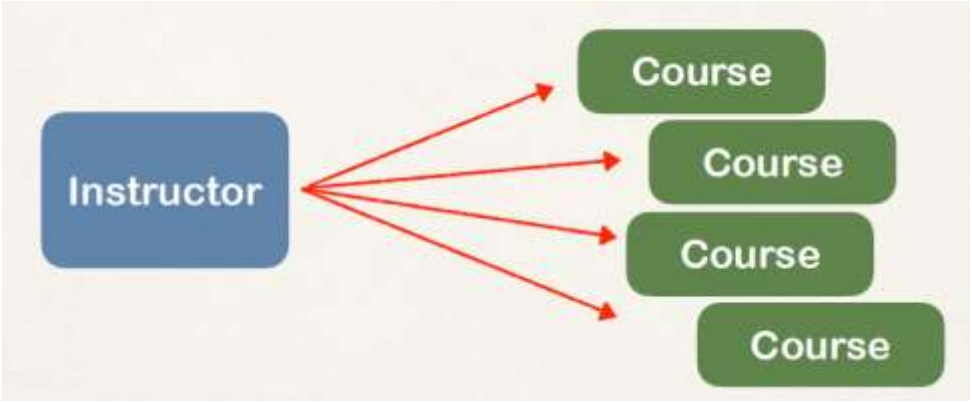
JPA
Entity Relationships

Entity Relationships

- In the database, you most likely will have
  - Multiple Tables
  - Relationships between Tables

- Need to model this with JPA/Hibernate



Entity Relationships

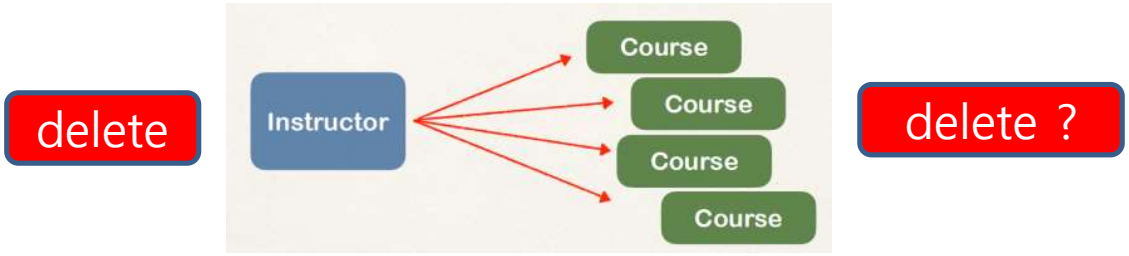
- There are four types of relationship multiplicities:
  - @OneToOne
  - @OneToMany, @ManyToOne
  - @ManyToMany
- The direction of a relationship can be:
  - bidirectional : owning side and inverse side
  - unidirectional : owning side only
- The 'owning' side is the entity whose table will hold the reference

Entity Relation Attributes

① ②

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

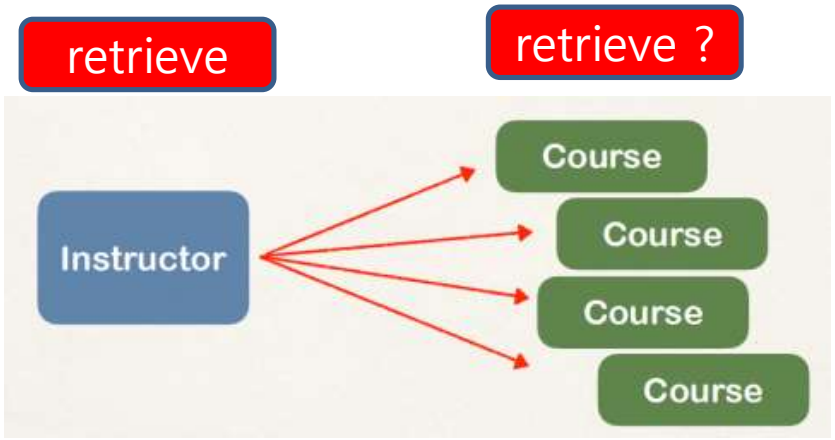
1) Cascading updates/deletes
   - Apply the same operation to associated entities
   - CascadeType
     - ALL, PERSIST, MERGE, REMOVE, REFRESH
     - By default, no operations are cascaded



Entity Relation Attributes

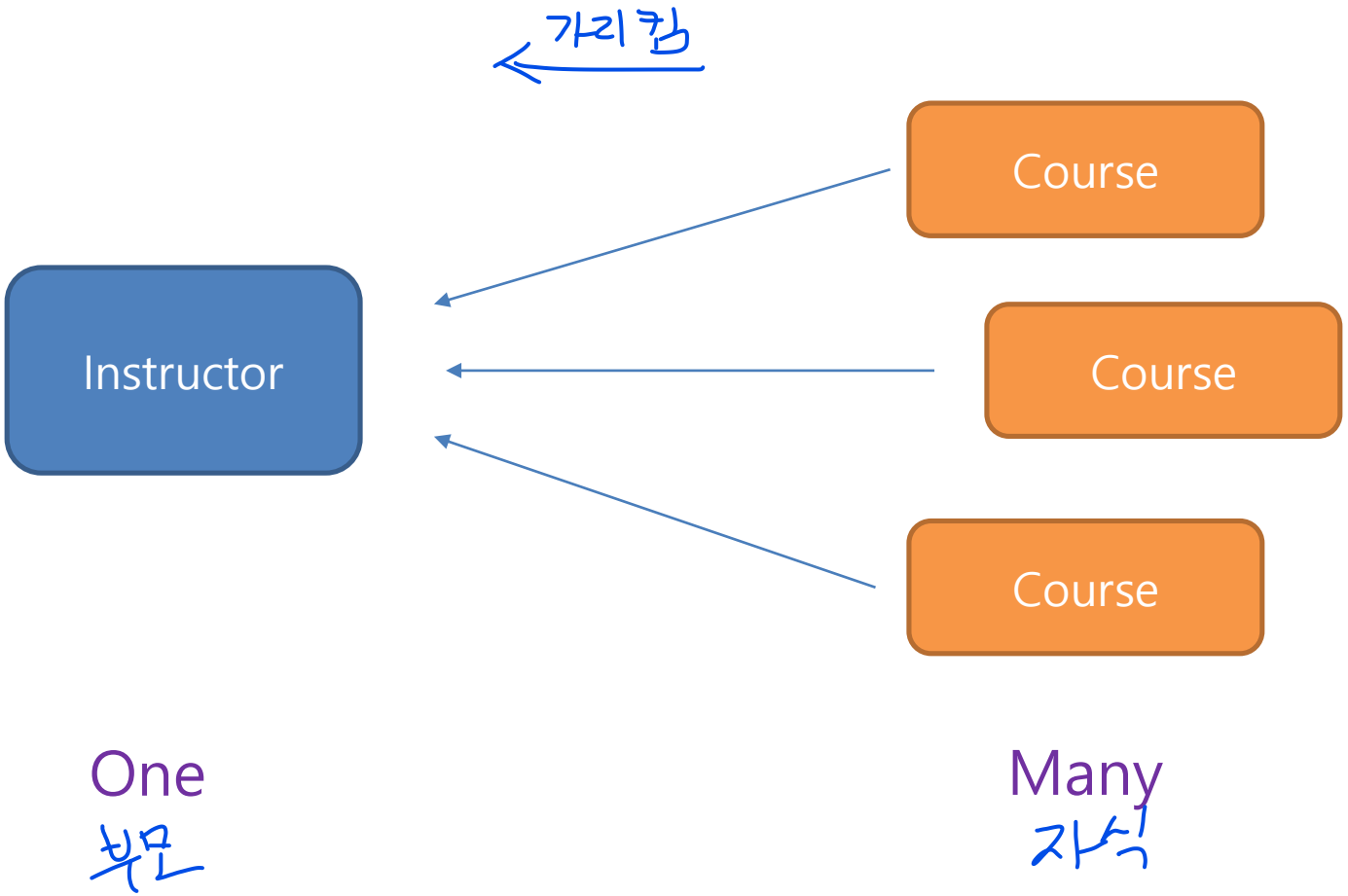
2) Fetching strategy to retrieve associated entities
   - FetchType: LAZY, EAGER
     - Eager will retrieve everything. Could easily turn into a performance nightmare
     - Lazy will retrieve on request. Lazy means don't load row until the property is retrieved

Mapping	Default Fetch Type
@OneToOne	FetchType.EAGER
@OneToMany	FetchType.LAZY
@ManyToOne	FetchType.EAGER
@ManyToMany	FetchType.LAZY



① OneToMany Unidirectional

- An instructor can have many courses



OneToMany Unidirectional

```java
@Entity
@Table(name="instructor")
public class Instructor {
    @Id
    @GeneratedValue
    @Column(name="id")
    private Long id;

    @Column(name="full_name")
    private String fullName;

    @Column(name="email")
    private String email;
}
```

```java
@Entity
@Table(name="course")
public class Course {
    @Id
    @GeneratedValue
    @Column(name="id")
    private Long id;

    @Column(name="title")
    private String title;

    @ManyToOne
    @JoinColumn(name="instructor_id")
    private Instructor instructor;
}
```

Course가 instructor 가지기

id	email	full_name
1	nykim@hansung.ac.kr	Namyun Kim
2	jmlee@hansung.ac.kr	Jaemon Lee

id	title	instructor_id
1	웹프레임워크	1
2	오픈소스소프트웨어	1
3	iOS 프로그래밍	2
4	안드로이드 프로그래밍	2

foreign key

(참고) Primary Key and Foreign Key

- Primary key
  - identify a unique row in a table
- Foreign key
  - link tables together
  - a field in one table that refers to primary key in another table

OneToMany Unidirectional

- For One-to-Many relationship, the foreign key is always in the "Many" side of the relationship
  - The parent table (instructor), child table (course)
  - The child table houses the foreign key
- We are able to tell JPA/Hibernate which object is the child object by assigning the @ManyToOne
- We are able to tell JPA/Hibernate which object is the parent object by assigning the @OneToMany annotation
- @JoinColumn annotation allows you to specify which of the columns you want to use as the join and it allows you to name the column as well

CourseDao

```
@Repository
@Transactional
public class CourseDao {
    @PersistenceContext
    private EntityManager entityManager;

    public void save(Course course) {
        entityManager.persist(course);
    }

    public Course findById(Long id) {
        return entityManager.find(Course.class, id);
    }

    public List<Course> findAll() {
        return entityManager.createQuery("SELECT p FROM Course p",
                        Course.class).getResultList();
    }
}
```

InstructorDao

```
@Repository
@Transactional
public class InstructorDao {
    @PersistenceContext
    private EntityManager entityManager;

    public void save(Instructor instructor) {
        entityManager.persist(instructor);
    }

    public Instructor findById(Long id) {
        return entityManager.find(Instructor.class, id);
    }

    public List<Instructor> findAll() {
        return entityManager.createQuery("SELECT c FROM Instructor c",
                                         Instructor.class).getResultList();
    }
```

main

```
Instructor instructor1 = new Instructor("Namyun Kim", "nykim@hansung.ac.kr");
Course course1 = new Course("웹프레임워크");
Course course2 = new Course("오픈소스소프트웨어");

Instructor Instructor2 = new Instructor("Jaemon Lee", "jmlee@hansung.ac.kr");
Course course3 = new Course("iOS 프로그래밍");
Course course4 = new Course("안드로이드 프로그래밍");

instructorDao.save(instructor1);
instructorDao.save(instructor2);

// Instructor 객체를 먼저 저장한 후, Course 객체를 저장해야 한다.
// Course 객체 내부에 Instructor 참조가 있기 때문에, Instructor가 먼저 영속화되어야 한다.
course1.setInstructor(instructor1);
course2.setInstructor(instructor1);
course3.setInstructor(instructor2);
course4.setInstructor(instructor2);

courseDao.save(course1);
courseDao.save(course2);
courseDao.save(course3);
courseDao.save(course4);
```

id	email	full_name
1	nykim@hansung.ac.kr	Namyun Kim
2	jmlee@hansung.ac.kr	Jaemon Lee

id	title	instructor_id
1	웹프레임워크	1
2	오픈소스소프트웨어	1
3	iOS 프로그래밍	2
4	안드로이드 프로그래밍	2

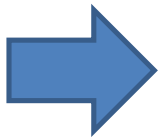
② OneToMany Bidirectional

- When we have a bidirectional relationship between objects, it means that we are able to access Object A from Object B, and Object B from Object A

- To use bidirectional, we can keep the existing database schema
  - No changes required to database
  - Simply update the Java code

Instructor

```java
@Entity
@Table(name="instructor")
public class Instructor {
    @Id
    @GeneratedValue
    @Column(name="id")
    private Long id;

    @Column(name="full_name")
    private String fullName;

    @Column(name="email")
    private String email;
}
```



```java
@Entity
@Table(name="instructor")
public class Instructor {

...
```

Instructor 와 Course 가리킴

```java
@OneToMany(mappedBy = "instructor",
        fetch = FetchType.LAZY,
        cascade=CascadeType.ALL)
private List<Course> courses =
        new ArrayList<>();
```

```java
// 연관 관계 편의 메소드
    public void addCourse(Course course) {
        courses.add(course);
        course.setInstructor(this);
    }
}
```

OneToMany Bidirectional

- mappedBy tells JPA/Hibernate
  - Look at the *instructor* property in the Course class
  - To help find associated courses for instructor

```
public class Instructor {
    ...

    @OneToMany(mappedBy="Instructor")
    private List<Course> courses;
```

```
public class Course {
    ...

    @ManyToOne
    @JoinColumn(name="Instructor_Id")
    private Instructor Instructor;
```

main

생성
```
Instructor instructor1 = new Instructor("Namyun Kim", "nykim@hansung.ac.kr");
Course course1 = new Course("웹프레임워크");
Course course2 = new Course("오픈소스소프트웨어");
```

접근
```
instructor1.addCourse(course1);
instructor1.addCourse(course2);

// cascade=CascadeType.ALL, fetch = FetchType.LAZY
instructorDao.save(instructor1);
```
Course도 저장됨

```
// 저장된 Instructor 조회 및 결과 확인
Instructor retrievedInstructor = instructorDao.findById(instructor1.getId());
System.out.println("Instructor: " + retrievedInstructor.getFullName());

for (Course Course : retrievedInstructor.getCourses()) {
    System.out.println("Course: " + Course.getTitle());
}
```

Exception in thread "main" org.hibernate.LazyInitializationException:

Solution

InstructorDao

```
@Transactional
public Instructor findByIdWithCourses(Long id) {
        Instructor instructor = entityManager.find(Instructor.class, id);
        if (instructor != null) {
            instructor.getCourses().size(); // 컬렉션 로드
        }
        return instructor;
}
```

main

```
// Instructor retrievedInstructor = instructorDao.findById(instructor1.getId());
Instructor retrievedInstructor = instructorDao.findByIdWithCourses(instructor1.getId())
```

To persist all the objects correctly,
you'll need to follow these generic steps:

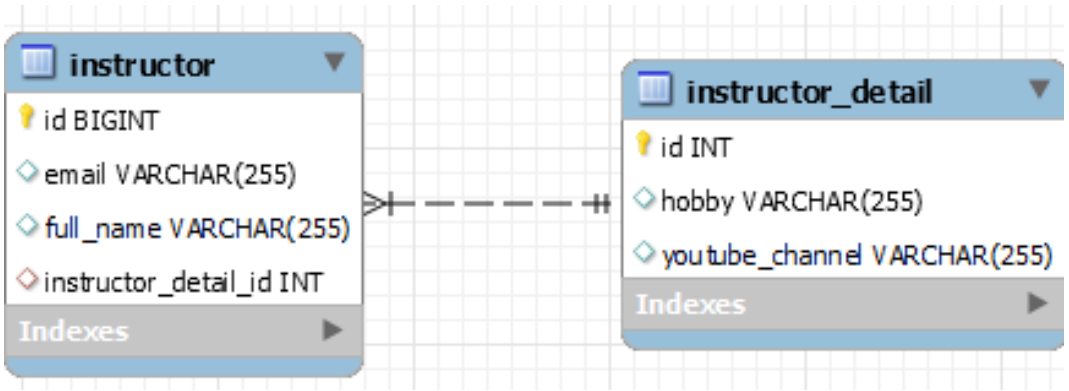
1) Instantiate parent object
2) Instantiate child objects

3) Set the parent object in the child objects
4) Set the collection of child objects on the parent

5) Save the parent

③ One-To-One Unidirectional

An instructor can have an “instructor detail” entity



InstructorDetail

```
@Entity
@Table(name="instructor_detail")
public class InstructorDetail {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;

    @Column(name = "youtube_channel")
    private String youtubeChannel;

    @Column(name = "hobby")
    private String hobby;
}
```

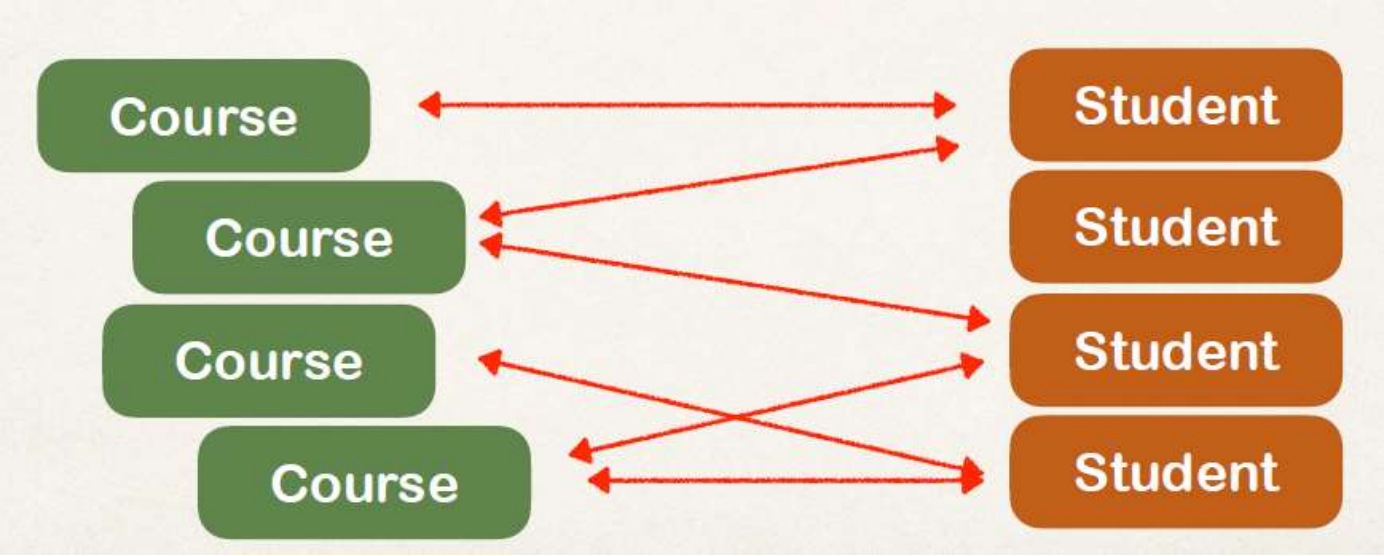
Instructor

```
@Entity
@Table(name="instructor")
public class Instructor {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id")
    private Long id;

    ...

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "instructor_detail_id")
    private InstructorDetail instructorDetail;

    }
```

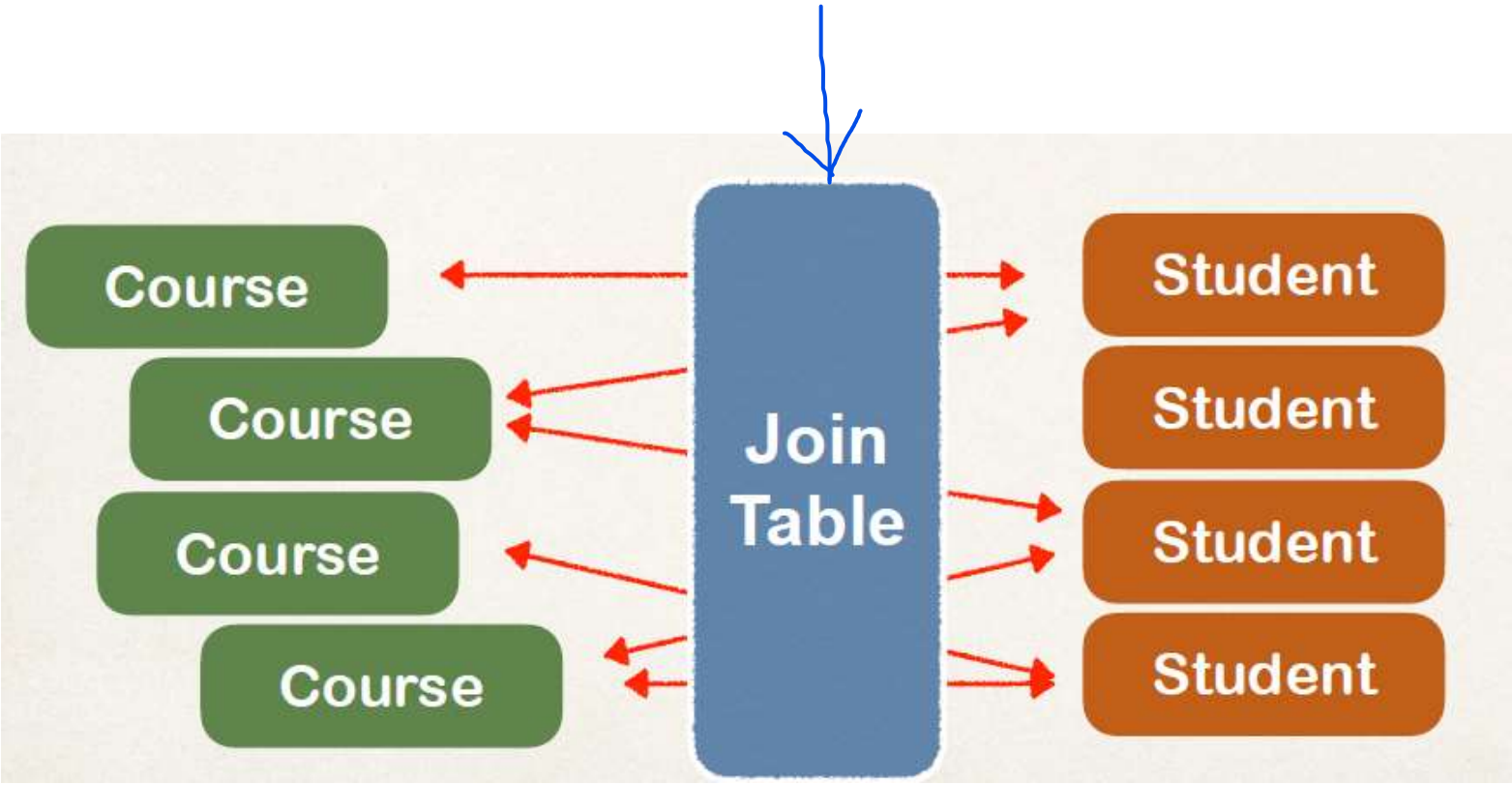
```
// InstructorDetail 객체 생성
InstructorDetail detail =
                new InstructorDetail("youtube.com/TheJavaChannel", "Coding");

// Instructor 객체 생성 및 InstructorDetail 설정
Instructor instructor = new Instructor("Namyun Kim", "nykim@hansung.ac.kr");
instructor.setInstructorDetail(detail);

instructorDao.save(instructor);  // cascade = CascadeType.ALL

// 저장된 instructor 객체 조회
Instructor storedInstructor = instructorDao.findById( instructor.getId());
System.out.println("Retrieved Instructor: " + storedInstructor.getFullName());

// 연결된 InstructorDetail 정보 출력
InstructorDetail storedDetail = storedInstructor.getInstructorDetail();
System.out.println("Instructor Detail: ");
System.out.println("YouTube Channel: " + storedDetail.getYoutubeChannel());
System.out.println("Hobby: " + storedDetail.getHobby());
```

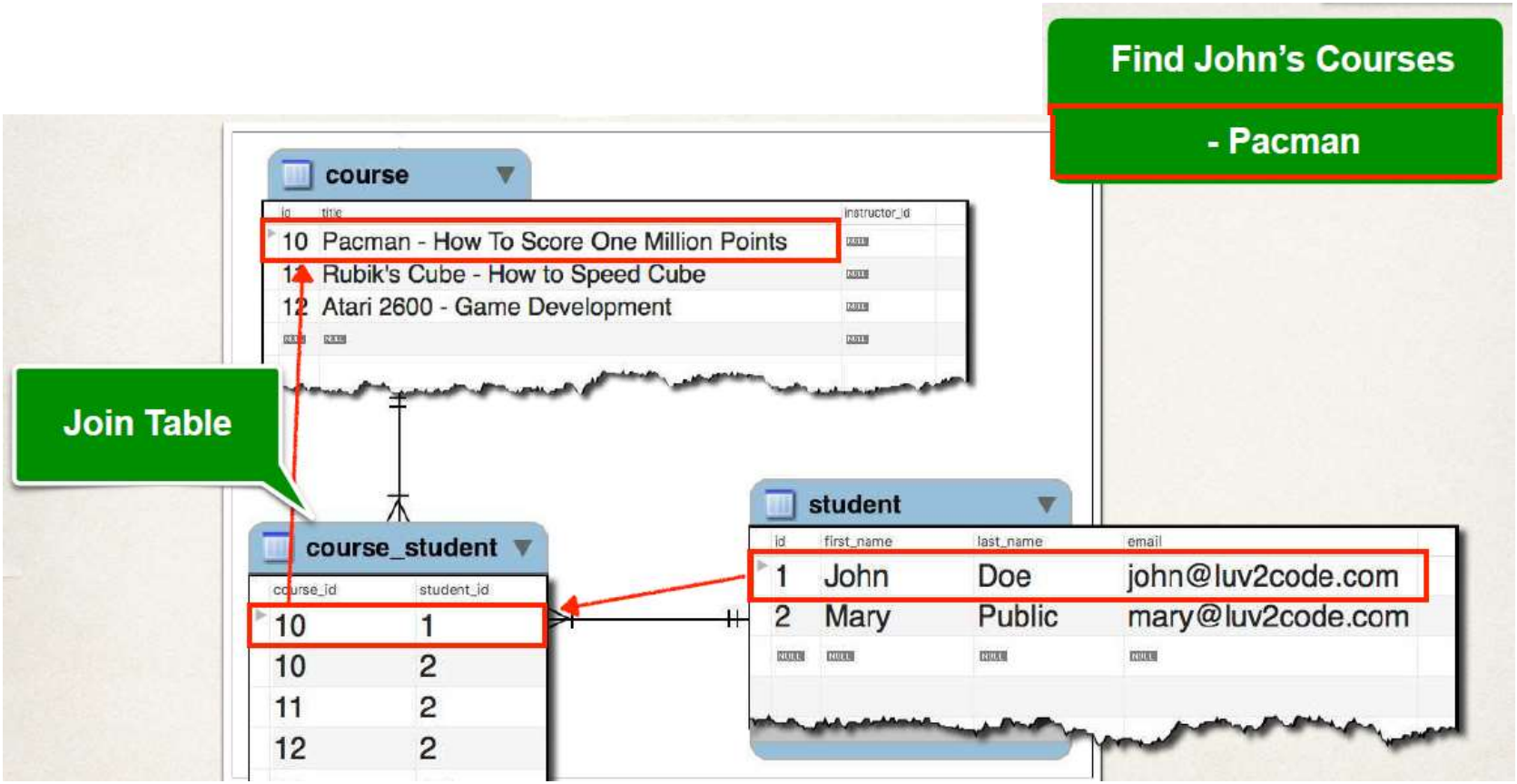
④ ManyToMany Unidirectional

- A course can have many students
- A student can have many courses



Keep track of relationships




Find John's Courses
- Pacman
course
id
title
instructor_id
10 Pacman - How To Score One Million Points
11 Rubik's Cube - How to Speed Cube
12 Atari 2600 - Game Development
Join Table
course_student
course_id
student_id
10 1
10 2
11 2
12 2
student
id
first_name
last_name
email
1 John Doe john@luv2code.com
2 Mary Public mary@luv2code.com

ManyToMany Unidirectional

- Join Table
  - A good design for a Many-to-Many relationship makes use of something called a join table. The term join table is just a fancy way of describing a third SQL table that only holds primary keys
  - By convention, the name of this join table is usually just the combination of the two tables of the many-to-many relationship. In this case it's just course_student
  - This join table only contains the primary keys from the course and student tables

Student

```
@Entity
@Table(name = "student")
public class Student {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;

    @Column(name = "full_name")
    private String fullName;

    @Column(name = "email")
    private String email;

    @ManyToMany
    @JoinTable(
            name = "student_course",
            joinColumns = @JoinColumn(name = "student_id"),
            inverseJoinColumns = @JoinColumn(name = "course_id")
    )
    private List<Course> courses;
```

Main

```
// create a course
Course course1 = new Course("웹프레임워크");
Course course2 = new Course("오픈소스소프트웨어");
Course course3 = new Course("정보보안");
Course course4 = new Course("웹서버프로그래밍");
Course course5 = new Course("클라우드컴퓨팅");

//        courseDao.save(course1);
//        courseDao.save(course2);
//        courseDao.save(course3);
//        courseDao.save(course4);
//        courseDao.save(course5);
// course1~ course5 저장하기
Arrays.asList(course1, course2, course3, course4, course5).forEach(
                courseDao::save);
```

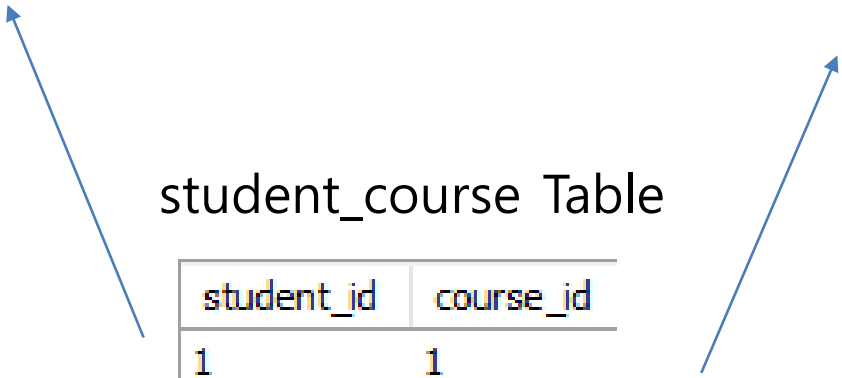
```
// create the students
Student student1 = new Student("Alice", "alice@hansung.ac.kr");
Student student2 = new Student("bob", "bob@hansung.ac.kr");
Student student3 = new Student("charlie", "charlie@hansung.ac.kr");

student1.setCourses(Arrays.asList(course1, course2));
student2.setCourses(Arrays.asList(course2, course3, course4));
student3.setCourses(Arrays.asList(course3, course4, course5));

// student1, student2, student3 저장하기
Arrays.asList(student1, student2, student3).forEach(
                studentDao::save);

// 저장된 학생 및 코스 정보 조회 및 출력
Student storedStudent = studentDao.findByIdWithCourses(student1.getId());
System.out.println("Retrieved Student: " + storedStudent.getFullName());
storedStudent.getCourses().forEach(
            course -> System.out.println("Enrolled in Course: " + course.getTitle())
);
```

student Table

id	email	full_name
1	alice@hansung.ac.kr	Alice
2	bob@hansung.ac.kr	bob
3	charlie@hansung.ac.kr	charlie

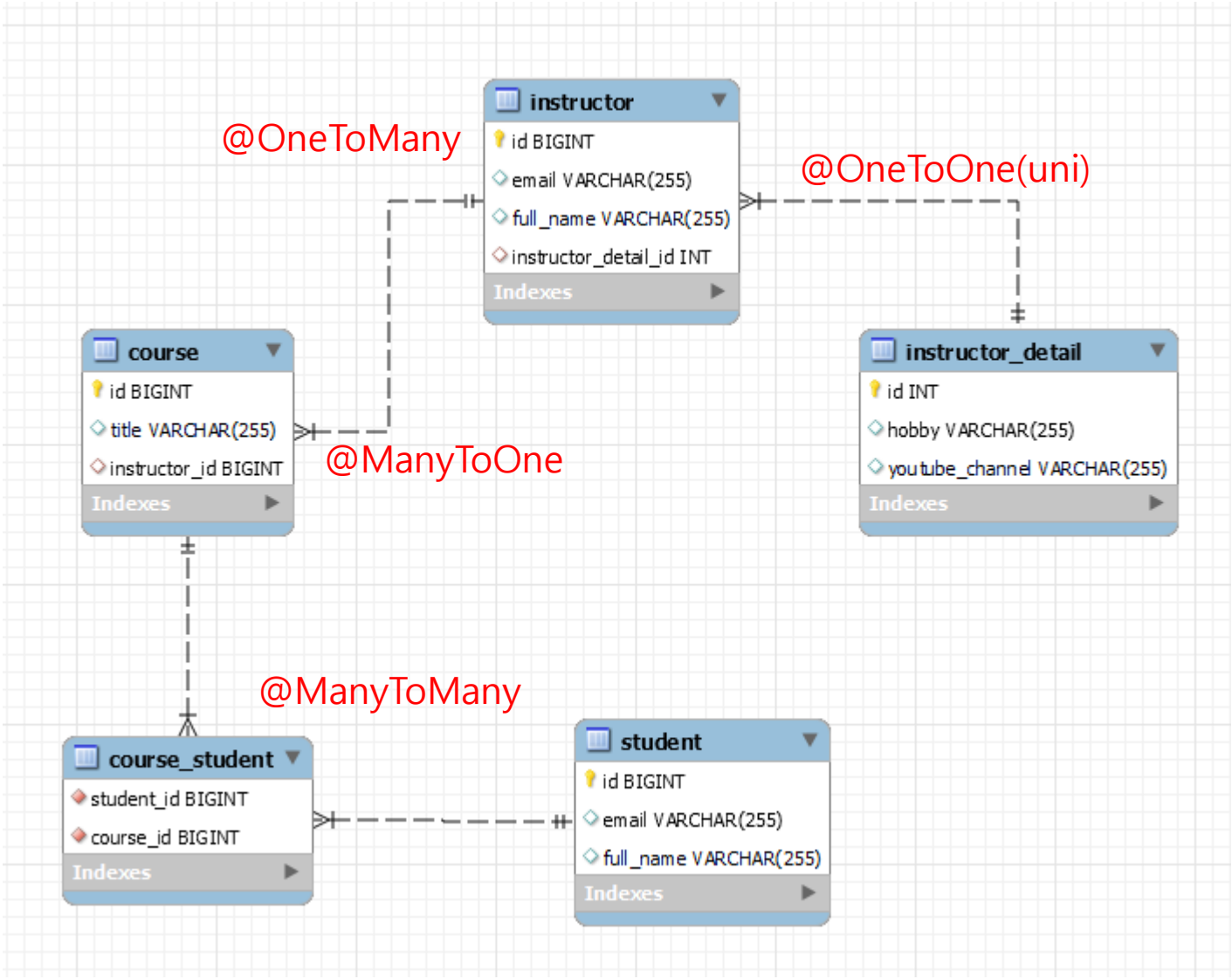
course Table

id	title	instructor_id
1	웹프레임워크	NULL
2	오픈소스소프트웨어	NULL
3	정보보안	NULL
4	웹서버프로그래밍	NULL
5	클라우드컴퓨팅	NULL



student_course Table

student_id	course_id
1	1
1	2
2	2
2	3
2	4
3	3
3	4
3	5


@OneToMany
instructor
id BIGINT
email VARCHAR(255)
full_name VARCHAR(255)
instructor_detail_id INT
Indexes
@OneToOne(uni)
course
id BIGINT
title VARCHAR(255)
instructor_id BIGINT
Indexes
@ManyToOne
instructor_detail
id INT
hobby VARCHAR(255)
youtube_channel VARCHAR(255)
Indexes
@ManyToMany
course_student
student_id BIGINT
course_id BIGINT
Indexes
student
id BIGINT
email VARCHAR(255)
full_name VARCHAR(255)
Indexes