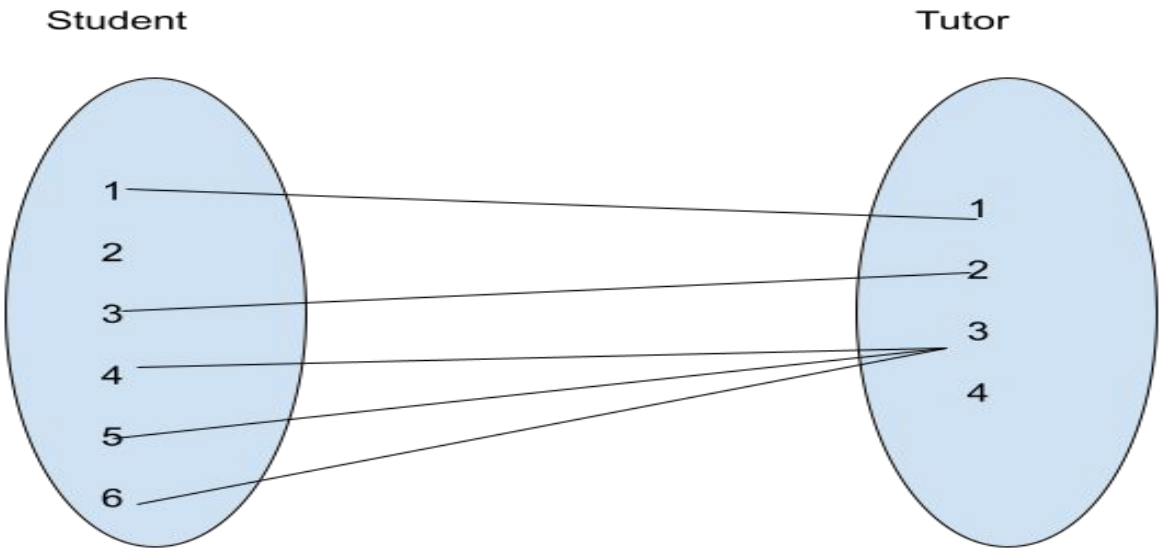
Hibernate
One_To_Many Relationship

Overview

In this slide, we cover:

- What is One to Many mapping/relation
- How to make a One to Many Relation (between two tables) using Hibernate
- Different type of collections (Set, Map, List)
- Unidirectional vs Bidirectional

One-To-Many Relationship



@OneToMany

To represent this relationship, the Tutor class will have a collection of Students:

@OneToMany

private Set<Student>teachingGroup;

Alternative options:

```
private Map<String,Student>teachingGroup;  Or
private List<Student>teachingGroup;
```

The many-to-one annotation in the Student class should be removed.

Database Impact

Table Structure

This implementation creates:

- student table
- tutor table
- tutor_student (link table containing foreign keys from both tables)

Alternatively, a foreign key in the student table can be used instead of a link table:

```
@OneToMany
@JoinColumn(name = "TUTOR_FK")
private Set<Student> teachingGroup;
```

This creates a tutor_fk column in the student table, removing the need for tutor_student

Methods for Managing the Relationship

Adding Students to a Tutor:

In the Tutor class:

```
public void addStudentToTeachingGroup(Student newStudent) {
    this.teachingGroup.add(newStudent);
}
```

Methods for Managing the Relationship

Retrieving Students in a Tutor's Group

In the Tutor class:

```
public Set<Student> getTeachingGroup() {

    set<Student>unmodifiable=Collections.unmodifiableSet(this.teachingGroup);

    return unmodifiable;

}
```

How to create tutors, students and add to the database

```
Tutor newTutor = new Tutor("ABC234", "Natalie Woodward", 387787);

Student student1 = new Student("Patrik Howard");

Student student2 = new Student("Marie Sani");

newTutor.addStudentToTeachingGroup(student1);

newTutor.addStudentToTeachingGroup(student2);

session.save(student1);

session.save(student2);

session.save(newTutor);
```

Unidirectional vs. Bidirectional

Unidirectional Relationship:

It is enough to have one of these annotations: one to many in one class or many to one in another class, because the code will work and the tables will be created correctly in both ways. This is called *unidirectional*.

In a unidirectional relationship, only one entity knows about the other.

Example:

```
@OneToMany
private Set<Student> teachingGroup;
```

Only tutor knows about the students.

Unidirectional

Pros:

- Simpler implementation
- Less coupling between entities

Cons:

- Harder navigation (requires extra queries to fetch related entities)

Bidirectional

Both entities are aware of each other.

Bidirectional association allows us to fetch details of dependent object from both side. In such case, we have the reference of two classes in each other.

Example:

```
@OneToMany(mappedBy = "tutor")
private List<Student> students;


@ManyToOne
private Tutor tutor;
```

Bidirectional

Pros:

- Easier navigation between entities
- More flexibility in queries

Cons:

- Increases complexity
- Risk of circular references

What is next?

In the next session, we will explore Many-To-Many relationships in Hibernate.