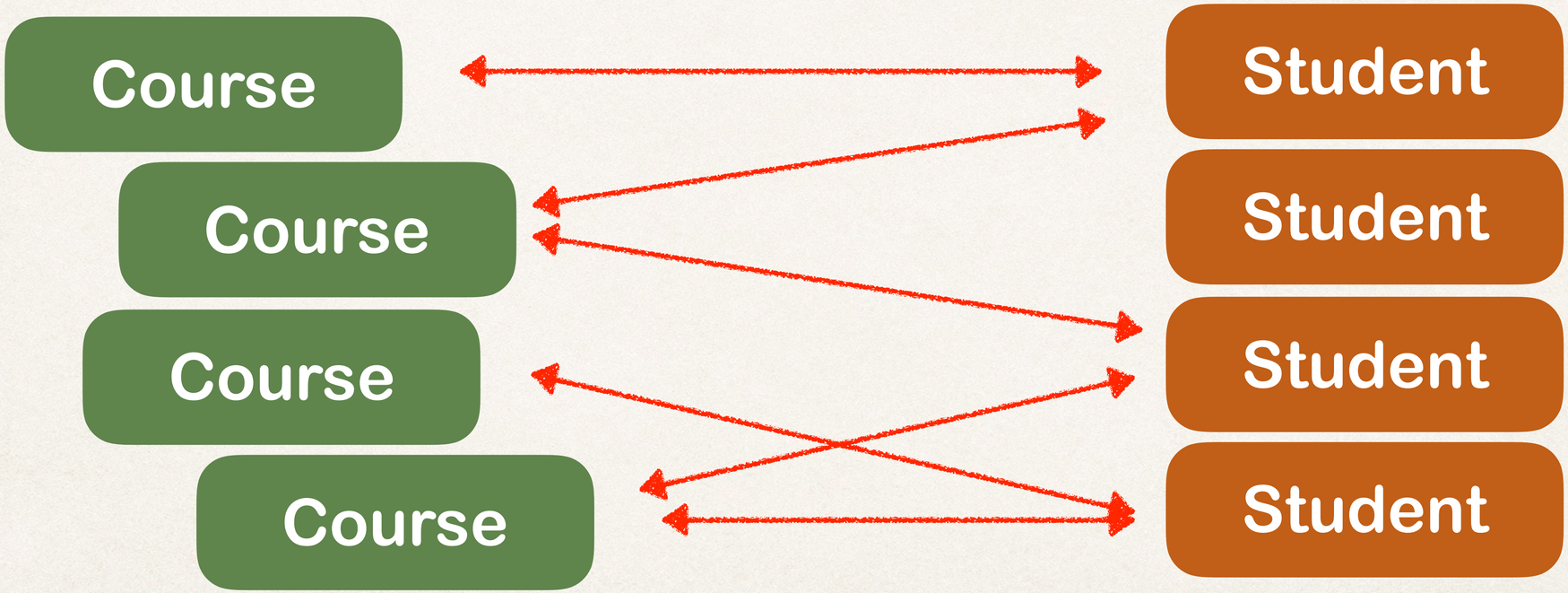
luv2code

Hibernate Many-to-Many

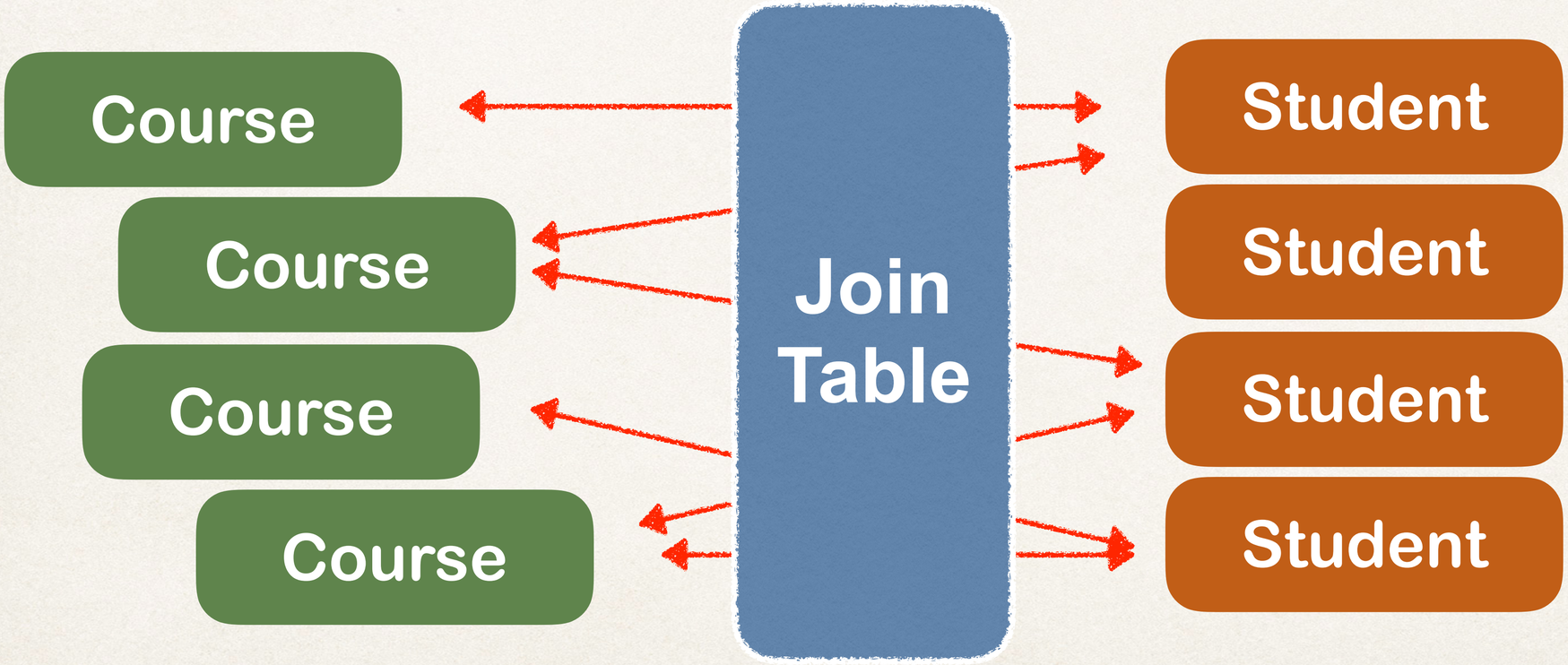
Many-to-Many Mapping

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



Keep track of relationships

- Need to track which student is in which course and vice-versa

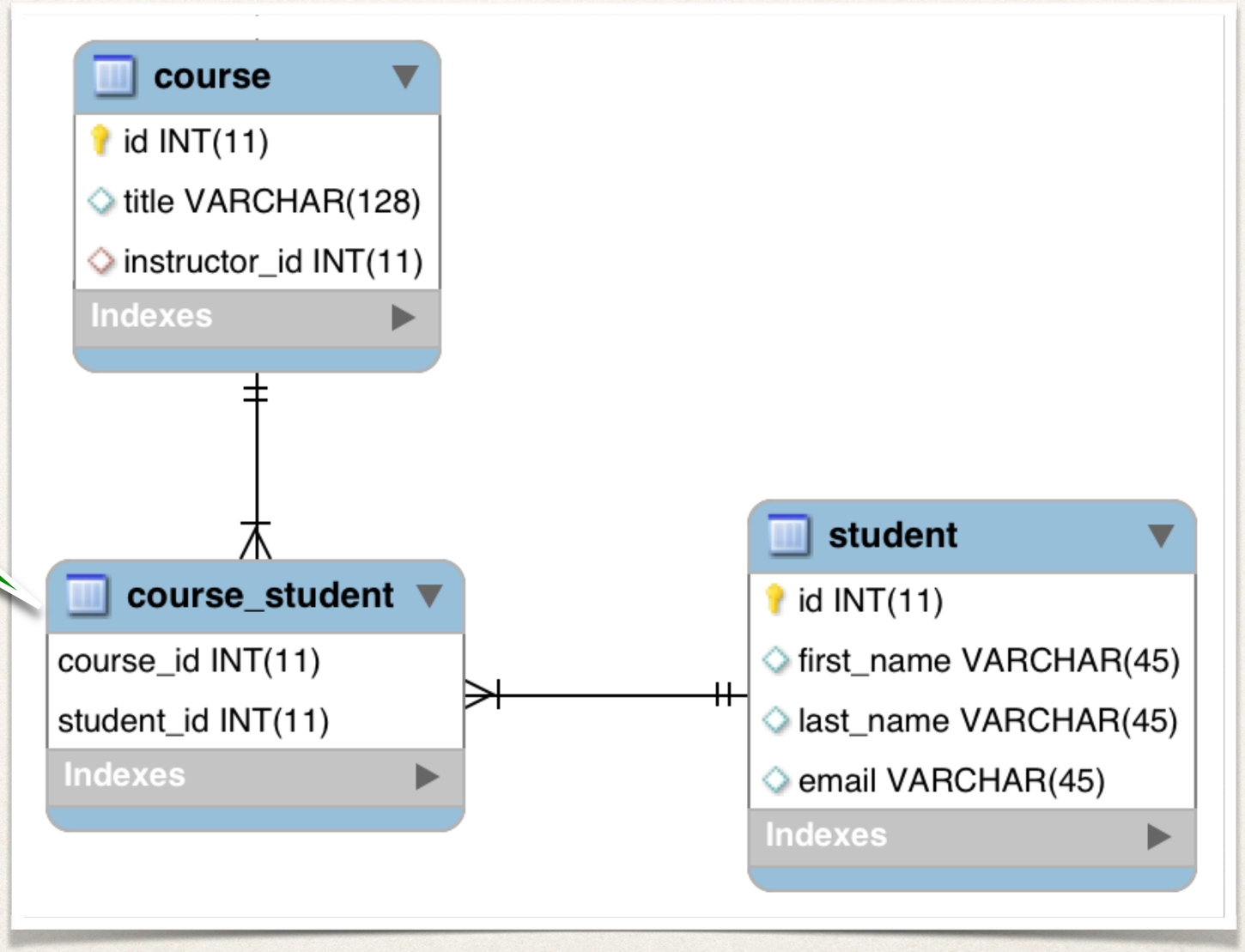


Join Table

A table that provides a mapping between two tables.

It has foreign keys for each table
to define the mapping relationship.

@ManyToMany

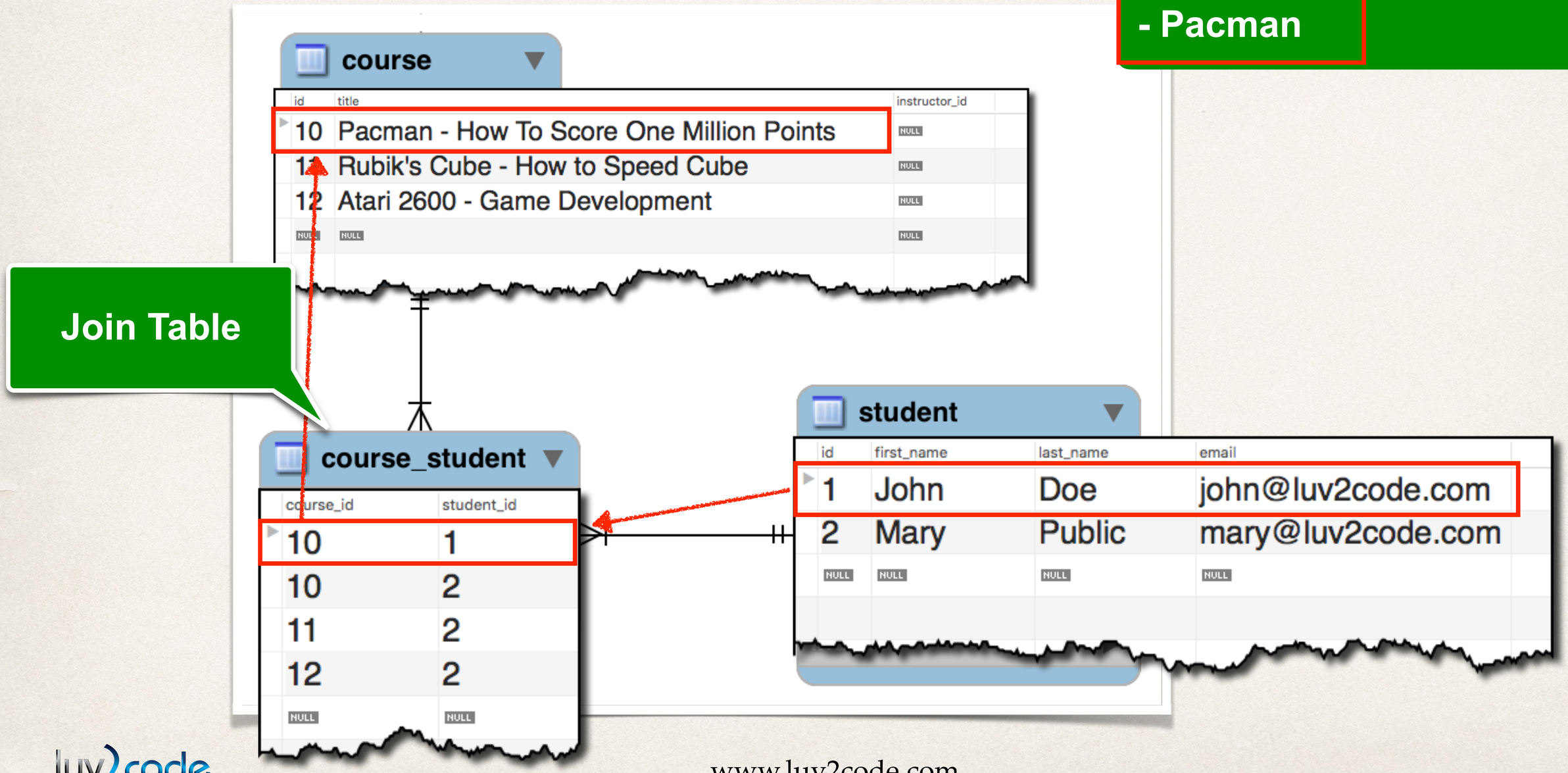


Join Table

Join Table Example

Find John's Courses

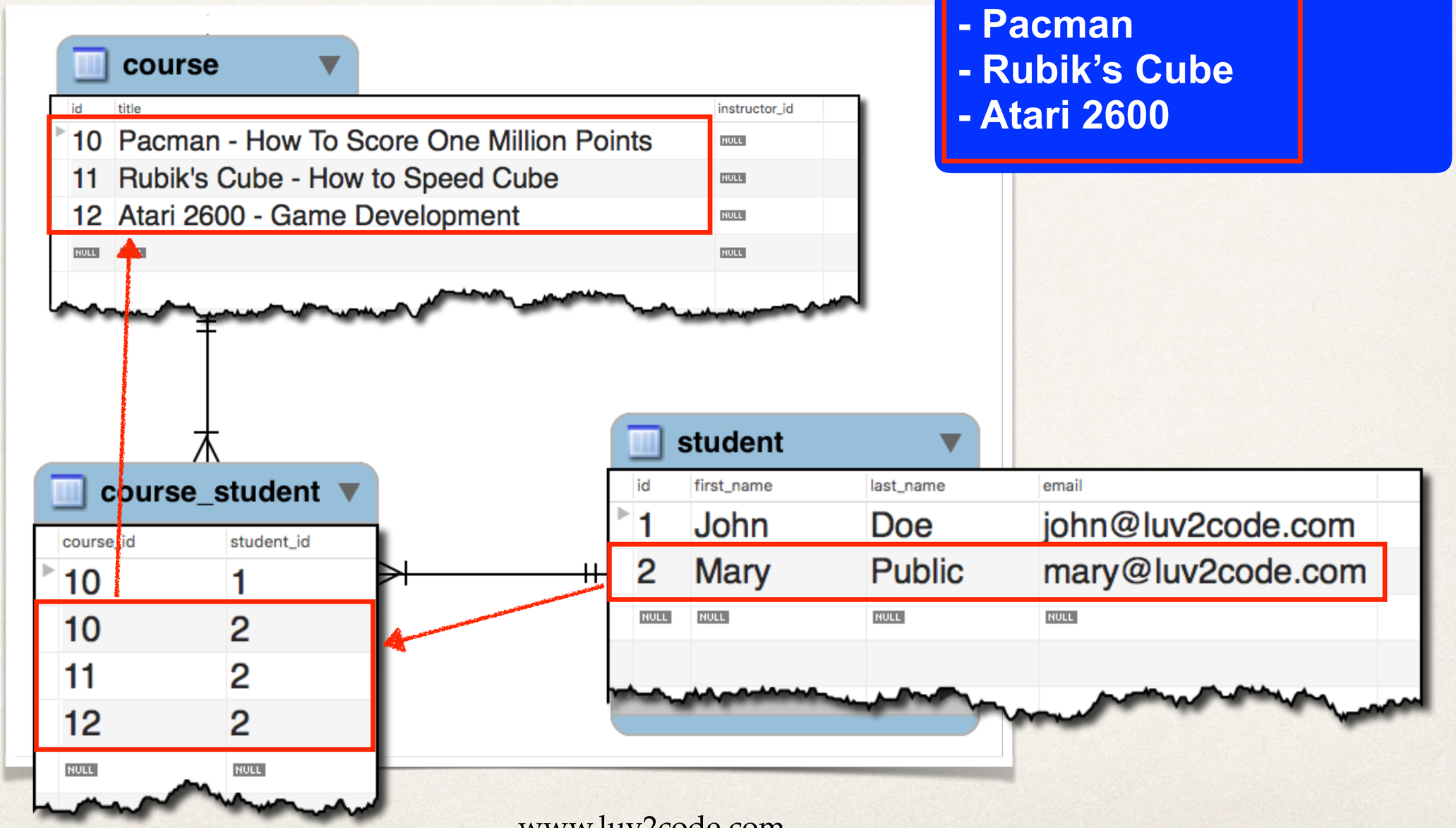
- Pacman



Join Table Example

Find Mary's Courses

- Pacman
- Rubik's Cube
- Atari 2600



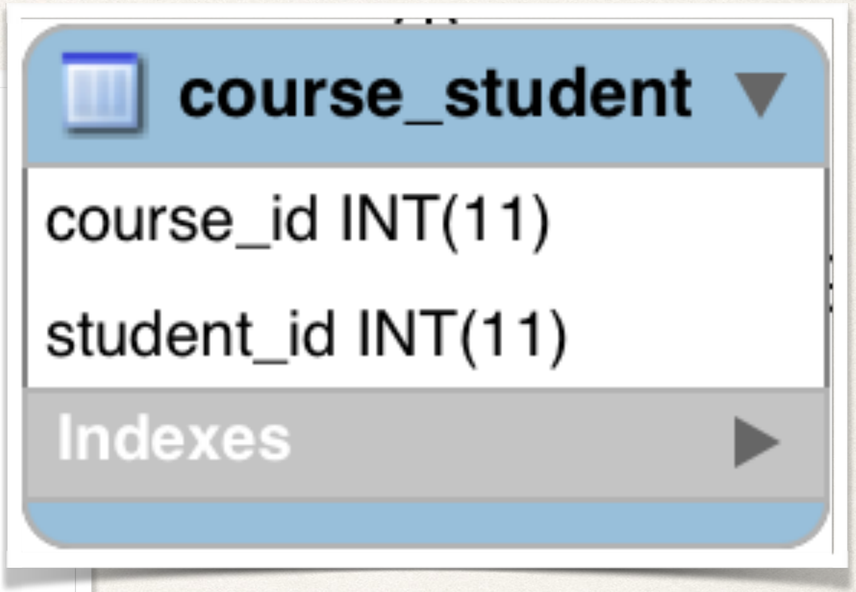
Development Process: Many-to-Many

1. Prep Work - Define database tables
2. Update **Course** class
3. Update **Student** class
4. Create Main App

join table: course_student

File: create-db.sql

```
CREATE TABLE `course_student` (

  `course_id` int(11) NOT NULL,
  `student_id` int(11) NOT NULL,

  PRIMARY KEY (`course_id`, `student_id`),

    ...
);
```



join table: course_student - foreign keys

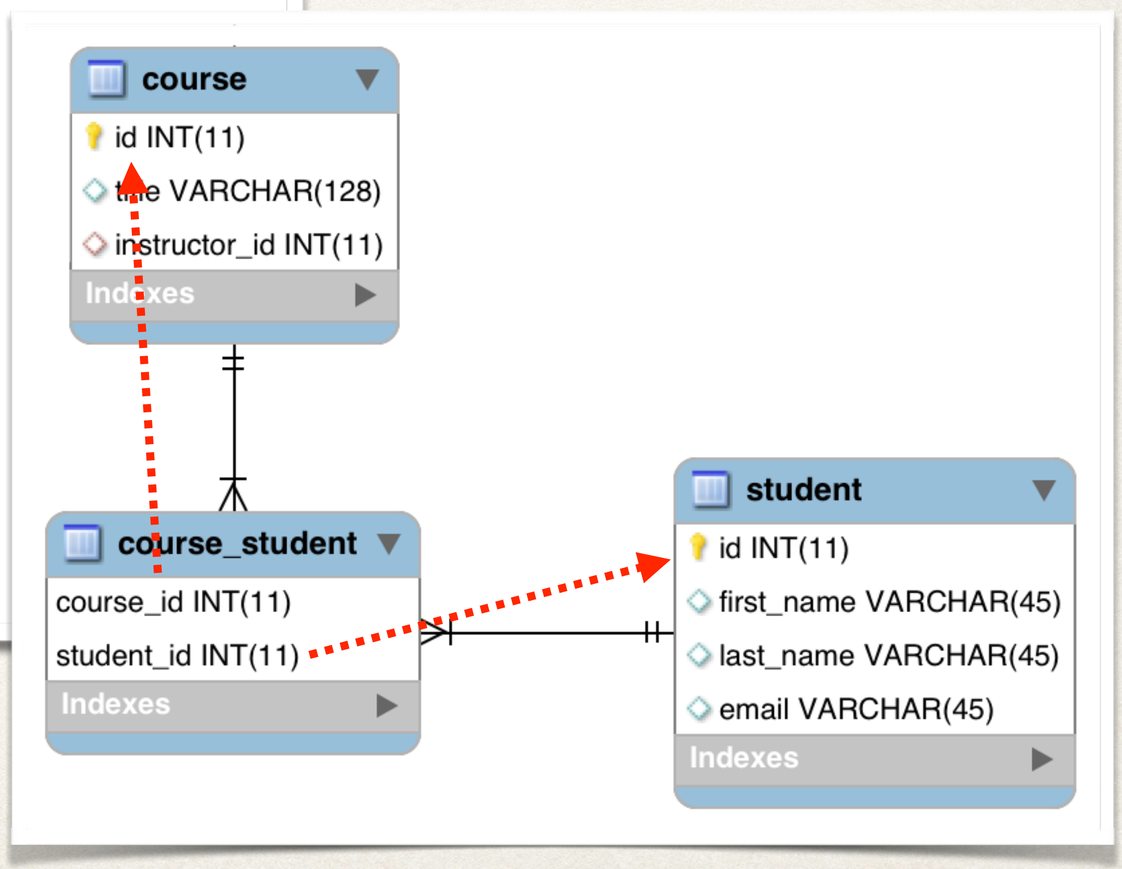
```
CREATE TABLE `course_student` (
  ...
  CONSTRAINT `FK_COURSE_05`
  FOREIGN KEY (`course_id`)
  REFERENCES `course` (`id`),

  CONSTRAINT `FK_STUDENT
  FORE (`student_i
  REFER `student` (
  ...
);
```

Table

Column

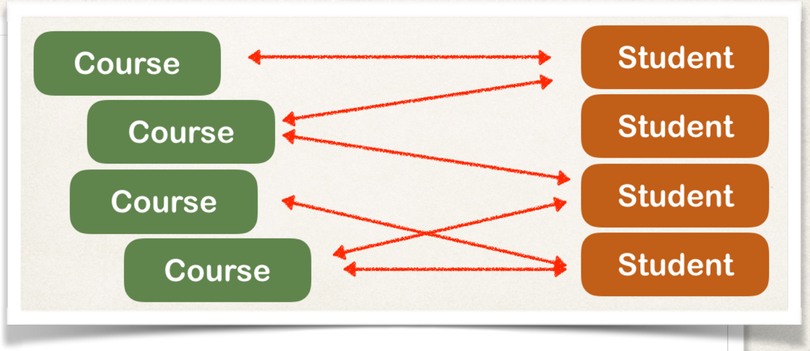
Table

Column



Step 2: Update Course - reference students

```
@Entity
@Table(name="course")
public class Course {
    ...


    private List<Student> students;

    // getter / setters
    ...

}
```

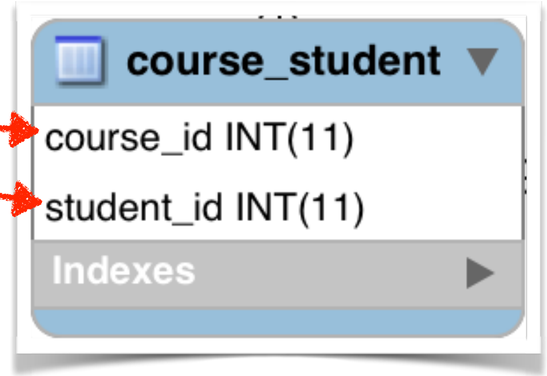


Add @ManyToMany annotation

```
@Entity
@Table(name="course")
public class Course {
    ...

    @ManyToMany
    @JoinTable(
        name="course_student",
        joinColumns=@JoinColumn(name="course_id"),
        inverseJoinColumns=@JoinColumn(name="student_id")
        )
    private List<Student> students;

    // getter / setters
    ...

}
```

Refers to
in “course

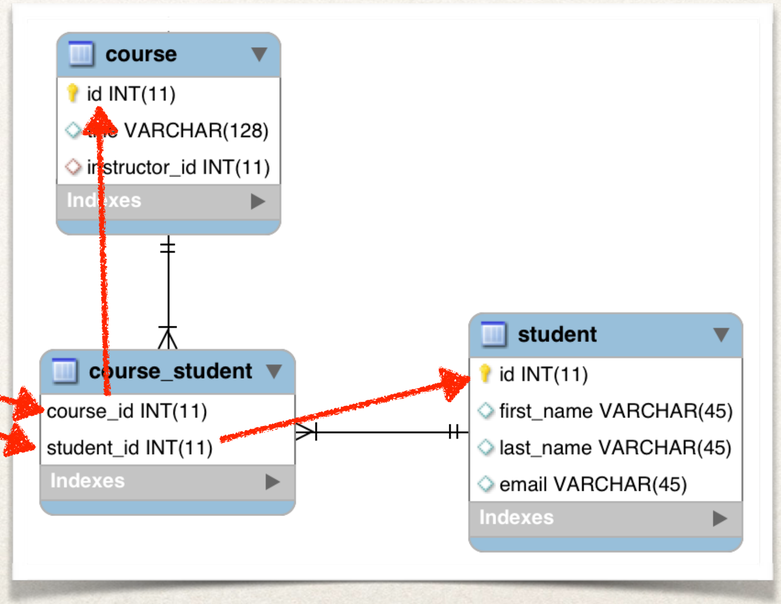
Refers to “student_id” column
in “course_student” join table



More: @JoinTable

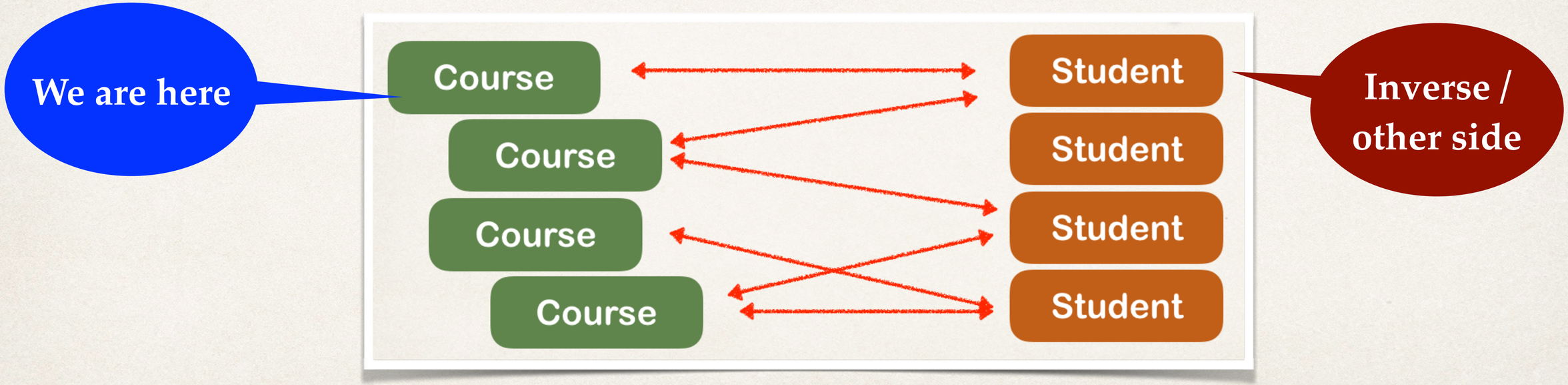
- **@JoinTable** tells Hibernate
  - Look at the **course_id** column in the **course_student** table
  - For other side (inverse), look at the **student_id** column in the **course_student** table
  - Use this information to find relationship between **course** and **students**

```
public class Course {

    @ManyToMany
    @JoinTable(
            name="course_student",
            joinColumns=@JoinColumn(name="course_id"),
            inverseJoinColumns=@JoinColumn(name="student_id")
            )
    private List<Student> students;

}
```



More on “inverse”

- In this context, we are defining the relationship in the **Course** class
- The **Student** class is on the “other side” … so it is considered the “inverse”
- “Inverse” refers to the “other side” of the relationship

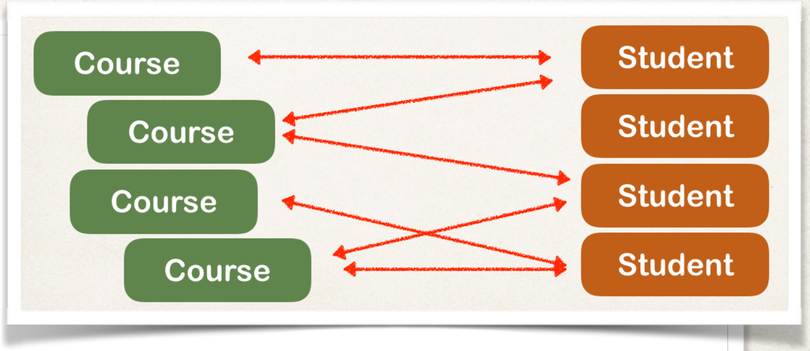


Now, let's do a similar thing for Student

just going the other way …

Step 3: Update Student - reference courses

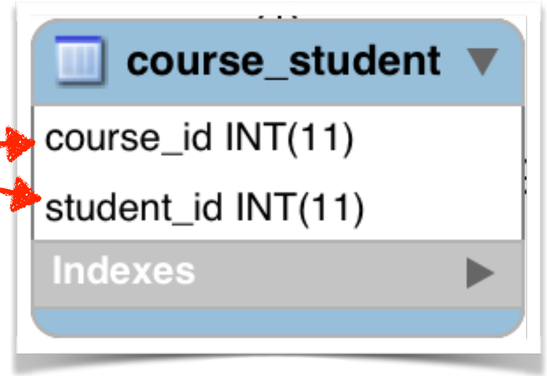
```
@Entity
@Table(name="student")
public class Student {
    ...


    private List<Course> courses;

    // getter / setters
    ...

}
```



Add @ManyToMany annotation

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

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

    // getter / setters
    ...

}
```

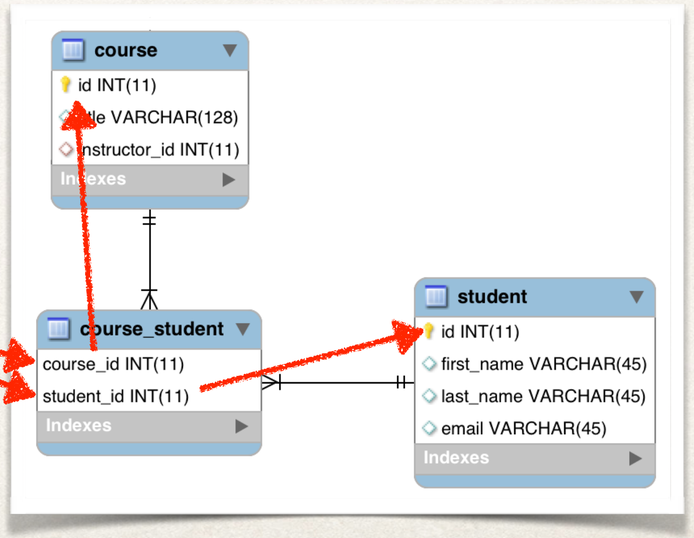
Refers to “student_id” column in “course_student” join table



More: @JoinTable

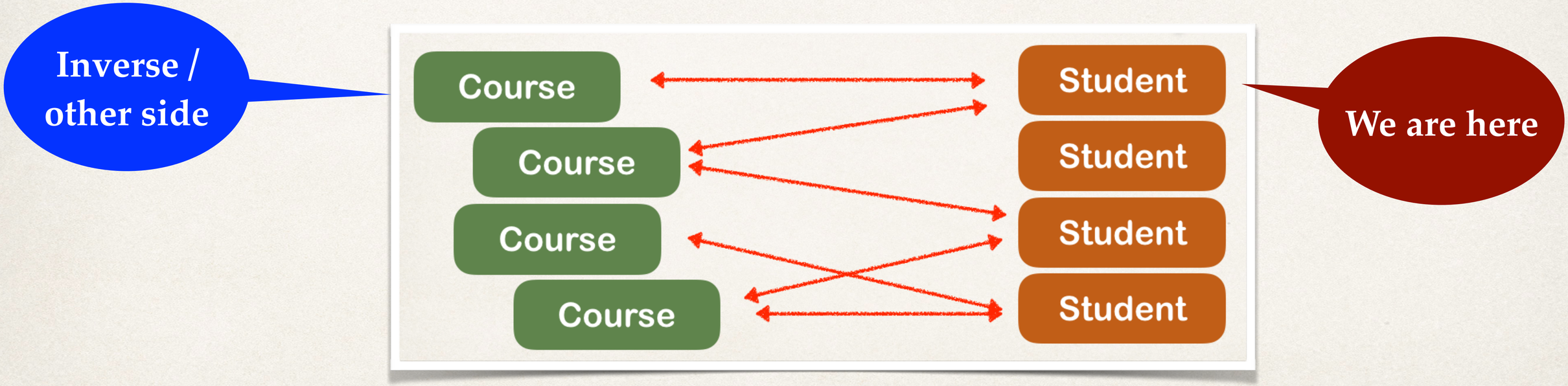
- **@JoinTable** tells Hibernate
  - Look at the **student_id** column in the **course_student** table
  - For other side (inverse), look at the **course_id** column in the **course_student** table
  - Use this information to find relationship between **student** and **courses**

```
public class Student {

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

}
```

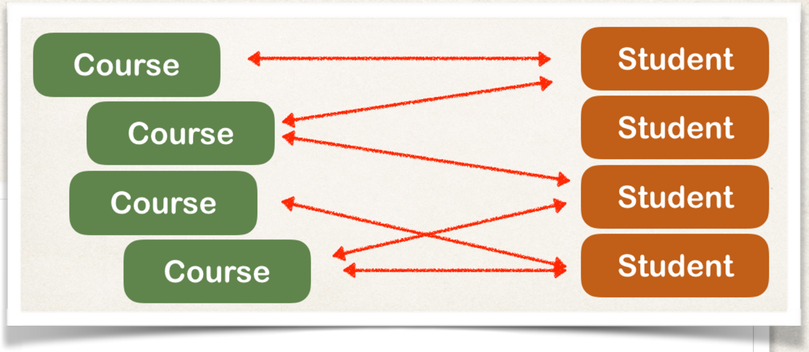


More on “inverse”

- In this context, we are defining the relationship in the **Student** class
- The **Course** class is on the “other side” … so it is considered the “inverse”
- “Inverse” refers to the “other side” of the relationship



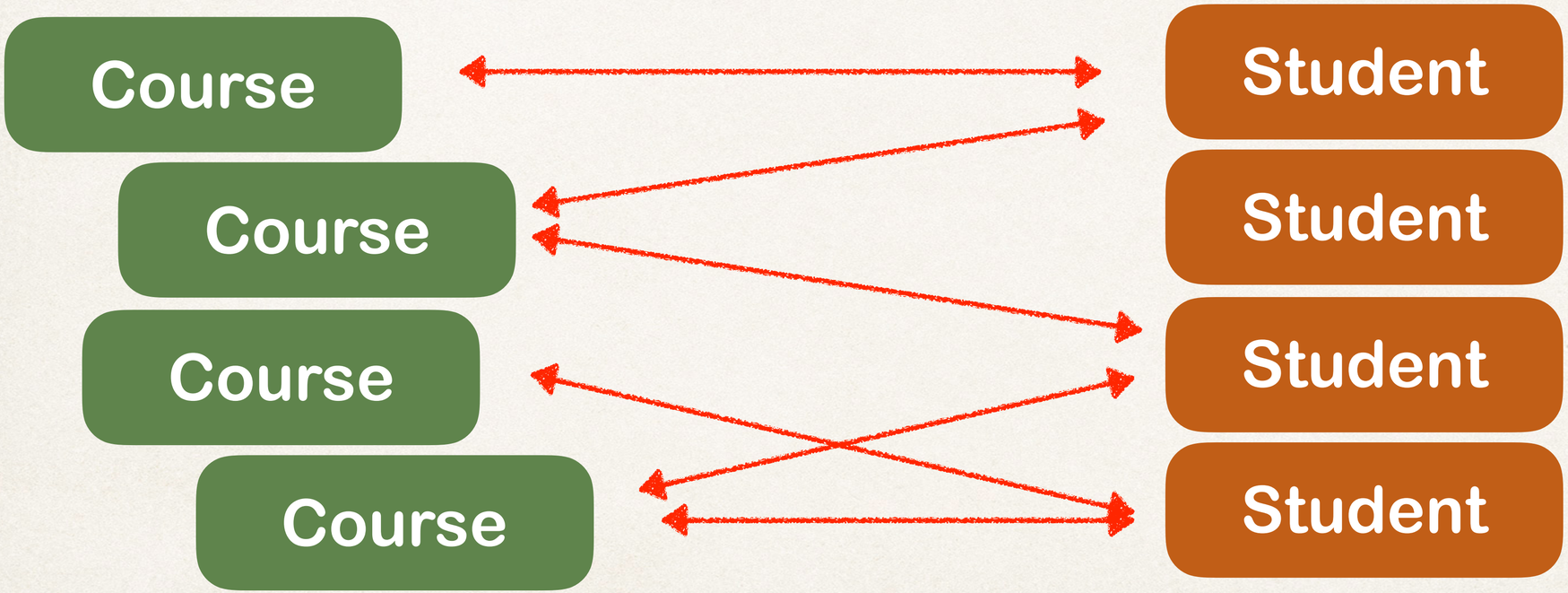
Step 4: Create Main App




```
public static void main(String[] args) {
        ...

        // get the course object
        int theId = 10;
        Course tempCourse = session.get(Course.class, theId);

        // print the course
        System.out.println("tempCourse: " + tempCourse);

        // print the associated students
        System.out.println("students: " + tempCourse.getStudents());
        ...
}
```

Real-World Project Requirement

- If you delete a course, DO NOT delete the students



Other features

- In the next set of videos, we'll add support for other features
- **Lazy Loading** of students and courses
- **Cascading** to handle cascading saves … but NOT deletes
  - If we delete a course, DO NOT delete students
  - If we delete a student, DO NOT delete courses