository.findAll(sortByTitle).getContent();  
 System.*out*.println("courses sortByTitle = " +courses);  
  
 List<Course> courses2 = courseRepository.findAll(sortByCreditDesc).getContent();  
 System.*out*.println("courses sortedByCreditsDesc = " + courses2);  
  
 List<Course> courses3 = courseRepository.findAll(sortByTytleAndCreditDesc).getContent();  
 System.*out*.println("courses sortByTytleAndCreditDesc = " + courses3);  
  
}

Result:

courses sortByTitle = [Course(courseId=1, title=COS-1, credit=6, courseMaterial=CourseMaterial(courseMaterialId=2, url=www.google.com), teacher=null), Course(courseId=2, title=COS-2, credit=6, courseMaterial=CourseMaterial(courseMaterialId=3, url=www.yahoo.com), teacher=null)]

courses sortedByCreditsDesc = [Course(courseId=1, title=COS-1, credit=6, courseMaterial=CourseMaterial(courseMaterialId=2, url=www.google.com), teacher=null), Course(courseId=2, title=COS-2, credit=6, courseMaterial=CourseMaterial(courseMaterialId=3, url=www.yahoo.com), teacher=null)]

courses sortByTytleAndCreditDesc = [Course(courseId=5, title=COS-5, credit=6, courseMaterial=null, teacher=Teacher(teacherId=2, firstName=Simon, lastName=Walker)), Course(courseId=4, title=COS-4, credit=3, courseMaterial=null, teacher=Teacher(teacherId=1, firstName=Adam, lastName=Smook))]

6.8.2

Add new method to CourseRepository interface:

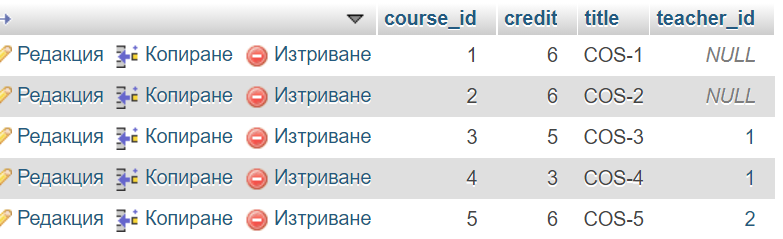
package com.soar.spring.data.jpa.repository;  
  
import com.soar.spring.data.jpa.entity.Course;  
import org.springframework.data.domain.Page;  
import org.springframework.data.domain.PageRequest;  
import org.springframework.data.domain.Pageable;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface CourseRepository extends JpaRepository<Course, Long> {  
  
 Page<Course> findByTitleContaining(String title, Pageable pageable);  
}

Test it into CourseRepositoryTest class:

@Test  
public void printFindByTitleContaining(){  
 Pageable firstPageTenRecords =  
 PageRequest.*of*(0,10);  
 List<Course> courses =  
 courseRepository.findByTitleContaining("C", firstPageTenRecords).getContent();  
  
 System.*out*.println("courses = " +courses);  
}

Result:

courses = [Course(courseId=1, title=COS-1, credit=6, courseMaterial=CourseMaterial(courseMaterialId=2, url=www.google.com), teacher=null), Course(courseId=2, title=COS-2, credit=6, courseMaterial=CourseMaterial(courseMaterialId=3, url=www.yahoo.com), teacher=null), Course(courseId=3, title=COS-3, credit=5, courseMaterial=null, teacher=Teacher(teacherId=1, firstName=Adam, lastName=Smook)), Course(courseId=4, title=COS-4, credit=3, courseMaterial=null, teacher=Teacher(teacherId=1, firstName=Adam, lastName=Smook)), Course(courseId=5, title=COS-5, credit=6, courseMaterial=null, teacher=Teacher(teacherId=2, firstName=Simon, lastName=Walker))]



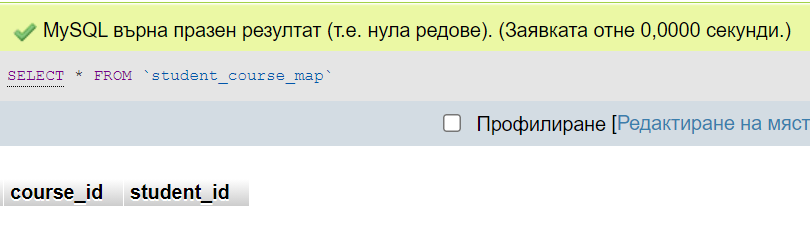
6.9 @ManyToMany relationships between Student and Course classes:

In Course class we define this relationships:

@ManyToMany(  
 cascade = CascadeType.*ALL,*

fetch = FetchType.*EAGER*)  
@JoinTable(  
 name = "student\_course\_map",  
 joinColumns = @JoinColumn(  
 name = "course\_id",  
 referencedColumnName = "courseId"  
 ),  
 inverseJoinColumns = @JoinColumn(  
 name = "student\_id",  
 referencedColumnName = "studentId"  
 )  
)  
private List<Student> students;  
  
public void addStudent(Student student){  
 if(students == null) {  
 students = new ArrayList<>();  
 }  
 students.add(student);  
}

New table is created after running the project:



Add this code to Student class:

@ManyToMany(mappedBy = "students",  
fetch = FetchType.*EAGER*)  
private List<Course> courses;

6.9.1

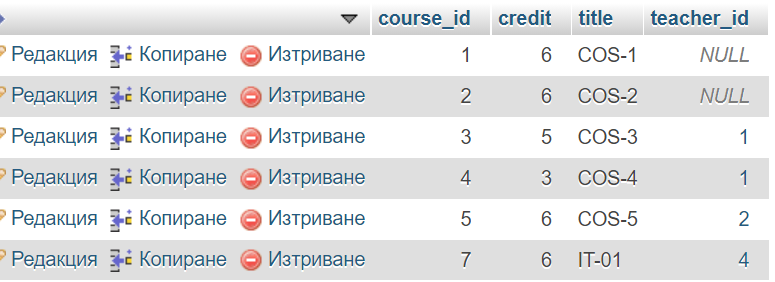
In CourseRepositoryTest class add new method:

@Test  
public void saveCourseWithStudentAndTeacher(){  
 Teacher teacher = Teacher.*builder*()  
 .firstName("Ivan")  
 .lastName("White")  
 .build();  
 Student student = Student.*builder*()  
 .firstName("Sonia")  
 .lastName("Arnaudova")  
 .emailId("soar@yahoo.com")  
 .build();  
 Course course = Course.*builder*()  
 .title("IT-01")  
 .credit(6)  
 .teacher(teacher)  
 .build();  
 course.addStudent(student);  
 courseRepository.save(course);  
}

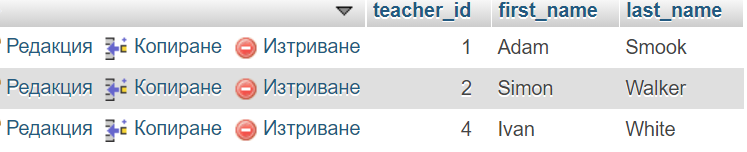
Hibernate: insert into course (credit, teacher\_id, title, course\_id) values (?, ?, ?, ?)

Hibernate: insert into student (email\_address, first\_name, guardian\_email, guardian\_mobile, guardian\_name, last\_name, student\_id) values (?, ?, ?, ?, ?, ?, ?)

Hibernate: insert into student\_course\_map (course\_id, student\_id) values (?, ?)







Inserting data into course\_student\_map table:

