**Model Student Course Registration – Use whatever technology you prefer, not necessarily JPA/Hibernate (preferred since that is what we use).**

Student: ID, Name

Course: ID, Name

A student can take many courses and a course can have many students.

1).Write skeleton code of entity bean (or whatever technologies you like) classes to model Student and Course and student course registration.

2). Write a skeleton Student DAO class that support

2.1). add a new student along with their course registrations.  
 2.2). Delete a student.

2.3).Get all students, sorted by their name, for a given course with course name as input.

**Bonus Points**:

2.4). What if we want to record course scores? What possible changes need to be made?   
 Explain briefly.

To record course scores in the existing student-course relationship, you need to make changes to both your entity models and the database schema. Specifically, you should introduce a new entity that represents the relationship between Student and Course, including an additional field for the score.  
  
**Changes Needed:**  
1. **Create a new StudentCourse entity:** This entity will have a composite key representing the Student and Course, as well as a score field to store the score for the course.   
2. **Modify the Student and Course entities:** They should be linked through the new StudentCourse entity instead of a direct many-to-many relationship.  
  
**Detailed Changes:**  
**1. Create StudentCourse Entity**  
This entity will represent the many-to-many relationship between Student and Course and include a score.  
package com.example.university.entity;

import javax.persistence.\*;  
import java.io.Serializable;

@Entity   
public class StudentCourse {

@EmbeddedId   
private StudentCourseKey id;

@ManyToOne   
@MapsId("studentId")   
@JoinColumn(name = "student\_id")   
private Student student;

@ManyToOne   
@MapsId("courseId")   
@JoinColumn(name = "course\_id")   
private Course course;

private Double score;

// Getters and Setters

}

@Embeddable   
class StudentCourseKey implements Serializable {

@Column(name = "student\_id")   
private Long studentId;

@Column(name = "course\_id")   
private Long courseId;

// Getters, Setters, hashCode, and equals

}  
  
**2.Modify Student Entity**  
Update the Student entity to include a one-to-many relationship with StudentCourse.   
package com.example.university.entity;

import javax.persistence.\*;   
import java.util.Set;

@Entity   
public class Student {

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

private String name;

@OneToMany(mappedBy = "student", cascade = CascadeType.ALL)   
private Set<StudentCourse> studentCourses;

// Getters and Setters

}

**3. Modify Course Entity**  
Update the Course entity to include a one-to-many relationship with StudentCourse.

package com.example.university.entity;

import javax.persistence.\*;   
import java.util.Set;

@Entity   
public class Course {

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

private String name;

@OneToMany(mappedBy = "course", cascade = CascadeType.ALL)   
private Set<StudentCourse> studentCourses;

// Getters and Setters

}

**4. Update Database Schema**  
Update the database schema to reflect these changes.   
CREATE TABLE student (   
id BIGINT AUTO\_INCREMENT PRIMARY KEY,   
name VARCHAR(255) NOT NULL   
);

CREATE TABLE course (   
id BIGINT AUTO\_INCREMENT PRIMARY KEY,   
name VARCHAR(255) NOT NULL );

CREATE TABLE student\_course (   
student\_id BIGINT,   
course\_id BIGINT,   
score DOUBLE,   
PRIMARY KEY (student\_id, course\_id), FOREIGN KEY (student\_id) REFERENCES student(id), FOREIGN KEY (course\_id) REFERENCES course(id)   
);  
  
**Modify Repositories**  
Create a repository for the StudentCourse entity.

package com.example.university.repository;   
import com.example.university.entity.StudentCourse;   
import com.example.university.entity.StudentCourseKey;   
import org.springframework.data.jpa.repository.JpaRepository;   
import org.springframework.stereotype.Repository;

@Repository   
public interface StudentCourseRepository extends JpaRepository<StudentCourse, StudentCourseKey> {

}  
  
**Modify Service Layer**  
Update the service layer to handle the new entity.

package com.example.university.service;

import com.example.university.entity.Student;   
import com.example.university.entity.StudentCourse;   
import com.example.university.repository.StudentRepository;   
import com.example.university.repository.StudentCourseRepository;   
import com.example.university.repository.CourseRepository;   
import org.springframework.beans.factory.annotation.Autowired;   
import org.springframework.stereotype.Service;   
  
import javax.transaction.Transactional;   
import java.util.List;

@Service   
public class StudentService {

@Autowired   
private StudentRepository studentRepository;

@Autowired   
private StudentCourseRepository studentCourseRepository;

@Autowired   
private CourseRepository courseRepository;

@Transactional   
public void addStudent(Student student, List<StudentCourse> studentCourses) { studentRepository.save(student);   
studentCourseRepository.saveAll(studentCourses);   
}

@Transactional   
public void deleteStudent(Long studentId) {   
studentRepository.deleteById(studentId);   
}

public List<Student> getStudentsByCourseName(String courseName) {   
return studentCourseRepository.findByCourseName(courseName)   
.stream()   
.map(StudentCourse::getStudent)   
.distinct()   
.sorted((s1, s2) -> s1.getName().compareTo(s2.getName()))   
.toList();   
}

public List<Student> getStudentsNotInCourse(String courseName) {   
return studentCourseRepository.findByCourseNameAndStudentIsNull(courseName)   
.stream()   
.map(StudentCourse::getStudent)   
.distinct()   
.toList();   
}   
}

2.5). How to find all students who don’t register for a given course?

**Notes**:

1. Code skeleton is enough. No need to write every single line.
2. Need both entity bean classes as well as table DDLs.
3. For the DAO/Repository classes need query details for relevant questions.
4. Show proper transaction management.
5. **Show best practice(s)** when you can solve the problems with multiple approaches. If possible comment why one approach is better than the others.
6. We love Hibernate and also hate it!! Would love to hear your opinion on best practices.
7. Please upload to Github and send us the link afterwards.

**Time**: 45 – 60 minutes.