What you will learn in Topic 3

Azzeddine
RIGAT

Topic 3: Introduction to Hibernate

- Introduction to Hibernate
- Setting Up Hibernate Development Environment
- Hibernate Configuration with Annotations
- Hibernate CRUD Features Create, Read, Update and Delete
- Hibernate Advanced Mappings
- Hibernate Advanced Mappings - @OneToOne
- Hibernate Advanced Mappings - @OneToMany
- Hibernate Advanced Mappings - Eager vs Lazy Loading
- Hibernate Advanced Mappings - @OneToMany - Unidirectional
- Hibernate Advanced Mappings - @ManyToMany

Topic 3, Introduction to Hibernate

Azzeddine
RIGAT

Introduction to Hibernate

Topic 3, Introduction to Hibernate

Azzeddine
RIGAT

Hibernate Overview

Topics

1. What is Hibernate?
2. Benefits of Hibernate
3. Code Snippets

Topic 3, Introduction to Hibernate

Azzeddine
RIGAT

Hibernate Overview

What is Hibernate?

1. A framework for persisting / saving Java objects in a database
2. www.hibernate.org

Topic 3, Introduction to Hibernate

Azzeddine
RIGAT

Hibernate Overview

Benefits of Hibernate

1. Hibernate handles all of the low-level SQL
2. Minimizes the amount of JDBC code you have to develop
3. Hibernate provides the Object-to-Relational Mapping (ORM)

Topic 3, Introduction to Hibernate

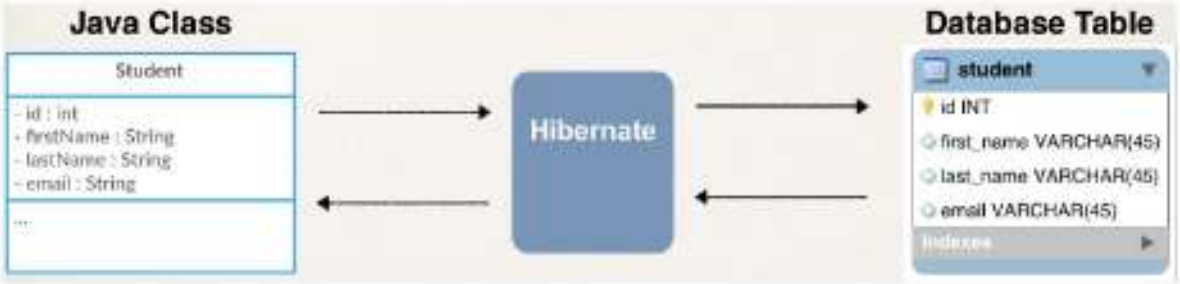
Azzeddine
RIGAT

Hibernate Overview

Benefits of Hibernate

Object-To-Relational Mapping (ORM)

The developer defines mapping between Java class and database table



Topic 3, Introduction to Hibernate

Azzeddine
RIGAT

Hibernate Overview

Saving a Java Object with Hibernate

```
// create Java object
Student theStudent = new Student("Xu", "Ming", "xu@javaweb.edu");

// save it to database
int theId = (Integer) session.save(theStudent);
```

Topic 3, Introduction to Hibernate

Azzeddine
RIGAT

Hibernate Overview

Retrieving a Java Object with Hibernate

```
// create Java object
Student theStudent = new Student("Xu", "Ming", "rigat@javaweb.edu");

// save it to database
int theId = (Integer) session.save(theStudent);

// now retrieve from database using the primary key
Student myStudent = session.get(Student.class, theId);
```

Topic 3, Introduction to Hibernate

Azzeddine
RIGAT

Hibernate Overview

Querying for Java Objects

```
Query query = session.createQuery("from Student");
List<Student> students= query.list();
```

Topic 3, Introduction to Hibernate

Azzeddine
RIGAT

Hibernate Overview

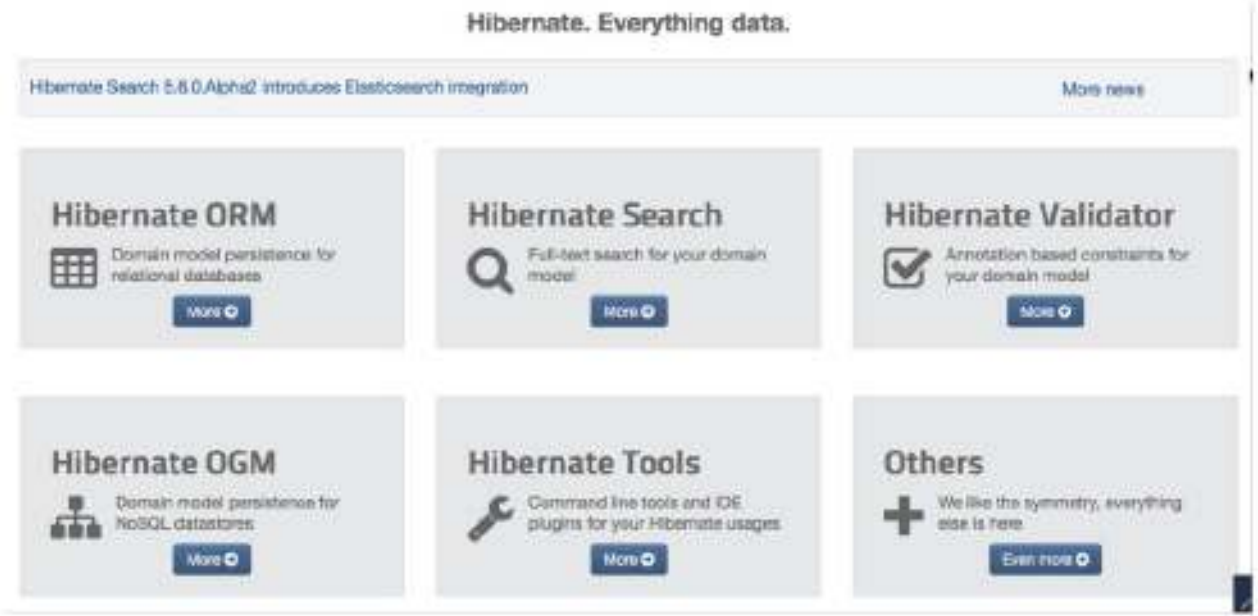
Hibernate CRUD Apps

- **C**reate objects
- **R**ead objects
- **U**pdate objects
- **D**elete objects

Topic 3, Introduction to Hibernate

Azzeddine
RIGAT

Hibernate is actually more than ORM!



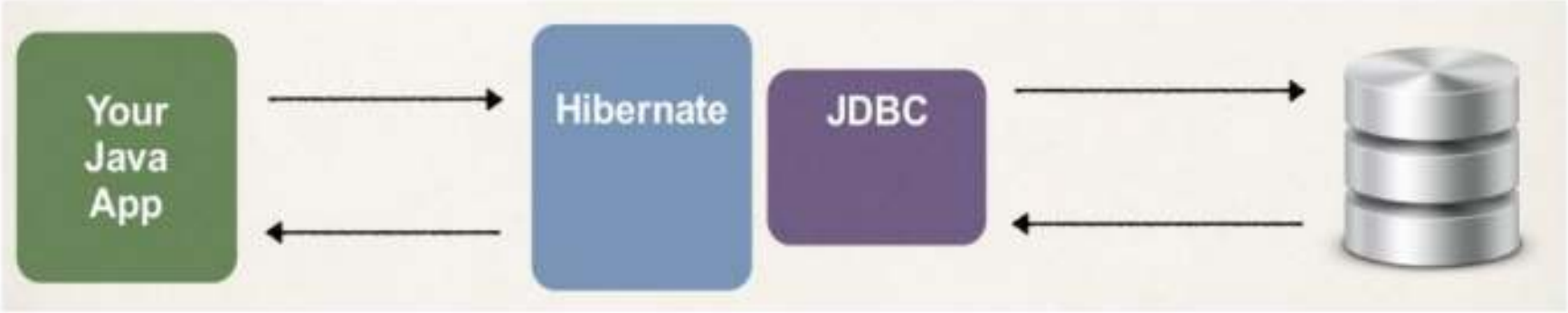
Topic 3, Introduction to Hibernate

Azzeddine
RIGAT

Hibernate and JDBC

How does Hibernate relate to JDBC?

Hibernate uses JDBC for all database communications



Topic 3, Setting Up Hibernate Development Environment

Azzeddine
RIGAT

Setting Up Hibernate Development Environment

Topic 3, Setting Up Hibernate Development Environment

Azzeddine
RIGAT

Required Software

To Build Hibernate Applications, you need the following:

1. Java Integrated Development Environment (IDE)
2. Database Server
3. Hibernate JAR files and JDBC Driver

Topic 3, Setting Up Hibernate Development Environment

Azzeddine
RIGAT

Install MySQL on MS Windows

1. Download MySQL
2. Install MySQL
3. Verify Installation

Topic 3, Setting Up Hibernate Development Environment

Azzeddine
RIGAT

Install MySQL on MS Windows

1. Download MySQL

   https://dev..mysql.com/downloads/installer/

Check **demo-1-Hibernate-sql-scripts-and-starter**

Topic 3, Setting Up Hibernate Development Environment

Azzeddine
RIGAT

Uninstall MySQL on MS Windows

To do so, you need to do it in four steps:

1.Run Command Prompt as Administrator and execute the following commands to stop then remove MySQL service

net stop MySQL80
sc delete MySQL80

2.Go to Control Panel>>Programs>>Uninstall program >> MySQL installer - Community >> uninstall

Topic 3, Setting Up Hibernate Development Environment

Azzeddine
RIGAT

Uninstall MySQL on MS Windows

3.Open Windows Explorer(that means any folder)>>View>>Options>> change folder and search options>>View>>tab and under "Hidden files and folders">> choose "**Show hidden files and folders**">>Apply. Now explore the following locations and delete the following folders:

C:\Program Files\MySQL
C:\Program Files(x86)\MySQL
C:\ProgramData\MySQL
C:\Users\[user-name]\AppData\Roaming\MySQL(very important, check all the users you have)

4.Restart you PC

Topic 3, Setting Up Hibernate Development Environment

Azzeddine
RIGAT

Setup Database scripts

Folder: sql-scripts

1. create-user.sql

2. student-tracker.sql

Topic 3, Setting Up Hibernate Development Environment

Azzeddine
RIGAT

Setup Database scripts

About:01-create-user.sql

1.Create a new MySQL user for our application
    1. user id: **hbstudent**
    2. password: **hbstudent**

Topic 3, Setting Up Hibernate Development Environment

Azzeddine
RIGAT

Setup Database scripts

About: 02-student-tracker.sql

1.Create a new database table: **student**

```
select * from hb_student_tracker.student;
```

Topic 3, Setting Up Hibernate Development Environment

Azzeddine
RIGAT

Setup Hibernate in Eclipse

To Do List

1. Create Eclipse Project
2. Download Hibernate Files
3. Download MySQL JDBC Driver
4. Add JAR files to Eclipse Project … Build Path

Topic 3, Hibernate Configuration with Annotations

Azzeddine
RIGAT

Hibernate Configuration with Annotations

Topic 3, Hibernate Configuration with Annotations

Azzeddine
RIGAT

Test JDBC Connection - Hibernate Dev Process

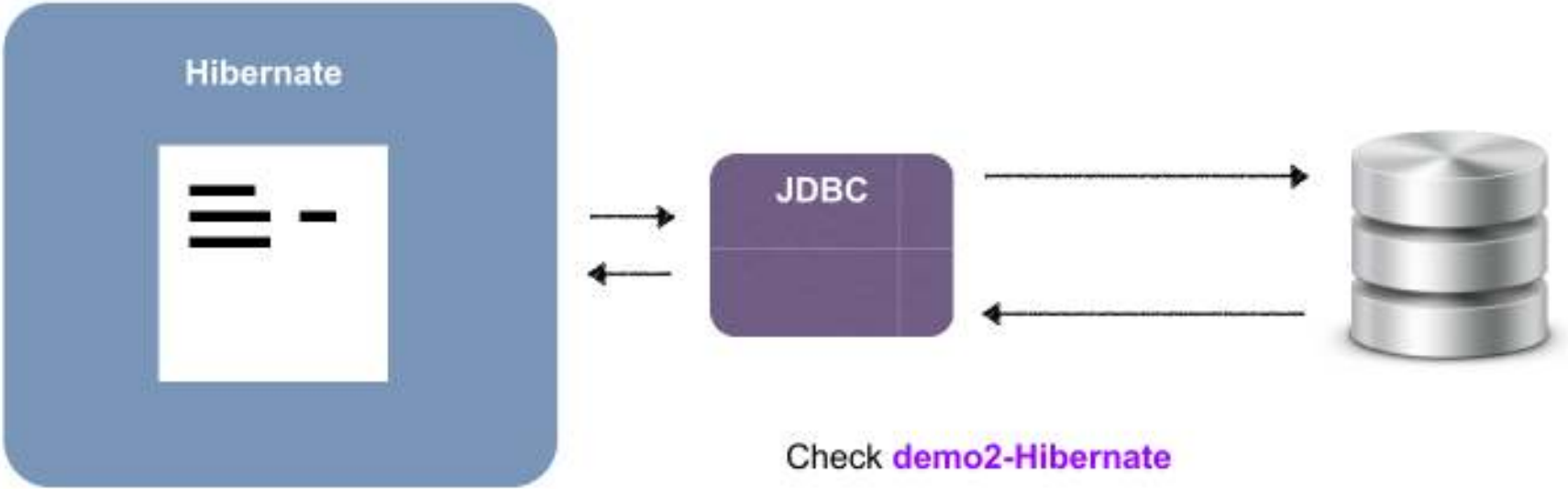
To Do List

1. Add Hibernate Configuration file
2. Annotate Java Class
3. Develop Java Code to perform database operations

Topic 3, Hibernate Configuration with Annotations

Azzeddine
RIGAT

Configuration File



Check **demo2-Hibernate**

Topic 3, Hibernate Configuration with Annotations

Azzeddine
RIGAT

Annotate Java Class

To Do List

1. Add Hibernate Configuration file

2. Annotate Java Class

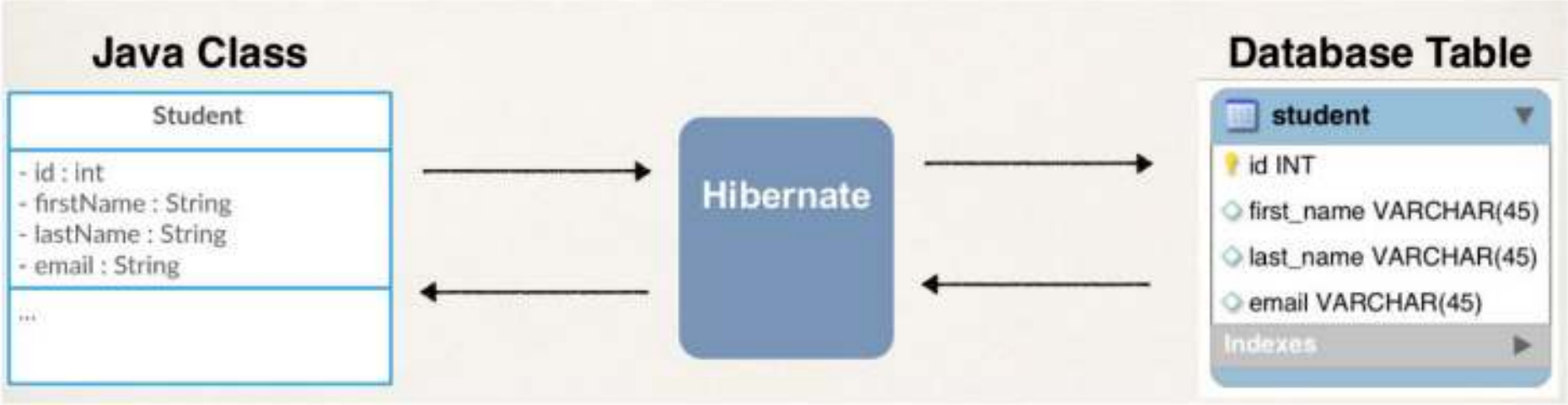
3. Develop Java Code to perform database operations

Topic 3, Hibernate Configuration with Annotations

Azzeddine
RIGAT

Entity Class Java class that is mapped to a database table

Object-to-Relational Mapping (ORM)



Topic 3, Hibernate Configuration with Annotations

Azzeddine
RIGAT

Annotate Java Class

Two Options for Mapping

1. Option 1: XML config file (legacy)

2. Option 2: Java Annotations (modern, preferred)

Topic 3, Hibernate Configuration with Annotations

Azzeddine
RIGAT

Annotate Java Class

Java Annotations

1. Step 1: Map class to database table

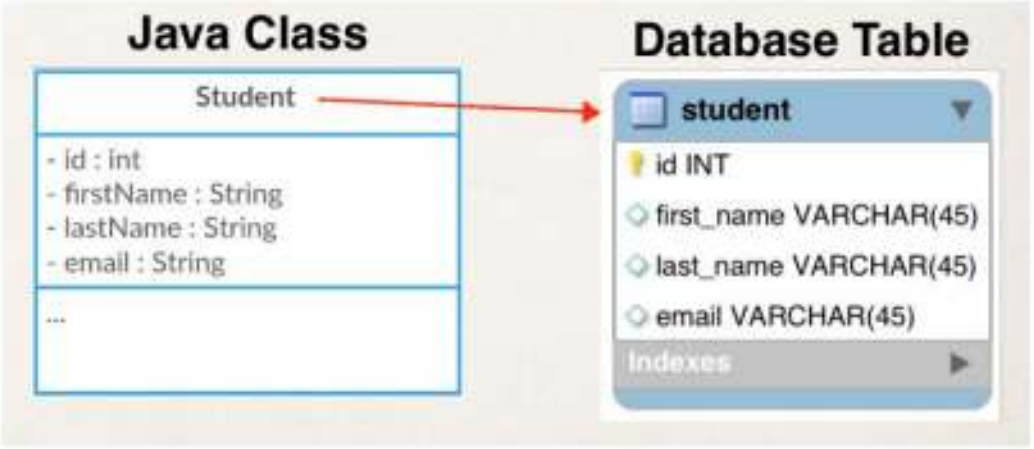
2. Step 2: Map fields to database columns

Topic 3, Hibernate Configuration with Annotations

Azzeddine
RIGAT

Step 1: Map class to database table

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

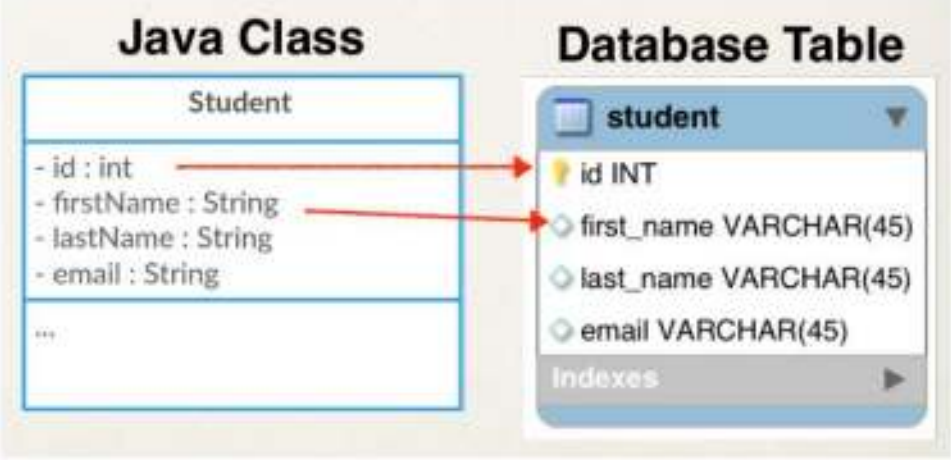


Topic 3, Hibernate Configuration with Annotations

Azzeddine
RIGAT

Step 2: Map fields to database columns

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

    @Id
    @Column(name="id")
    private int id;

    @Column(name="first_name")
    private String firstName;
    …
}
```



Check **demo3-Hibernate-Config**

Topic 3, Hibernate CRUD Features Create, Read, Update and Delete

Azzeddine
RIGAT

Hibernate CRUD Features Create, Read, Update and Delete

Topic 3, Hibernate CRUD Features Create, Read, Update and Delete

Azzeddine
RIGAT

Hibernate Dev Process - To Do List

1. Add Hibernate Configuration file
2. Annotate Java Class
3. Develop Java Code to perform database operations

Topic 3, Hibernate CRUD Features Create, Read, Update and Delete

Azzeddine
RIGAT

Two Key Players

Class	Description
SessionFactory	Reads the hibernate config file Creates Session objects Heavy-weight object Only create once in your app
Session	Wraps a JDBC connection Main object used to save/retrieve objects Short-lived object Retrieved from SessionFactory

Topic 3, Hibernate CRUD Features Create, Read, Update and Delete

Azzeddine
RIGAT

Java Code Setup

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

    SessionFactory factory = new Configuration()
                             .configure("hibernate.cfg.xml")
                             .addAnnotatedClass(Student.class)
                             .buildSessionFactory();

    Session session = factory.getCurrentSession();

    try {

        // now use the session object to save/retrieve Java objects

    } finally {
        factory.close();
    }

}
```

Topic 3, Hibernate CRUD Features Create, Read, Update and Delete

Azzeddine
RIGAT

Save a Java Object

```
try {

    // create a student object
    Student tempStudent = new Student("Paul", "Wall", "paul@luv2code.com");

    // start transaction
    session.beginTransaction();

    // save the student
    session.save(tempStudent);

    // commit the transaction
    session.getTransaction().commit();

} finally {
    factory.close();
}
```

Check **demo4-Hibernate-CreateStudent**

Topic 3, Hibernate CRUD Features Create, Read, Update and Delete

Azzeddine
RIGAT

Hibernate and Primary Key

Primary Key

1. Uniquely identifies each row in a table

2. Must be a unique value

3. Cannot contain NULL values

Topic 3, Hibernate CRUD Features Create, Read, Update and Delete

Azzeddine
RIGAT

MySQL - Auto Increment

```
CREATE TABLE student (

id int(11) NOT NULL AUTO_INCREMENT,
first_name varchar(45) DEFAULT NULL,  last_name varchar(45) DEFAULT NULL,  email varchar(45) DEFAULT
NULL,  PRIMARY KEY (id)

)
```

Topic 3, Hibernate CRUD Features Create, Read, Update and Delete

Azzeddine
RIGAT

Hibernate Identity - Primary Key

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

    @Id
    @Column(name="id")
    private int id;

    ...
}
```

Topic 3, Hibernate CRUD Features Create, Read, Update and Delete

Azzeddine
RIGAT

Hibernate Identity - Primary Key

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

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

    ...
}
```

Check **demo5-Hibernate-AddMoreStudents**

Topic 3, Hibernate CRUD Features Create, Read, Update and Delete

Azzeddine
RIGAT

ID Generation Strategies

Name	Description
GenerationType.AUTO	Pick an appropriate strategy for the particular database
GenerationType.IDENTITY	Assign primary keys using database identity column
GenerationType.SEQUENCE	Assign primary keys using a database sequence
GenerationType.TABLE	Assign primary keys using an underlying database table to ensure uniqueness

Topic 3, Hibernate CRUD Features Create, Read, Update and Delete

Azzeddine
RIGAT

Change the start point of your table index

```
alter table hb_student_tracker.student AUTO_INCREMENT=3000;
```

Delete all rows and set the index to 1

```
truncate hb_student_tracker.student;
```

Topic 3, Hibernate CRUD Features Create, Read, Update and Delete

Azzeddine
RIGAT

Tricks

- You can define your own CUSTOM generation strategy :-)
- Create implementation of **org.hibernate.id.IdentifierGenerator**
- Override the method: **public Serializable generate(…)**

- **Always generate unique value**
- **Work in high-volume, multi-volume, multi-threaded environment**
- **If using server clusters, always generate unique value.**

Topic 3, Hibernate CRUD Features Create, Read, Update and Delete

Azzeddine
RIGAT

Retrieving a Java Object with Hibernate

```
// create Java object
Student theStudent = new Student("Mary", "Public", "mary@javaweb.com");

// save it to database
session.save(theStudent);
.....

//now retrieve/read from database usign the primary key
Student = myStudent = session.get(Student.class, theStudent.getId());
```

Check **demo6-Hibernate-read**

Topic 3, Hibernate CRUD Features Create, Read, Update and Delete

Azzeddine
RIGAT

Querying Objects

- Query language for retrieving objects
- Similar in nature to SQL
- **where, like, order by, join, in, etc…**

```
List<Student> theStudents = session.createQuery("from Student").getResultList();
```

Topic 3, Hibernate CRUD Features Create, Read, Update and Delete

Azzeddine
RIGAT

Querying Objects

```
List<Student> theStudents =  session
.createQuery("from Student s where s.lastName='Doe'")
.getResultList();

List<Student> theStudents =  session
.createQuery("from Student s where s.lastName='Doe'"
+ " OR s.firstName='Daffy'")
.getResultList();

List<Student> theStudents =  session
.createQuery("from Student s where"
+ " s.email LIKE '%javaweb.edu'")
.getResultList();
```

Check **demo7-Hibernate-query**

Topic 3, Hibernate CRUD Features Create, Read, Update and Delete

Azzeddine
RIGAT

Updating Object(s)

```
int studentId = 1;

Student myStudent = session.get(Student.class, studentId);

// update first name to "Scooby"
myStudent.setFirstName("Scooby");

// commit the transaction
session.getTransaction().commit();

// update all the students emails
session.createQuery("update Student set email='foo@gmail.com'")
.executeUpdate();
```

Check **demo8-Hibernate-update**

Topic 3, Hibernate CRUD Features Create, Read, Update and Delete

Azzeddine
RIGAT

Delete Object(s)

```
int studentId =1 ;
student myStudent = session.get(Studetn.class, studentId);

// delete the student
session.delete(myStudent)

// commit the transaction
session.getTransaction().commit();


session.createQuery("delete from Student where id=2")
       .executeUpdate();
```

Check **demo9-Hibernate-delete**

Topic 3, Hibernate Advanced Mappings

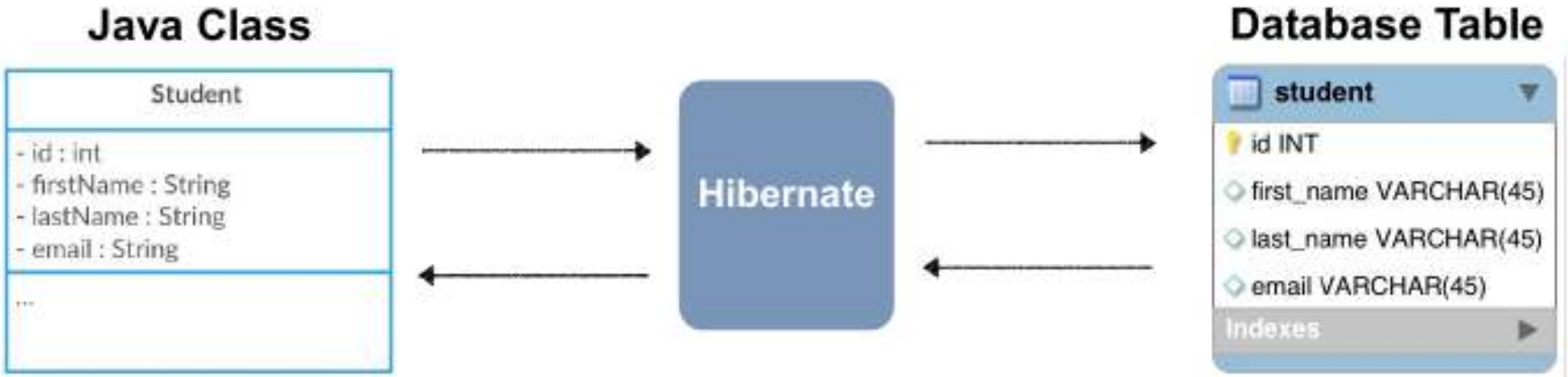
Azzeddine
RIGAT

Hibernate Advanced Mappings

Topic 3, Hibernate Advanced Mappings

Azzeddine
RIGAT

Basic Mapping



Topic 3, Hibernate Advanced Mappings

Azzeddine
RIGAT

Advanced Mappings

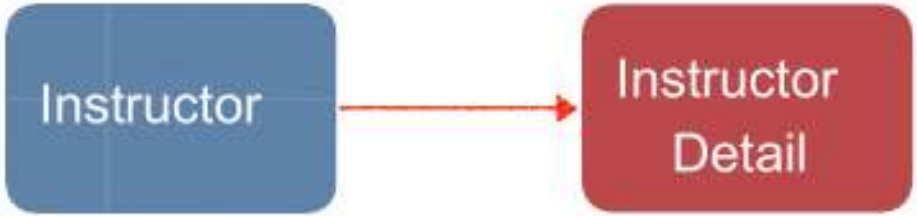
- In the database, you most likely will have
  - Multiple Tables
  - Relationships between Tables
- Need to model this with Hibernate

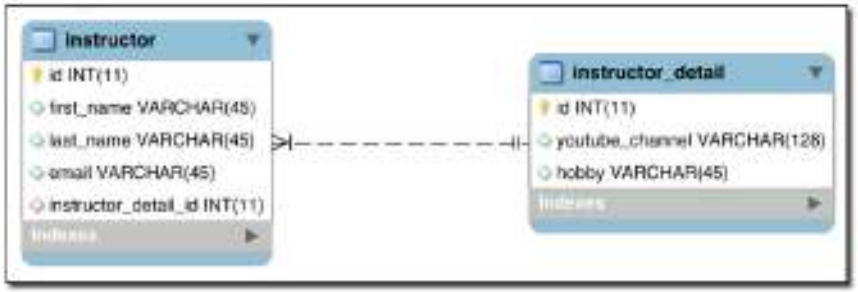
- One-to-One
- One-to-Many, Many-to-One
- Many-to-Many

Topic 3, Hibernate Advanced Mappings

Azzeddine
RIGAT

One-to-One Mapping



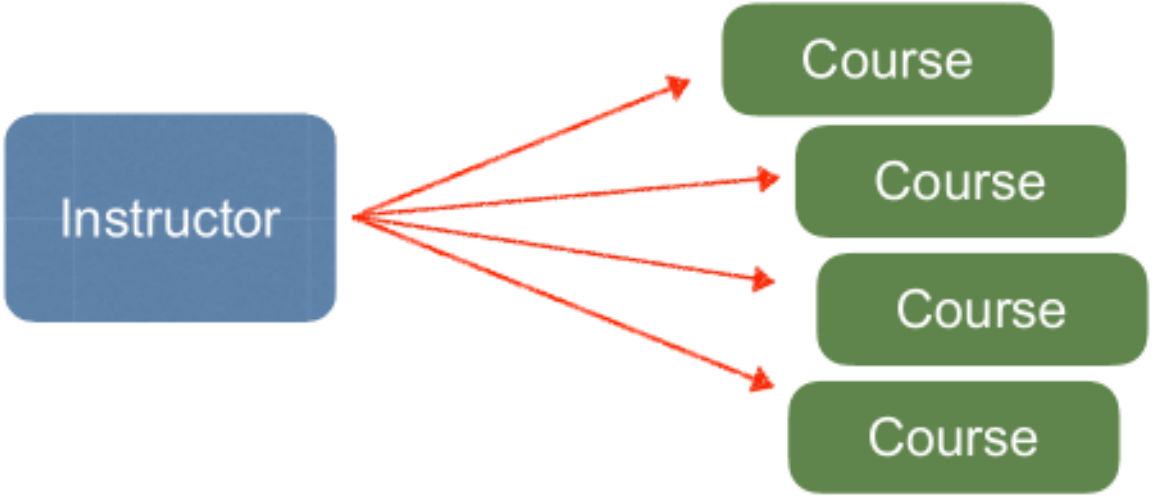


Topic 3, Hibernate Advanced Mappings

Azzeddine
RIGAT

One-to-Many Mapping

An instructor can have many courses

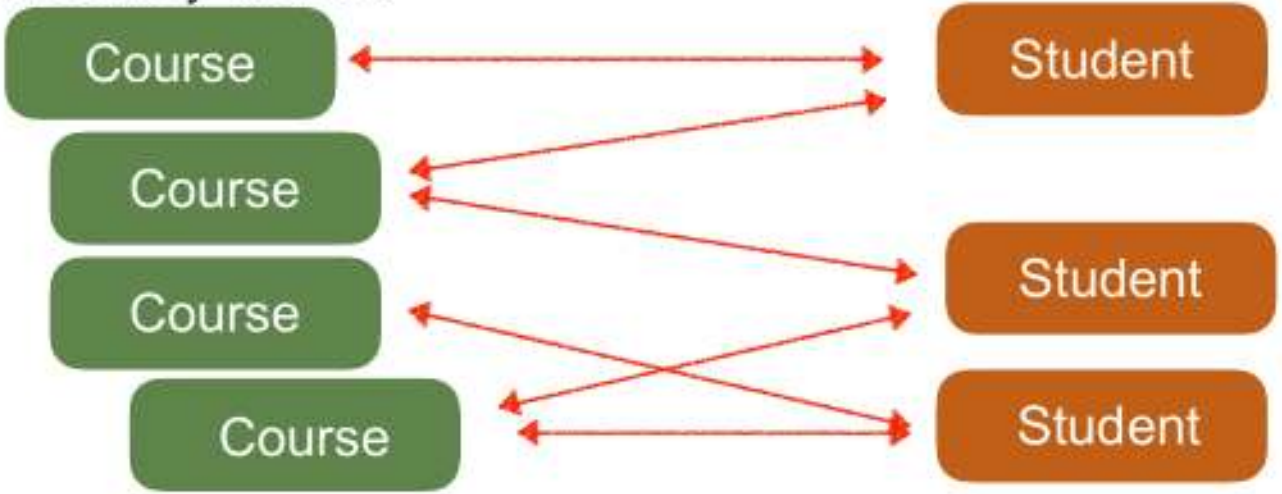


Topic 3, Hibernate Advanced Mappings

Azzeddine
RIGAT

Many-to-Many Mapping

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



Topic 3, Hibernate Advanced Mappings

Azzeddine
RIGAT

Important Database Concepts

- Primary key and foreign key

- Cascade

Topic 3, Hibernate Advanced Mappings

Azzeddine
RIGAT

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

Topic 3, Hibernate Advanced Mappings

Azzeddine
RIGAT

Foreign Key Example

Foreign key column

Table: instructor

id	first_name	last_name	instructor_detail_id
1	Chad	Darby	100
2	Madhu	Patel	200

Table: instructor_detail

id	youtube_channel	hobby
100	ww.youtube.com/Javaweb	Luv 2 Code!!!
200	www.youtube.com	Guitar

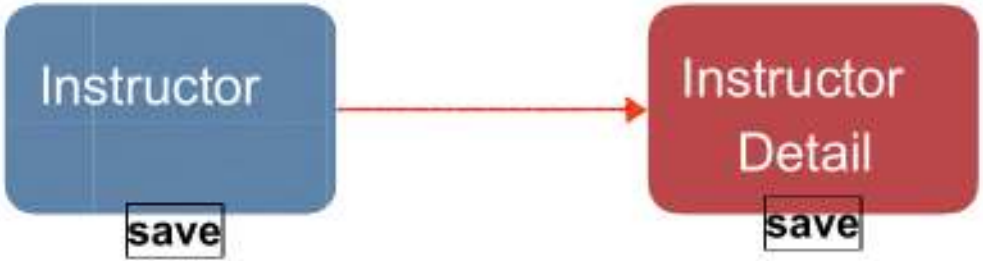
Topic 3, Hibernate Advanced Mappings

Azzeddine
RIGAT

Cascade

- You can cascade operations

- Apply the same operation to related entities



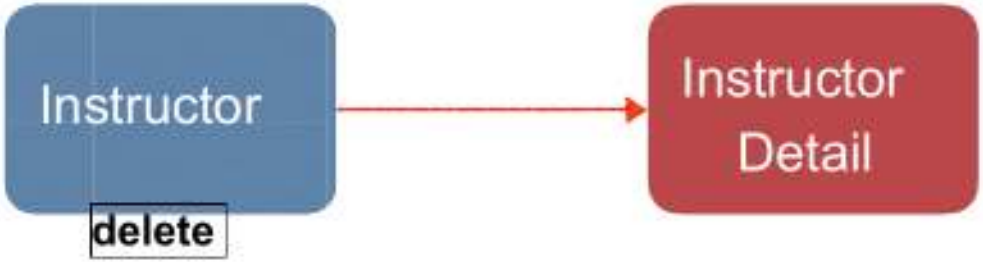
Topic 3, Hibernate Advanced Mappings

Azzeddine
RIGAT

Cascade

If we delete an instructor, we should also delete their instructor_detail

- This is known as “CASCADE DELETE”



Topic 3, Hibernate Advanced Mappings

Azzeddine
RIGAT

Cascade Delete

Foreign key column

Table: instructor

id	first_name	last_name	instructor_detail_id
~~1~~	~~Chad~~	~~Darby~~	~~100~~
2	Madhu	Patel	200

Table: instructor_detail

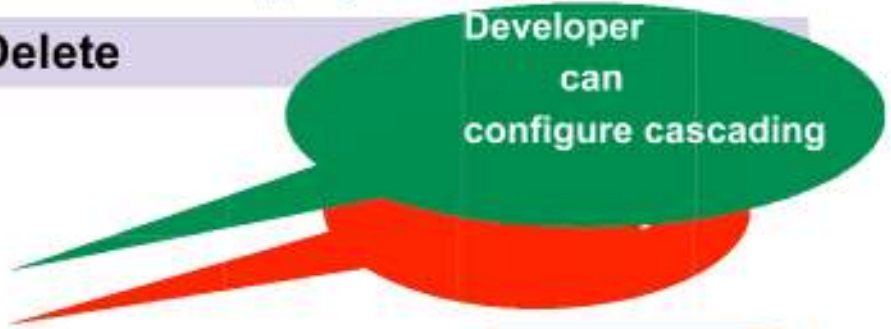
id	youtube_channel	hobby
~~100~~	~~www. Javaweb~~ /youtube	~~Javaweb~~
200	www.youtube.com	Guitar

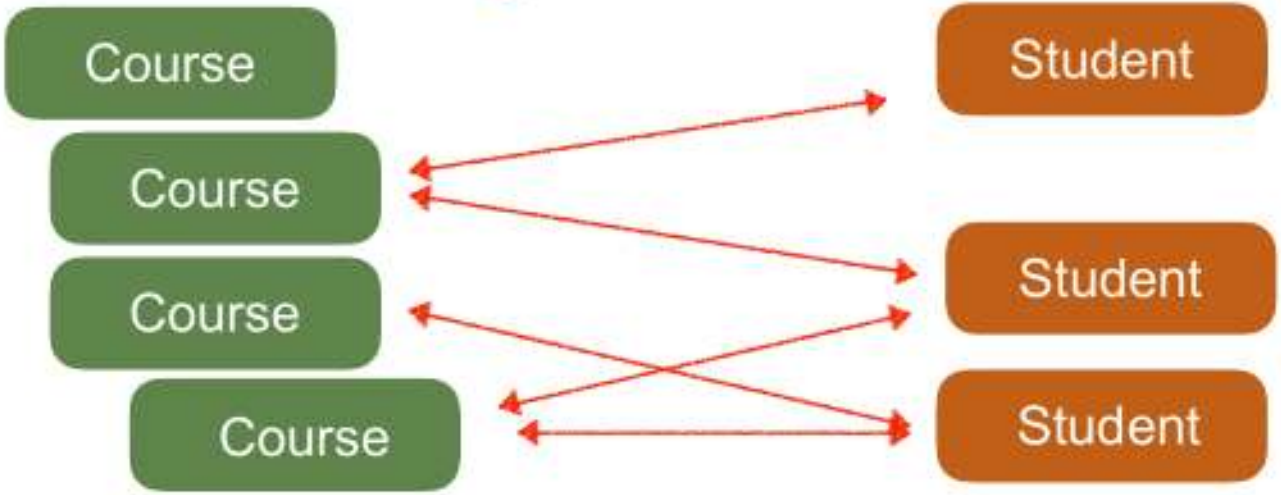
Topic 3, Hibernate Advanced Mappings

Azzeddine
RIGAT

Cascade Delete

- Cascade delete depends on the use case
- Should we do cascade delete here???





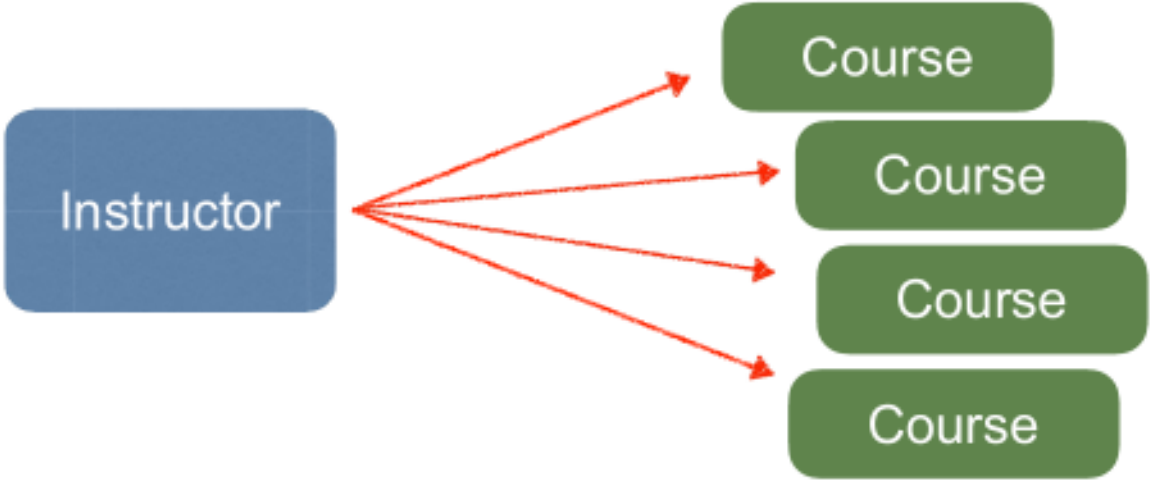
Topic 3, Hibernate Advanced Mappings

Azzeddine
RIGAT

Fetch Types: Eager vs Lazy Loading

When we fetch / retrieve data, should we retrieve EVERYTHING?

- **Eager** will retrieve everything
- **Lazy** will retrieve on request



Topic 3, Hibernate Advanced Mappings

Azzeddine
RIGAT

Uni-Directional

Topic 3, Hibernate Advanced Mappings

Azzeddine
RIGAT

Bi-Directional

Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

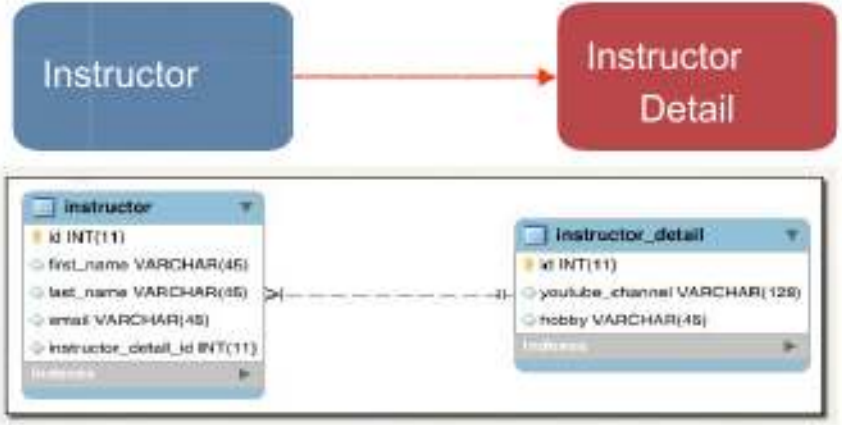
Hibernate Advanced Mappings - @OneToOne

Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

One-to-One Mapping

- An instructor can have an “instructor detail” entity
- Similar to an “instructor profile”



Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

Uni-Directional

Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

Development Process: One-to-One

1. Prep Work - Define database tables
2. Create Instructor Detail class
3. Create Instructor class
4. Create Main App

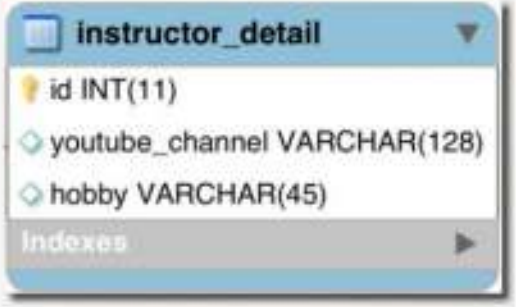
Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

table: instructor_detail

File: create-db.sql

```
CREATE TABLE `instructor_detail` (

    `id` int(11) NOT NULL AUTO_INCREMENT,
    `youtube_channel` varchar(128) DEFAULT NULL,
    `hobby` varchar(45) DEFAULT NULL,

);
```



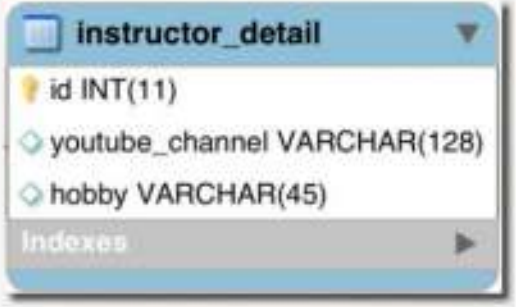
Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

table: instructor_detail

File: create-db.sql

```
CREATE TABLE `instructor_detail` (

    `id` int(11) NOT NULL AUTO_INCREMENT,
    `youtube_channel` varchar(128) DEFAULT NULL,
    `hobby` varchar(45) DEFAULT NULL,
     PRIMARY KEY (`id`)

);
```



Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

table: instructor_detail

File: create-db.sql

```
CREATE TABLE `instructor` (

    `id` int(11) NOT NULL AUTO_INCREMENT,
    `first_name` varchar(45) DEFAULT NULL,
    `last_name` varchar(45) DEFAULT NULL,
    `email` varchar(45) DEFAULT NULL,
    `instructor_detail_id` int(11) DEFAULT NULL,

    PRIMARY KEY (`id`)
    ...
);
```

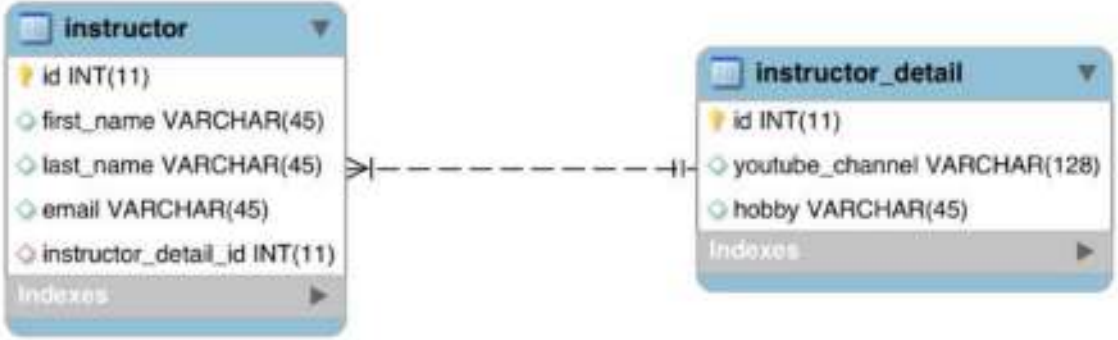
Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

Foreign Key

1. Link tables together

2. A field in one table that refers to primary key in another table



Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

Foreign Key Example

Foreign key column

Table: instructor

id	first_name	last_name	instructor_detail_id
1	Chad	Darby	100
2	Madhu	Patel	200

Table: instructor_detail

id	youtube_channel	hobby
100	ww.youtube.com/Javaweb	Luv 2 Code!!!
200	www.youtube.com	Guitar

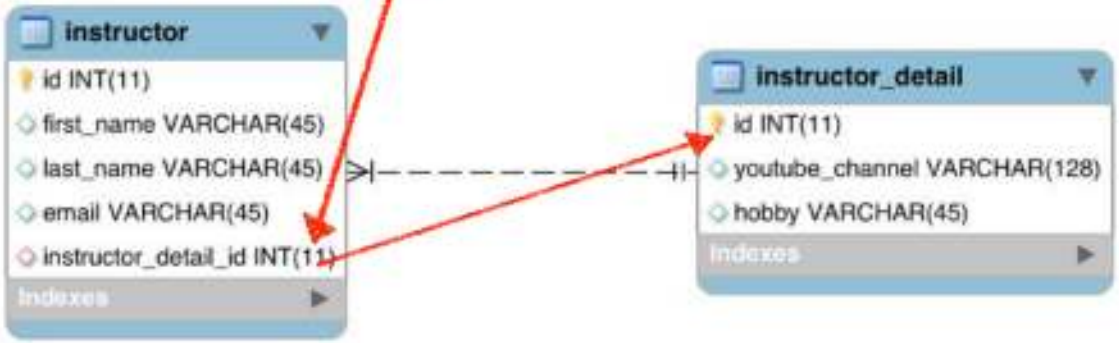
Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

Defining Foreign Key

File: create-db.sql

```
CREATE TABLE `instructor` (
    ...
    CONSTRAINT `FK_DETAIL` FOREIGN KEY (`instructor_detail_id`)
    REFERENCES `instructor_detail` (`id`)
);
```



Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

More on Foreign Key

- Main purpose is to preserve relationship between tables
- Referential Integrity
    - Prevents operations that would destroy relationship
    - Ensures only valid data is inserted into the foreign key column
        - Can only contain valid reference to primary key in other table

Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

More on Foreign Key

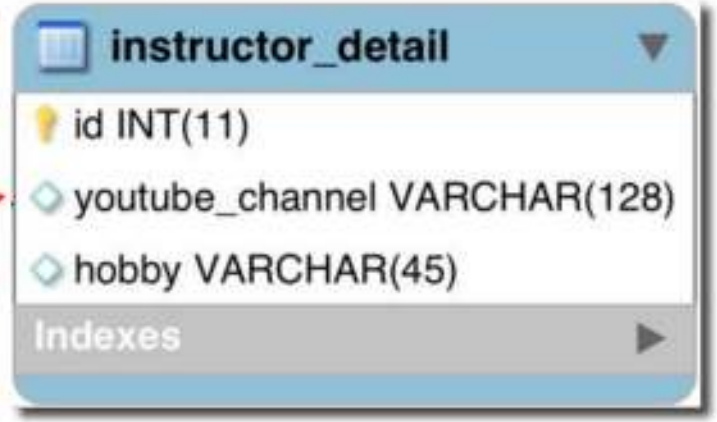
1. Prep Work - Define database tables
2. Create InstructorDetail class
3. Create Instructor class
4. Create Main App

Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

Step 2: Create InstructorDetail class

```
@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;
    // constructors
    // getters / setters
```



Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

Step 3: Create Instructor class

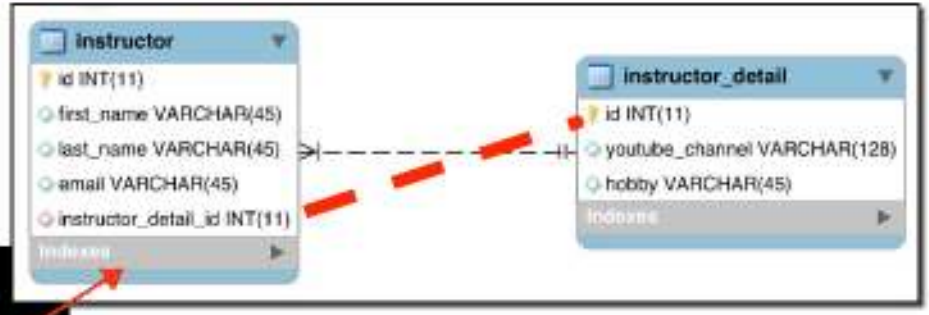
```
@Entity  @Table(name="instructor")
public class Instructor {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="id")
    private int id;
    @Column(name="first_name")
    private String firstName;
    @Column(name="last_name")
    private String lastName;
    @Column(name="email")
    private String email;
    …
    // constructors, getters / setters
}
```

Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

Step 3: Create Instructor class

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

   @OneToOne
   @JoinColumn(name="instructor_detail_id")
   private InstructorDetail instructorDetail;

...
// constructors, getters / setters
}
```



Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

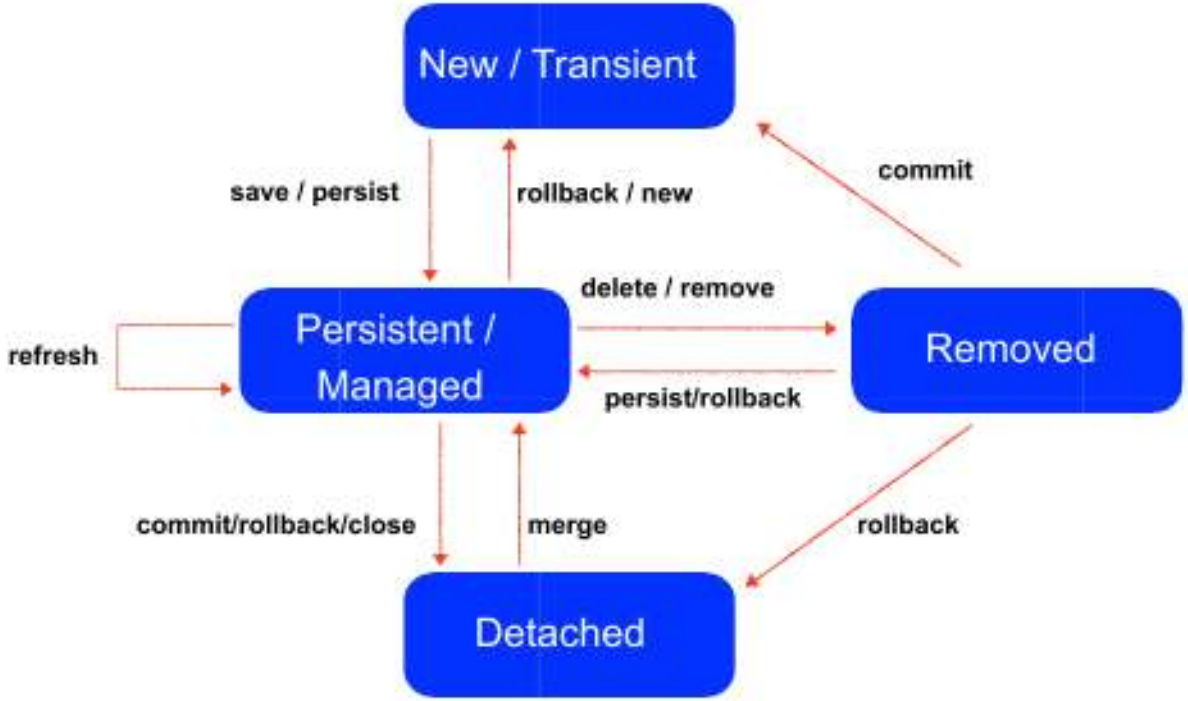
Entity Lifecycle

Operations	Description
Detach	If entity is detached, it is not associated with a Hibernate session
Merge	If instance is detached from session, then merge will reattach to session
Persist	Transitions new instances to managed state. Next flush / commit will save in db.
Remove	Transitions managed entity to be removed. Next flush / commit will delete from db.
Refresh	Reload / synch object with data from db. Prevents stale data

Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

Entity Lifecycle - session method calls

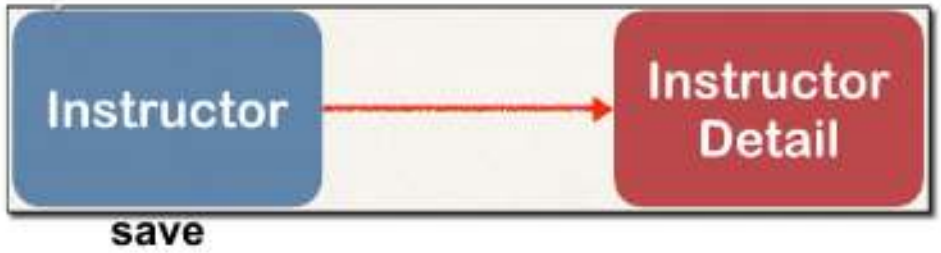


Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

Cascade

- Recall: You can cascade operations
- Apply the same operation to related entities



Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

Cascade Delete

Foreign key column

Table: instructor

id	first_name	last_name	instructor_detail_id
1	Chad	Darby	100
2	Madhu	Patel	200

Table: instructor_detail

id	youtube_channel	hobby
100	ww.youtube.com/Javaweb	Luv 2 Code!!!
200	www.youtube.com	Guitar

Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

@OneToOne - Cascade Types

Cascade type	Description
REMOVE	If entity is removed / deleted, related entity will also be deleted
PERSIST	If entity is persisted / saved, related entity will also be persisted
REFRESH	If entity is refreshed, related entity will also be refreshed
DETACH	If entity is detached (not associated w/ session), then related entity will also be detached
MERGE	If entity is merged, then related entity will also be merged
ALL	All of above cascade types

Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

Configure Cascade Type

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

    ...

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

    ...
    // constructors, getters / setters
}
```

By default, no operations are cascaded.

Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

Configure Multiple Cascade Types

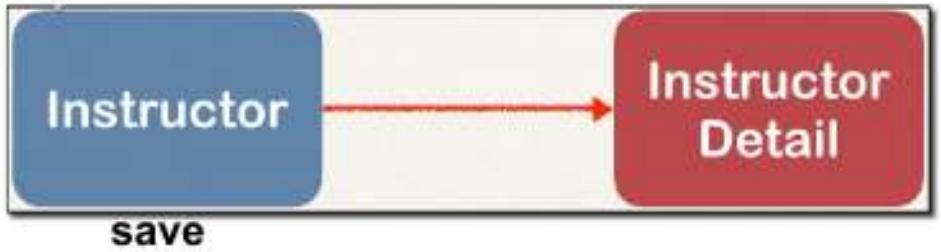
```
@OneToOne(cascade={CascadeType.DETACH,
                   CascadeType.MERGE,
                   CascadeType.PERSIST,
                   CascadeType.REFRESH,
                   CascadeType.REMOVE})
```

Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

Step 4: Create Main App

```
public static void main(String[] args) {
    // create the objects
    Instructor tempInstructor = new Instructor("Chad", "Darby", "darby@javaweb.edu");
    InstructorDetail tempInstructorDetail =
    new InstructorDetail("http://www.javaweb.edu/youtube", "Java Web 4 coding!!!");

    // associate the objects
    tempInstructor.setInstructorDetail(tempInstructorDetail);

    // start a transaction
    session.beginTransaction();
    session.save(tempInstructor);

    // commit transaction
    session.getTransaction().commit();
}
```



Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

Development Process: One-to-One

1. Prep Work

-

2. Create InstructorDetail class

3. Create Instructor class

4. Create Main App

Try to do **demo-10-Hibernate-one-to-one**

Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

New Use Case

- If we load an InstructorDetail,
- Then we'd like to get the associated Instructor.
- Can't do this with current uni-directional relationship :-(

Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

Bi-Directional relationship is the solution

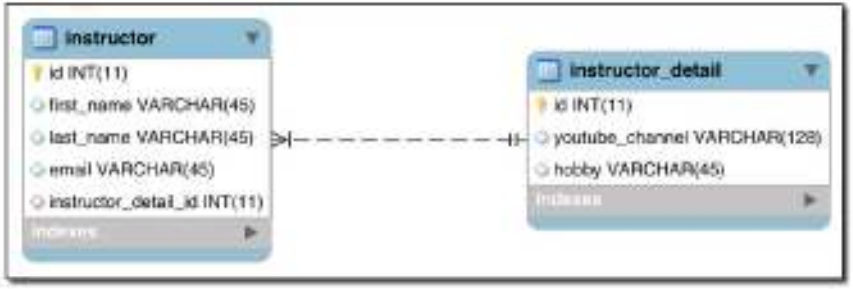
- We can start with InstructorDetail and make it back to the Instructor

Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

Bi-Directional - The Good News

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



Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

Development Process: One-to-One (Bi-Directional)

1. Make updates to InstructorDetail class:

    a. Add new field to reference Instructor

    b. Add getter/setter methods for Instructor

    c. Add @OneToOne annotation

2. Create Main App

Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

Step 1.a: Add new field to reference Instructor

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

    private Instructor instructor;

}
```

Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

Step 1.b: Add getter/setter methods Instructor

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

    public Instructor getInstructor() {
        return instructor;
    }

    public void setInstructor(Instructor instructor) {
        this.instructor = instructor;
    }
    ...

}
```

Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

Step 1.c: Add @OneToOne annotation

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

    ...
    @OneToOne(mappedBy="instructorDetail")
    private Instructor instructor;

    public Instructor getInstructor() {
        return instructor;
    }

    public void setInstructor(Instructor instructor) {
        this.instructor = instructor;
    }
    ...
}
```

Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

More on mappedBy

```
public class InstructorDetail {
    ...

    @OneToOne(mappedBy="instructorDetail")
    private Instructor instructor;
```

mappedBy tells Hibernate

- Look at the instructorDetail property in the Instructor class
- Use information from the Instructor class @JoinColumn To help find associated instructor

```
public class Instructor {
    ...
    @OneToOne(cascade=CascadeType.ALL)
    @JoinColumn(name="instructor_detail_id")
    private InstructorDetail instructorDetail;
```

Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

Add support for Cascadi

Cascade all operations

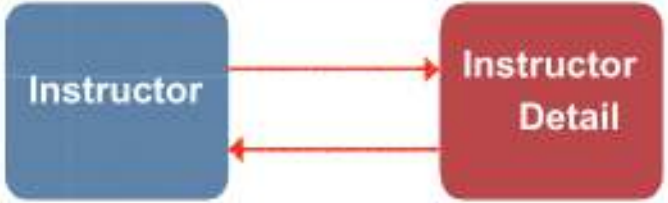
```
@Entity
@Table(name="instructor_detail")
public class InstructorDetail {
    …
    @OneToOne(mappedBy="instructorDetail", cascade=CascadeType.ALL)
    private Instructor instructor;
    public Instructor getInstructor() {
        return instructor;
    }
    public void setInstructor(Instructor instructor) {
        this.instructor = instructor;
    }
    …

}
```

Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

Step 2: Create Main App

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

    // get the instructor detail object
    int theId = 1;
    InstructorDetail tempInstructorDetail =
session.get(InstructorDetail.class, theId);

    // print detail
    System.out.println("tempInstructorDetail: " + tempInstructorDetail);

    // print the associated instructor
    System.out.println("the associated instructor: "
            + tempInstructorDetail.getInstructor());

    …
}
```



Topic 3, Hibernate Advanced Mappings - @OneToOne

Azzeddine
RIGAT

Assignment 17

Make an implementation of @OneToOne in the bidirectional way using the sides code

Deadline: 2019-November-21

Topic 3, Hibernate Advanced Mappings - @OneToMany

Azzeddine
RIGAT

Hibernate Advanced Mappings - @OneToMany

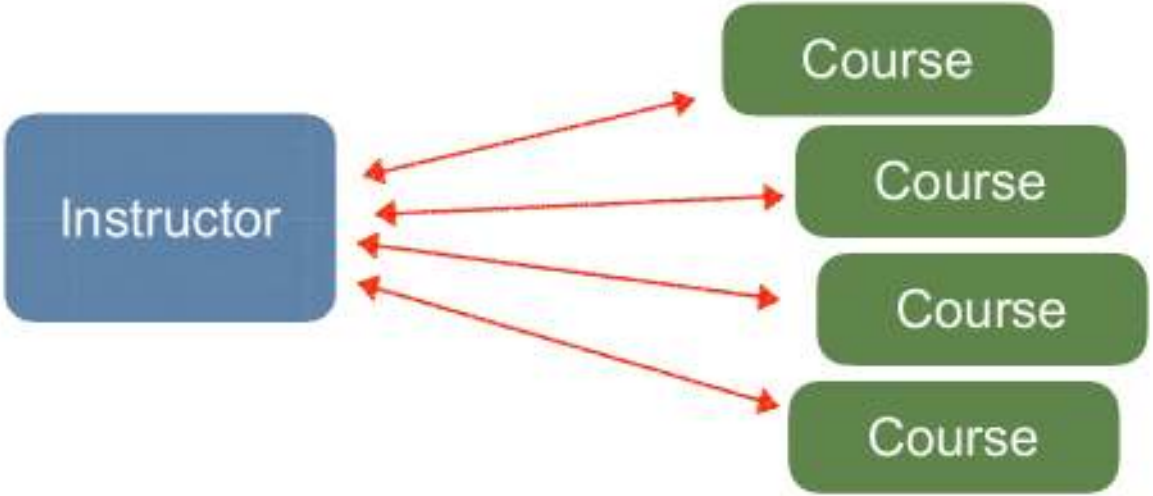
Topic 3, Hibernate Advanced Mappings - @OneToMany

Azzeddine
RIGAT

Development Process

- An instructor can have many courses

  a. Bi-directional



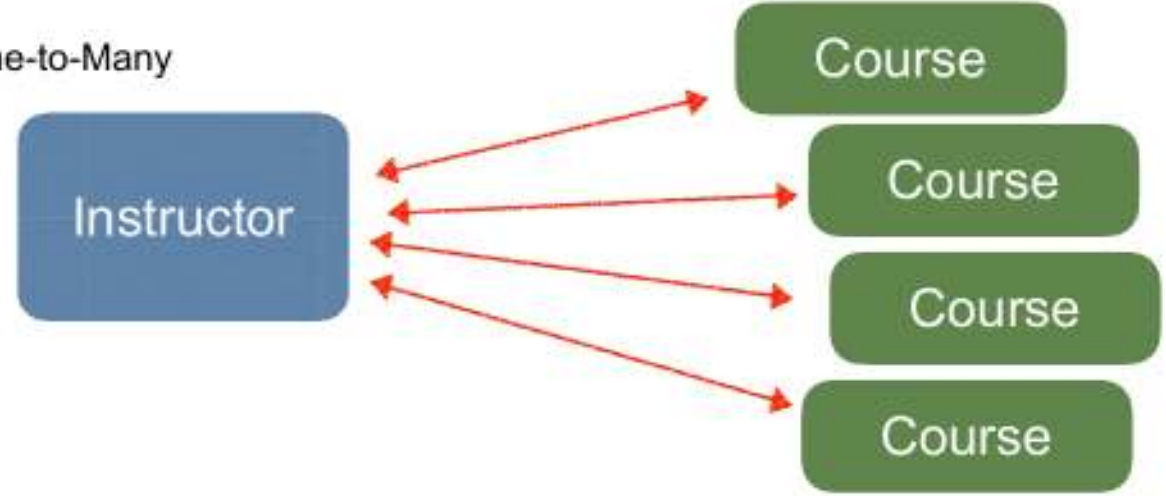
Topic 3, Hibernate Advanced Mappings - @OneToMany

Azzeddine
RIGAT

Development Process

- Many courses can have one instructor

  a. Inverse / opposite of One-to-Many

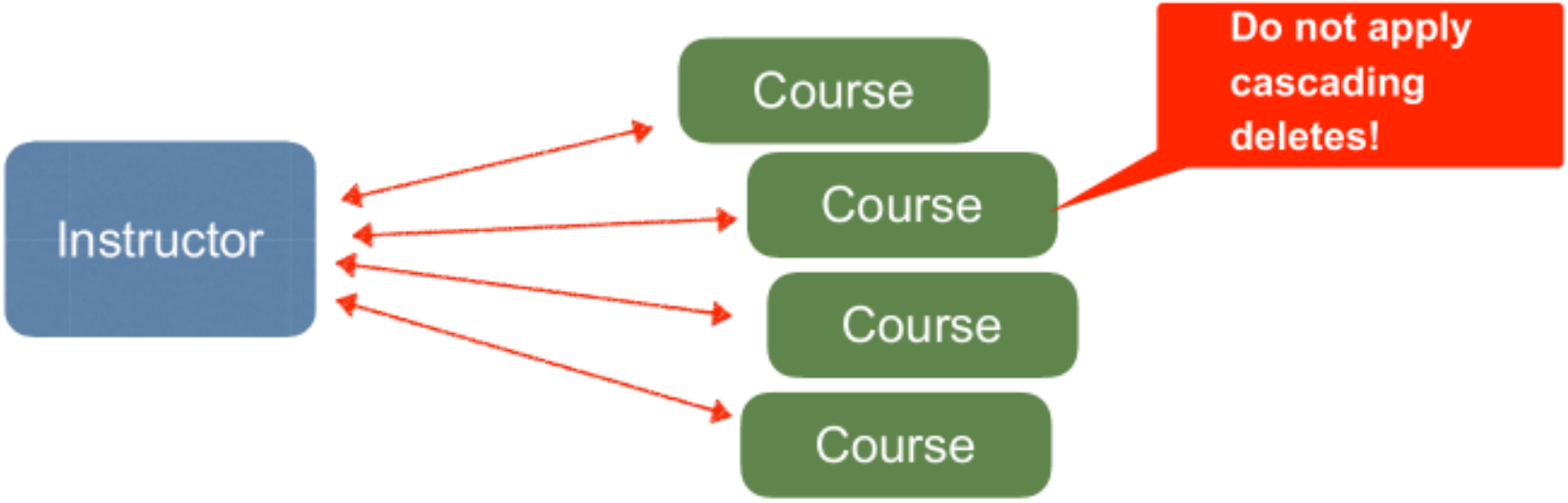


Topic 3, Hibernate Advanced Mappings - @OneToMany

Azzeddine
RIGAT

Real-World Project Requirement

- If you delete an instructor, DO NOT delete the courses
- If you delete a course, DO NOT delete the instructor



Topic 3, Hibernate Advanced Mappings - @OneToMany

Azzeddine
RIGAT

Development Process: One-to-Many

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

Topic 3, Hibernate Advanced Mappings - @OneToMany

Azzeddine
RIGAT

table: course

File: create-db.sql

```
CREATE TABLE `course` (
    `id` int(11) NOT NULL AUTO_INCREMENT,
    `title` varchar(128) DEFAULT NULL,
    `instructor_id` int(11) DEFAULT NULL,

    PRIMARY KEY (`id`),

    UNIQUE KEY `TITLE_UNIQUE` (`title`),
    ...
);
```

Prevent duplicate course

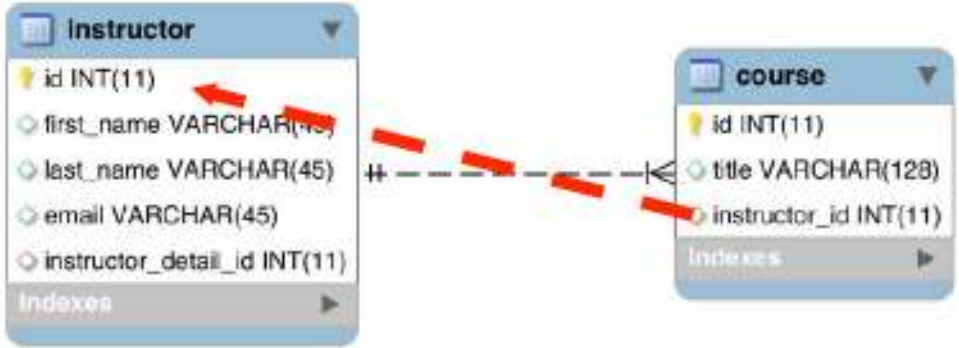
Topic 3, Hibernate Advanced Mappings - @OneToMany

Azzeddine
RIGAT

table: course - foreign key

File: create-db.sql

```
CREATE TABLE `course` (
    ...
    KEY `FK_INSTRUCTOR_idx` (`instructor_id`),
    CONSTRAINT `FK_INSTRUCTOR`
    FOREIGN KEY (`instructor_id`)
    REFERENCES `instructor` (`id`)
    ...
);
```



Topic 3, Hibernate Advanced Mappings - @OneToMany

Azzeddine
RIGAT

Step 2: Create Course class

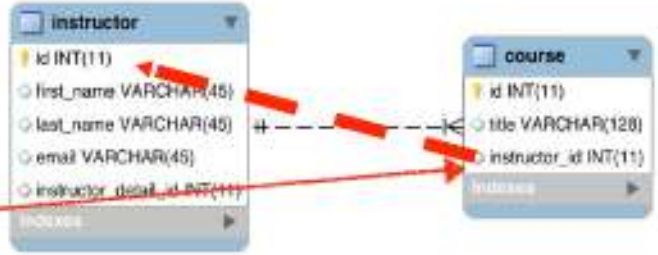
```
@Entity
@Table(name="course")
public class Course {

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

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

    ...
    // constructors, getters / setters

}
```

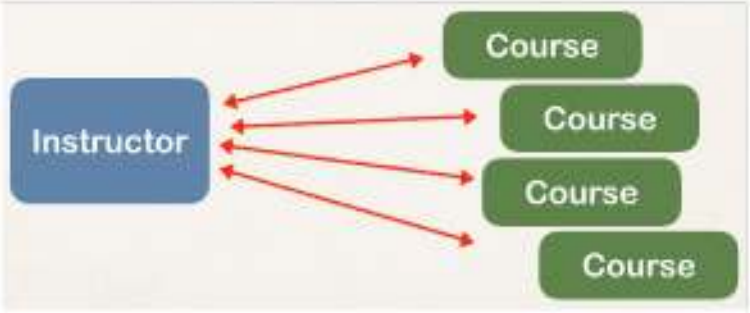
Topic 3, Hibernate Advanced Mappings - @OneToMany

Azzeddine RIGAT

Step 2: Create Course class - @ManyToOne

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

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

    ...
    // constructors, getters / setters
}
```





Topic 3, Hibernate Advanced Mappings - @OneToMany

Azzeddine
RIGAT

Step 3: Update Instructor - reference courses

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

private List<Course> courses;

  public List<Course> getCourses() {
      return courses;
  }

  public void setCourses(List<Course> courses) {
      this.courses = courses;
  }
  …
}
```

Topic 3, Hibernate Advanced Mappings - @OneToMany

Azzeddine
RIGAT

Add @OneToMany annotation

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

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


   public List<Course> getCourses() {
       return courses;
   }

   public void setCourses(List<Course> courses) {
       this.courses = courses;
   }
   …
```

Refers to “instructor” property in “Course” class

Topic 3, Hibernate Advanced Mappings - @OneToMany

Azzeddine
RIGAT

Add @OneToMany annotation

mappedBy tells Hibernate

- Look at the instructor property in the Course class
- Use information from the Course class @JoinColumn
- 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;
```

Topic 3, Hibernate Advanced Mappings - @OneToMany

Azzeddine RIGAT

Add support for Cascading

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

    @OneToMany(mappedBy="instructor",
        cascade={CascadeType.PERSIST, CascadeType.MERGE
        CascadeType.DETACH, CascadeType.REFRESH})
    private List<Course> courses;

    …

}
```

Do not apply cascading deletes!

Topic 3, Hibernate Advanced Mappings - @OneToMany

Azzeddine RIGAT

Add support for Cascading

```
@Entity
@Table(name="course")
public class Course {
    …
    @ManyToOne(cascade={CascadeType.PERSIST,
        CascadeType.MERGE  CascadeType.DETACH, CascadeType.REFRESH})
    @JoinColumn(name="instructor_id")
    private Instructor instructor;
    …
    // constructors, getters / setters

}
```

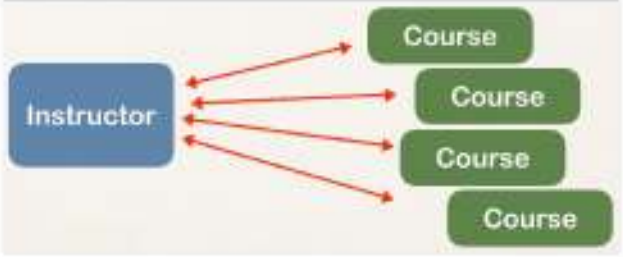
Do not apply cascading deletes!

Topic 3, Hibernate Advanced Mappings - @OneToMany

Azzeddine
RIGAT

Add convenience methods for bi-directional

```
@Entity
@Table(name="instructor")
public class Instructor {
    …
    // add convenience methods for bi-directional relationship
    public void add(Course tempCourse) {
        if (courses == null) {
            courses = new ArrayList<>();
        }
    courses.add(tempCourse);

    tempCourse.setInstructor(this);

    }
    …
}
```

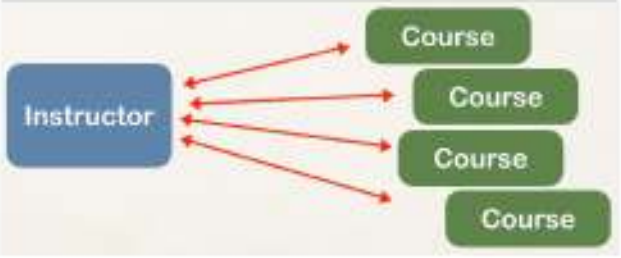


Topic 3, Hibernate Advanced Mappings - @OneToMany

Azzeddine
RIGAT

Step 4: Create Main App

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

    …
    // get the instructor object
    int theId = 1;
    Instructor tempInstructor = session.get(Instructor.class, theId);

    // print instructor
    System.out.println("tempInstructor: " + tempInstructor);

    // print the associated courses
    System.out.println("courses: " + tempInstructor.getCourses());
    …
}
```



Topic 3, Hibernate Advanced Mappings - @OneToMany

Azzeddine
RIGAT

Assignment 18

Make an implementation of @OneToMany in the bidirectional way using the sides code

Deadline: 2019-November-21

Topic 3, Hibernate Advanced Mappings - Eager vs Lazy Loading

Azzeddine
RIGAT

Hibernate Advanced Mappings - Fetch Types: Eager vs Lazy Loading

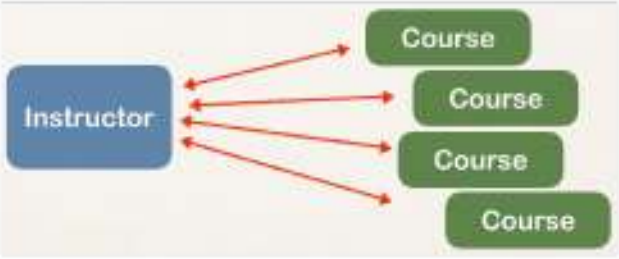
Topic 3, Hibernate Advanced Mappings - Eager vs Lazy Loading

Azzeddine
RIGAT

Fetches types

When we fetch / retrieve data, should we retrieve EVERYTHING?

- **Eager** will retrieve everything
- **Lazy** will retrieve on request



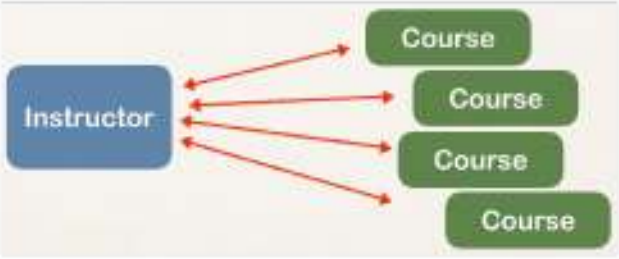
Topic 3, Hibernate Advanced Mappings - Eager vs Lazy Loading

Azzeddine
RIGAT

Eager Loading

- Eager loading will load all dependent entities

- Load instructor and all of their courses at once

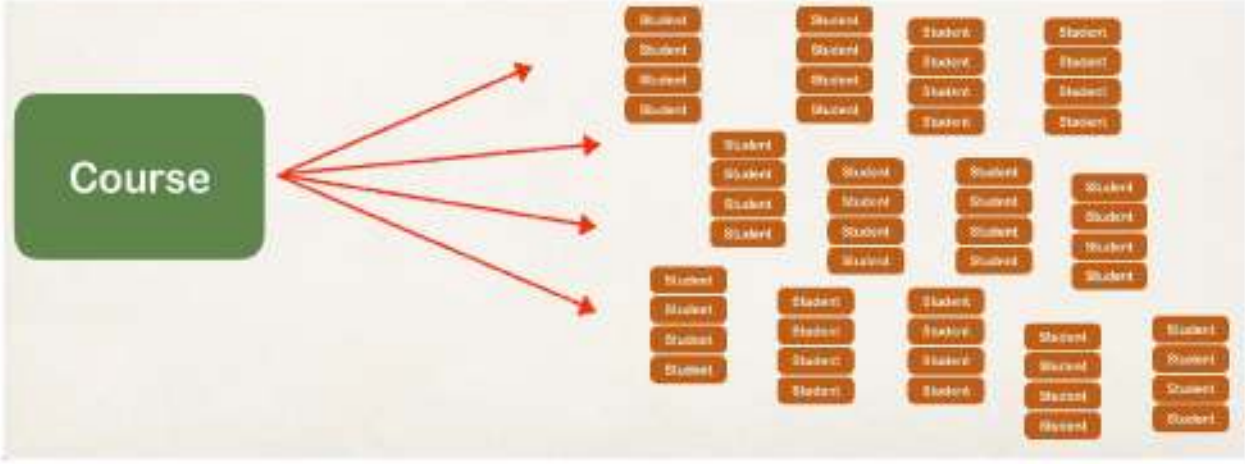


Topic 3, Hibernate Advanced Mappings - Eager vs Lazy Loading

Azzeddine
RIGAT

Eager Loading

- What about course and students?
- Could easily turn into a performance nightmare ….

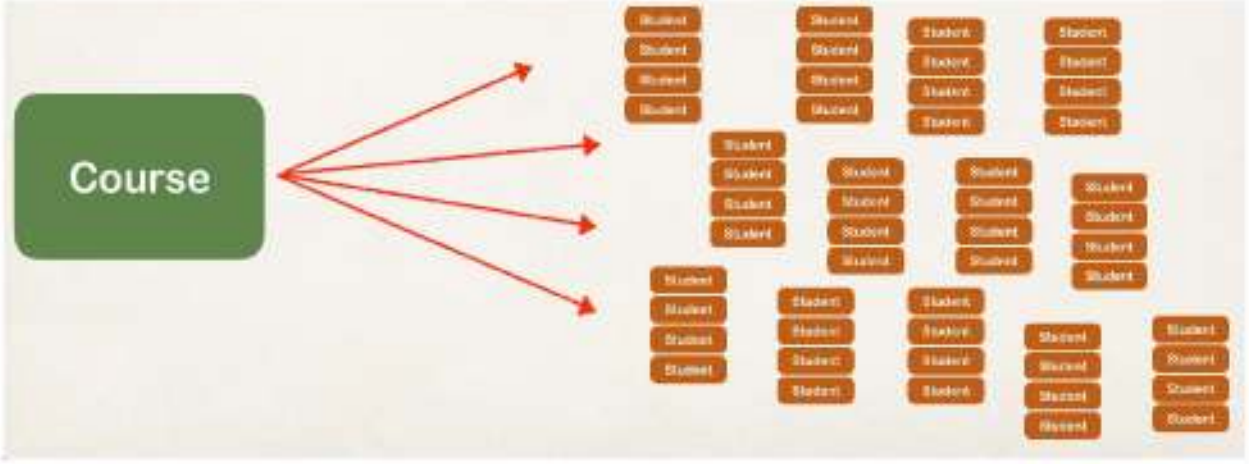


Topic 3, Hibernate Advanced Mappings - Eager vs Lazy Loading

Azzeddine
RIGAT

Eager Loading

- In our app, if we are searching for a course by keyword
  a. Only want a list of matching courses
- Eager loading would still load **all students** for **each course** …. not good!



Topic 3, Hibernate Advanced Mappings - Eager vs Lazy Loading

Azzeddine
RIGAT

Eager Loading

Best Practice

Only load data when absolutely needed

Prefer Lazy loading
instead of

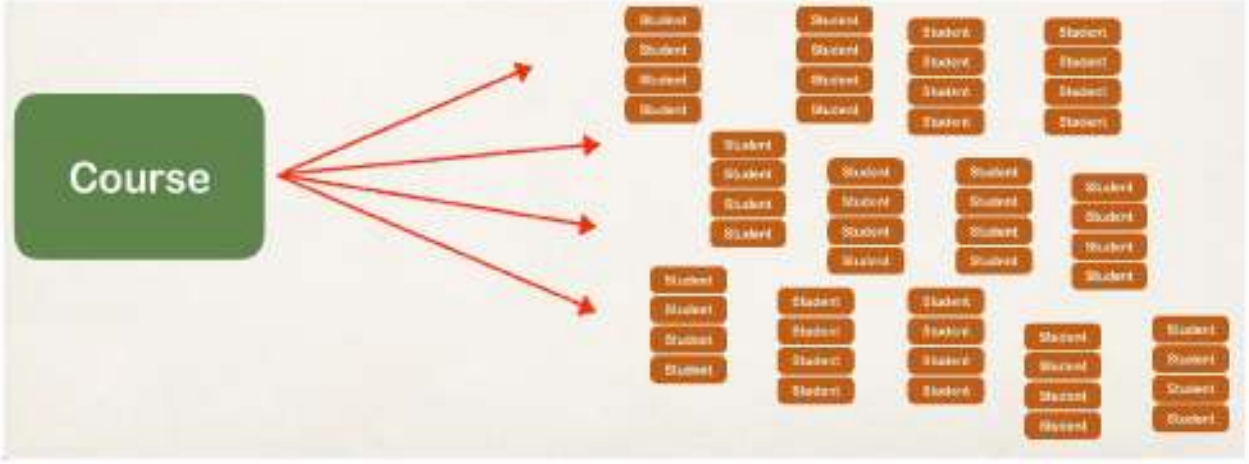
Eager loading

Topic 3, Hibernate Advanced Mappings - Eager vs Lazy Loading

Azzeddine
RIGAT

Eager Loading

- In our app, if we are searching for a course by keyword
  a. Only want a list of matching courses
- Eager loading would still load **all students** for **each course** …. not good!

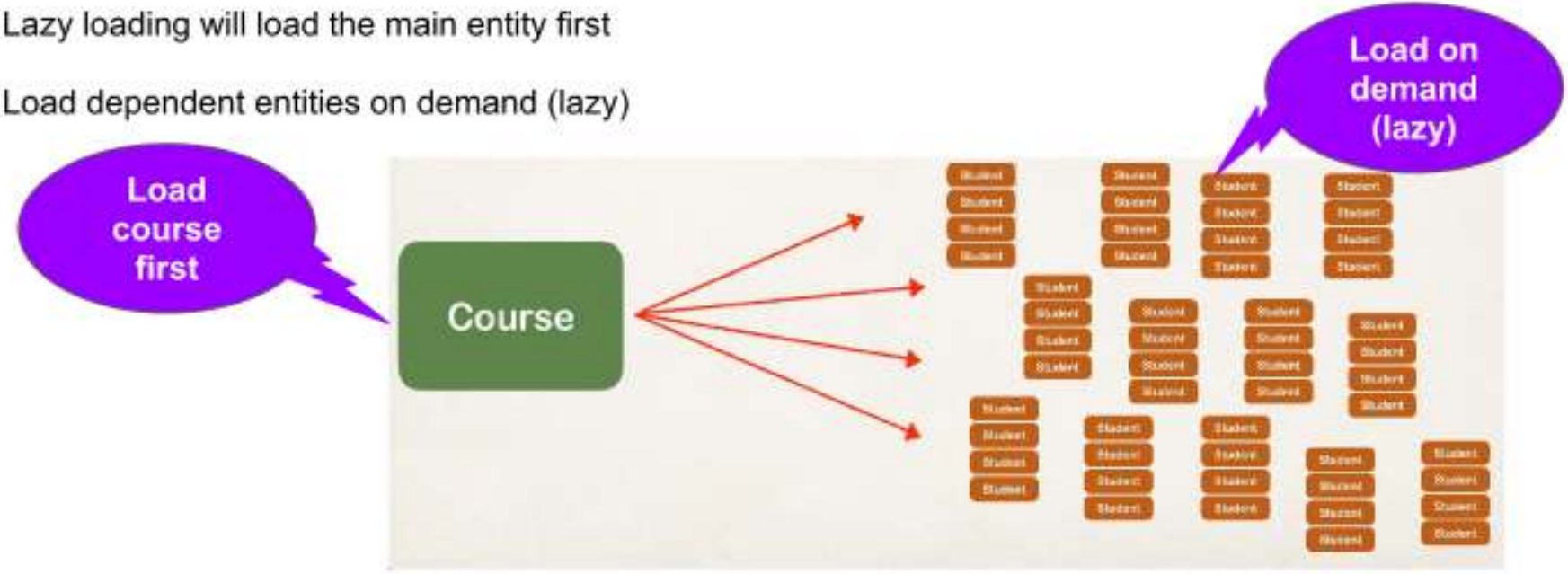


Topic 3, Hibernate Advanced Mappings - Eager vs Lazy Loading

Azzeddine
RIGAT

Lazy Loading

- Lazy loading will load the main entity first
- Load dependent entities on demand (lazy)



Topic 3, Hibernate Advanced Mappings - Eager vs Lazy Loading

Azzeddine
RIGAT

Real-World Use Case

Search for instructors

Javaweb academy

Search for Instructors | Go

Last Name	First Name	Email	Action
Public	Mary	mary@javaweb.edu	View Details
Doe	John	john@javaweb.edu	View Details
Rao	Ajay	ajay@javaweb.edu	View Details
Kublanov	Victor	victor@javaweb.edu	View Details
O'Keefe	Paul	paul@javaweb.edu	View Details
Peterson	Anthony	anthony@javaweb.edu	View Details
Patel	Trupti	trupti@javaweb.edu	View Details

Topic 3, Hibernate Advanced Mappings - Eager vs Lazy Loading

Azzeddine
RIGAT

Real-World Use Case: Master view

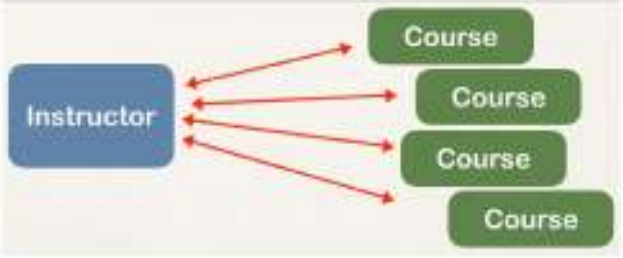
- In Master view, use lazy loading
- In Detail view, retrieve the entity and necessary dependent entities
- In Master view, use lazy loading for search results
- Only load instructors … not their courses

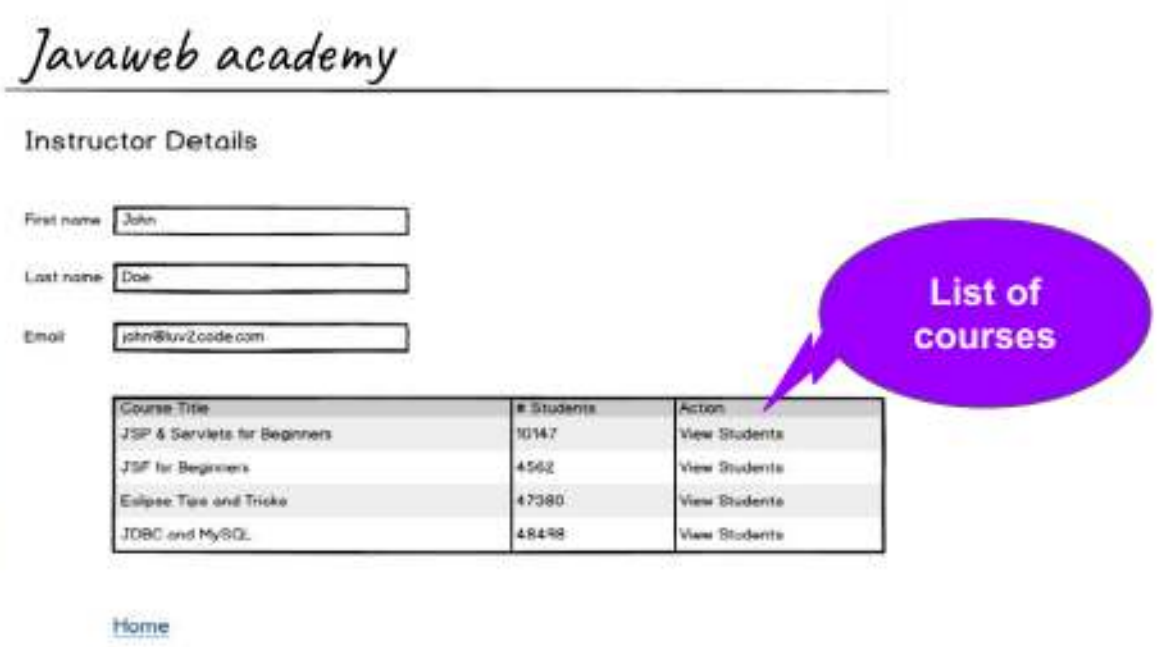
Topic 3, Hibernate Advanced Mappings - Eager vs Lazy Loading

Azzeddine
RIGAT

Real-World Use Case: Detail view

- In Detail view, retrieve the entity and necessary dependent entities
- Load instructor AND their courses





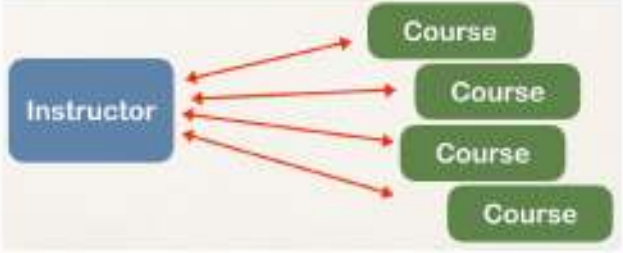
Topic 3, Hibernate Advanced Mappings - Eager vs Lazy Loading

Azzeddine
RIGAT

Fetch Type

- When you define the mapping relationship
- You can specify the fetch type: EAGER or LAZY

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

    @OneToMany(fetch=FetchType.LAZY, mappedBy="instructor")
    private List<Course> courses;

    …
}
```



Topic 3, Hibernate Advanced Mappings - Eager vs Lazy Loading

Azzeddine
RIGAT

Default Fetch Types

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

Topic 3, Hibernate Advanced Mappings - Eager vs Lazy Loading

Azzeddine
RIGAT

Overriding Default Fetch Type

Specifying the fetch type, overrides the defaults

```
@ManyToOne(fetch=FetchType.LAZY)
@JoinColumn(name="instructor_id")
private Instructor instructor;
```

Mapping	Default Fetch Type
@ManyToOne	FetchType.EAGER

Topic 3, Hibernate Advanced Mappings - Eager vs Lazy Loading

Azzeddine
RIGAT

More about Lazy Loading

- When you lazy load, the data is only retrieved on demand
- However, this requires an open Hibernate session
    - need an connection to database to retrieve data

Topic 3, Hibernate Advanced Mappings - Eager vs Lazy Loading

Azzeddine
RIGAT

More about Lazy Loading

- If the Hibernate session is closed
  - And you attempt to retrieve lazy data
  - Hibernate will throw an exception

Topic 3, Hibernate Advanced Mappings - Eager vs Lazy Loading

Azzeddine
RIGAT

More about Lazy Loading

- To retrieve lazy data, you will need to open a Hibernate session
- Retrieve lazy data using
    - Option 1: session.get and call appropriate getter method(s)
    - Option 2: Hibernate query with HQL
- Many other techniques available but the two above are most common

Topic 3, Hibernate Advanced Mappings - @OneToMany - Unidirectional

Azzeddine RIGAT

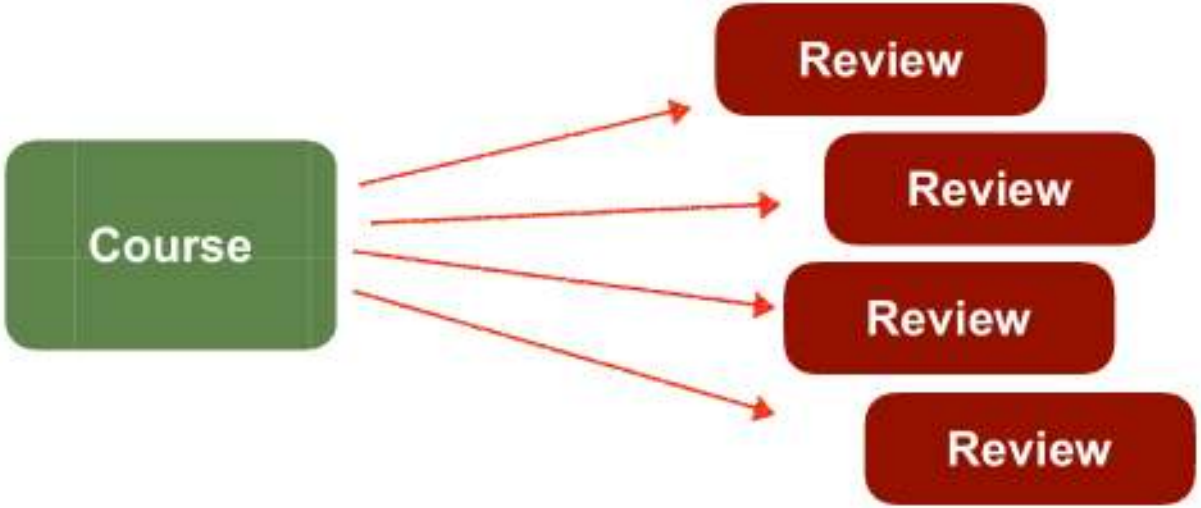
Hibernate Advanced Mappings - @OneToMany - Unidirectional

Topic 3, Hibernate Advanced Mappings - @OneToMany - Unidirectional

Azzeddine RIGAT

One-to-Many Mapping

- A course can have many reviews
  a. Uni-directional



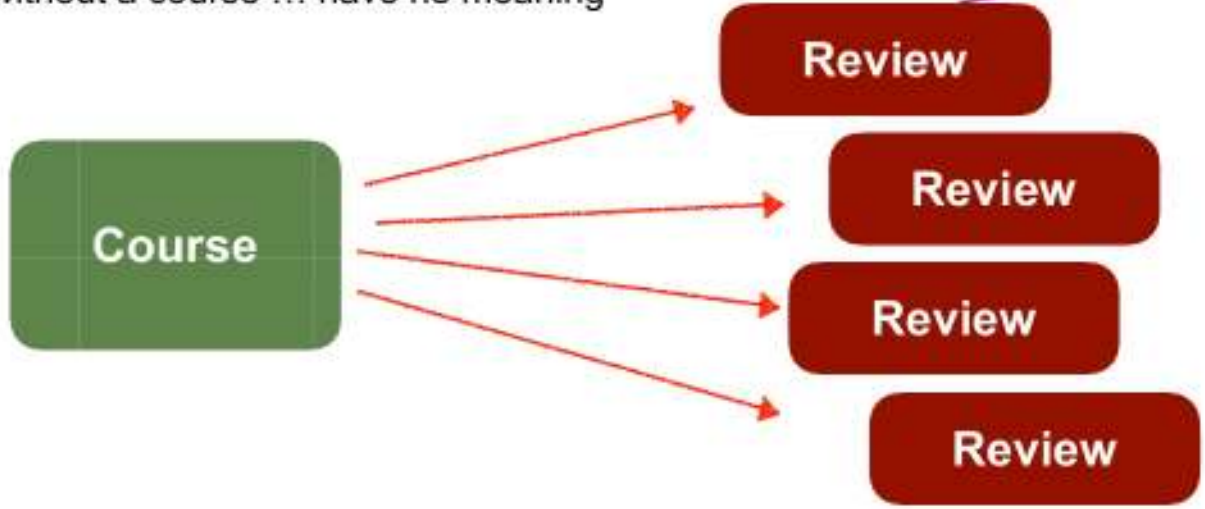
Topic 3, Hibernate Advanced Mappings - @OneToMany - Unidirectional

Azzeddine RIGAT

One-to-Many Mapping

- If you delete a course, also delete the reviews
- Reviews without a course … have no meaning

Apply cascading deletes!



Topic 3, Hibernate Advanced Mappings - @OneToMany - Unidirectional

Azzeddine RIGAT

@OneToMany

Topic 3, Hibernate Advanced Mappings - @OneToMany - Unidirectional

Azzeddine RIGAT

Look Mom … our project is growing!

Topic 3, Hibernate Advanced Mappings - @OneToMany - Unidirectional

Azzeddine RIGAT

Development Process: One-to-Many

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

Topic 3, Hibernate Advanced Mappings - @OneToMany - Unidirectional

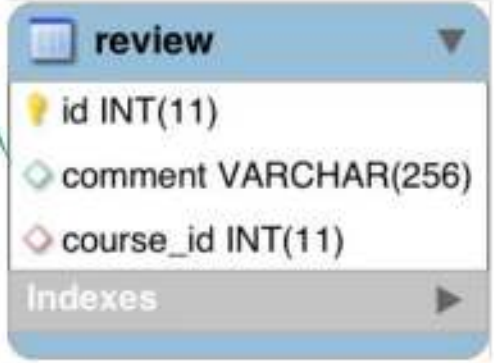
Azzeddine RIGAT

table: review

File: create-db.sql

```
CREATE TABLE `review` (

    `id` int(11) NOT NULL AUTO_INCREMENT,
    `comment` varchar(256) DEFAULT NULL,
    `course_id` int(11) DEFAULT NULL,

    ...
);
```

comment:
"Wow ... this course awesome!"



Topic 3, Hibernate Advanced Mappings - @OneToMany - Unidirectional

Azzeddine RIGAT

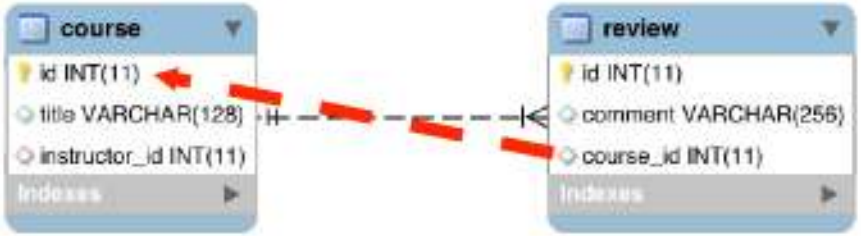
table: review - foreign key

File: create-db.sql

```
CREATE TABLE `review` (
    ...
    KEY `FK_COURSE_ID_idx` (`course_id`),
    CONSTRAINT `FK_COURSE`
    FOREIGN KEY (`course_id`)
    REFERENCES `course` (`id`)
    ...
);
```

Table

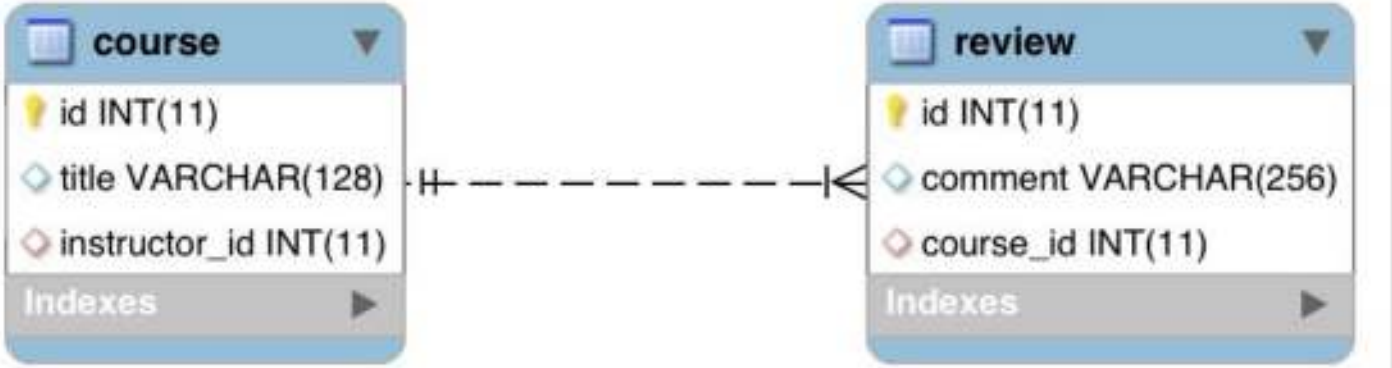
Column



Topic 3, Hibernate Advanced Mappings - @OneToMany - Unidirectional

Azzeddine RIGAT

table: course - no changes

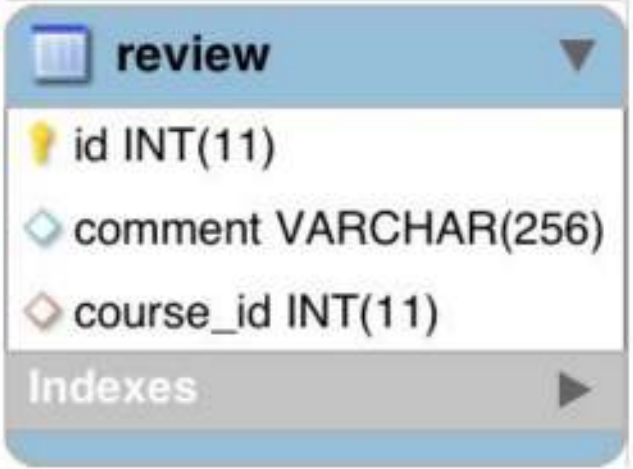


Topic 3, Hibernate Advanced Mappings - @OneToMany - Unidirectional

Azzeddine RIGAT

Step 2: Create Review class

```
@Entity
@Table(name="review")
public class Review {

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

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

    ...
    // constructors, getters / setters
}
```

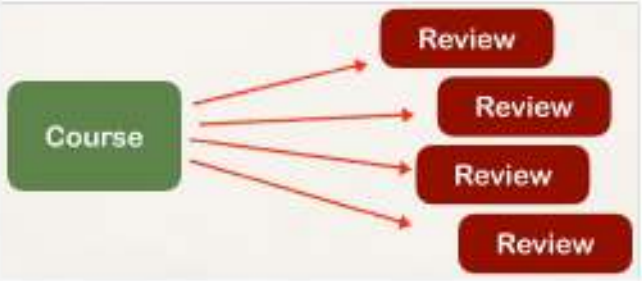


Topic 3, Hibernate Advanced Mappings - @OneToMany - Unidirectional

Azzeddine RIGAT

Step 3: Update Course - reference reviews

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

    private List<Review> reviews;

    // getter / setters
    ...

}
```



Topic 3, Hibernate Advanced Mappings - @OneToMany - Unidirectional

Azzeddine RIGAT

Add @OneToMany annotation

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


    @OneToMany
    @JoinColumn(name="course_id")
    private List<Review> reviews;

    // getter / setters
    …

}
```

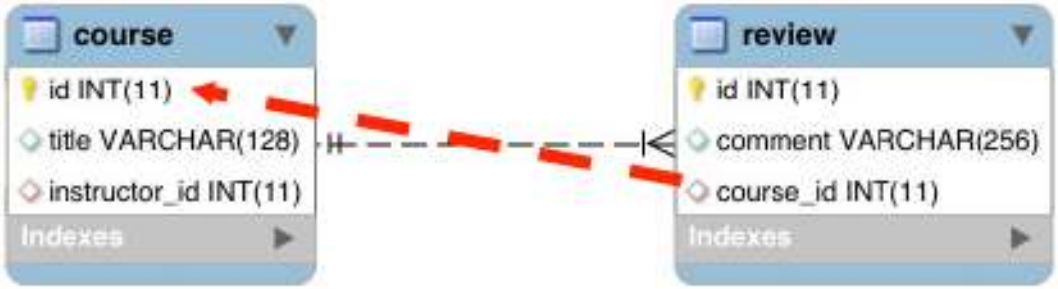
Refers to “course_id” column in “review” table

Topic 3, Hibernate Advanced Mappings - @OneToMany - Unidirectional

Azzeddine RIGAT

More: @JoinColumn

- In this scenario, @JoinColumn tells Hibernate
  - Look at the course_id column in the **review** table
  -
  - Use this information to help find associated reviews for a course




```
public class Course {
    …

    @OneToMany
    @JoinColumn(name="course_id")
    private List<Review> reviews;
```

Topic 3, Hibernate Advanced Mappings - @OneToMany - Unidirectional

Azzeddine RIGAT

Add support for Cascading

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

    @OneToMany(cascade=CascadeType.ALL)
    @JoinColumn(name="course_id")
    private List<Review> reviews;

    …

}
```

Cascade all operations including deletes!

Topic 3, Hibernate Advanced Mappings - @OneToMany - Unidirectional

Azzeddine RIGAT

Add support for Lazy loading

Lazy load the reviews

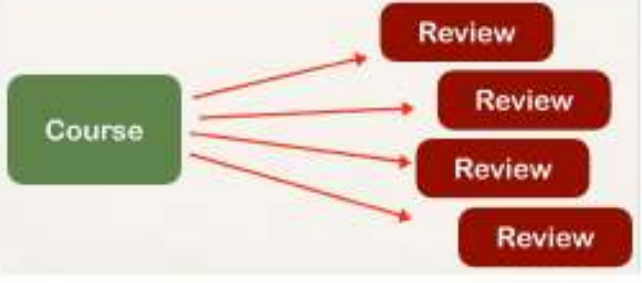
```
@Entity  @Table(name="course")
public class Course {
    …

    @OneToMany(fetch=FetchType.LAZY, cascade=CascadeType.ALL)
    JoinColumn(name="course_id")
    private List<Review> reviews;

    …

}
```

Topic 3, Hibernate Advanced Mappings - @OneToMany - Unidirectional

Azzeddine RIGAT

Add convenience method for adding review

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

    …
    // add convenience methods for adding reviews
    public void add(Review tempReview) {
        if (reviews == null) {
            reviews = new ArrayList<>();
        }

    reviews.add(tempReview);

    }
    …
}
```

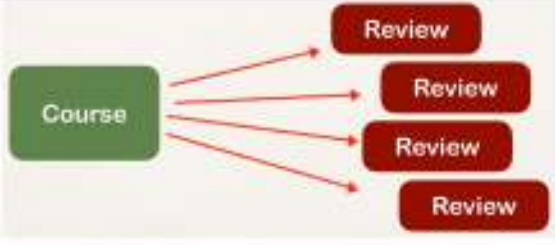


Topic 3, Hibernate Advanced Mappings - @OneToMany - Unidirectional

Azzeddine RIGAT

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 reviews
    System.out.println("reviews: " + tempCourse.getReviews());
    …
}
```



Lazy load the reviews

Topic 3, Hibernate Advanced Mappings - @ManyToMany

Azzeddine
RIGAT

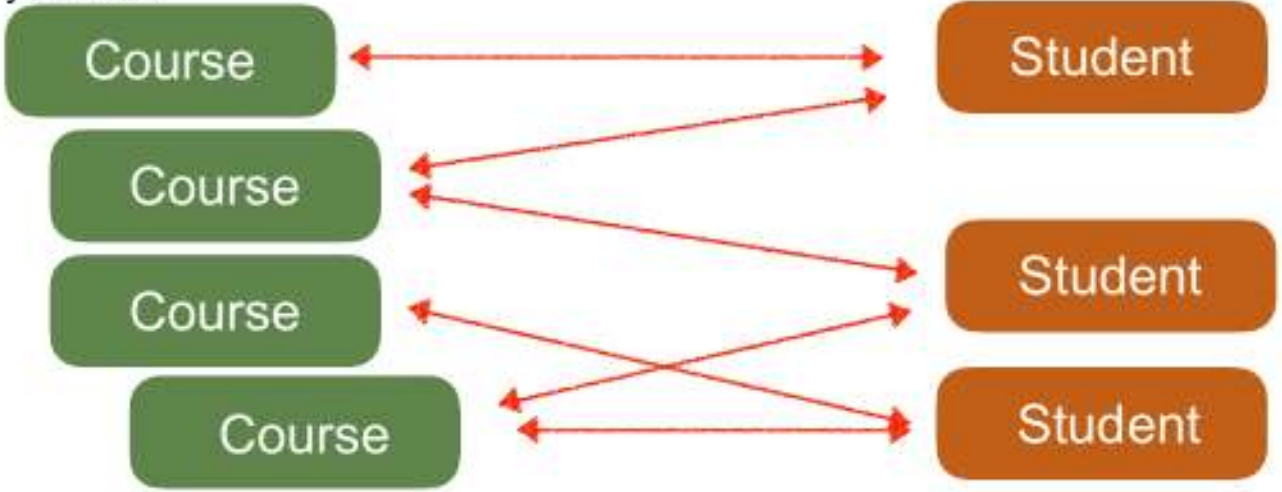
Hibernate Advanced Mappings - @ManyToMany

Topic 3, Hibernate Advanced Mappings - @ManyToMany

Azzeddine
RIGAT

Many-to-Many Mapping

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

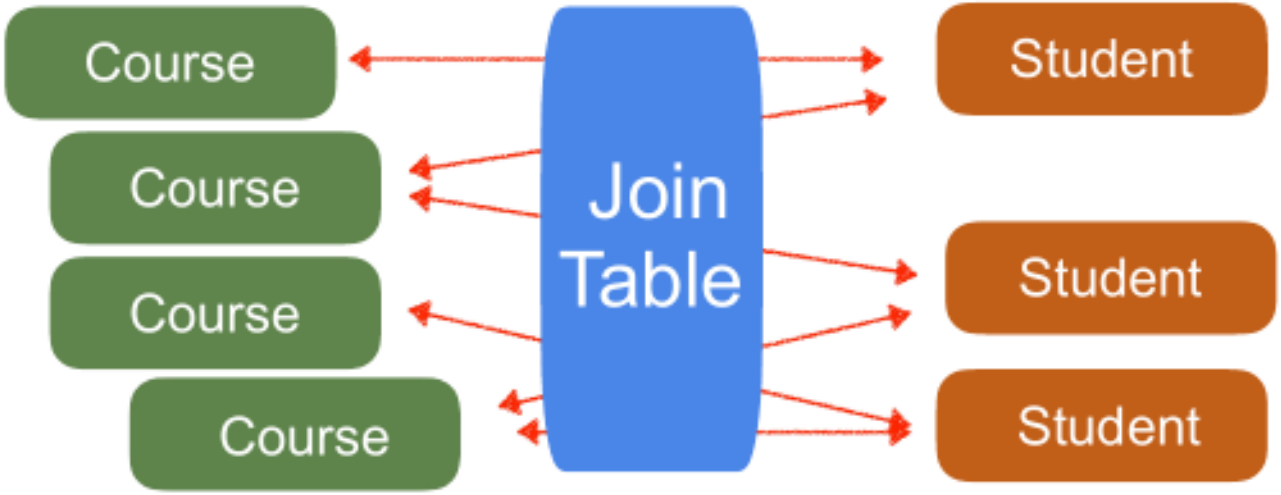


Topic 3, Hibernate Advanced Mappings - @ManyToMany

Azzeddine
RIGAT

Keep track of relationships

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



Topic 3, Hibernate Advanced Mappings - @ManyToMany

Azzeddine
RIGAT

Join Table

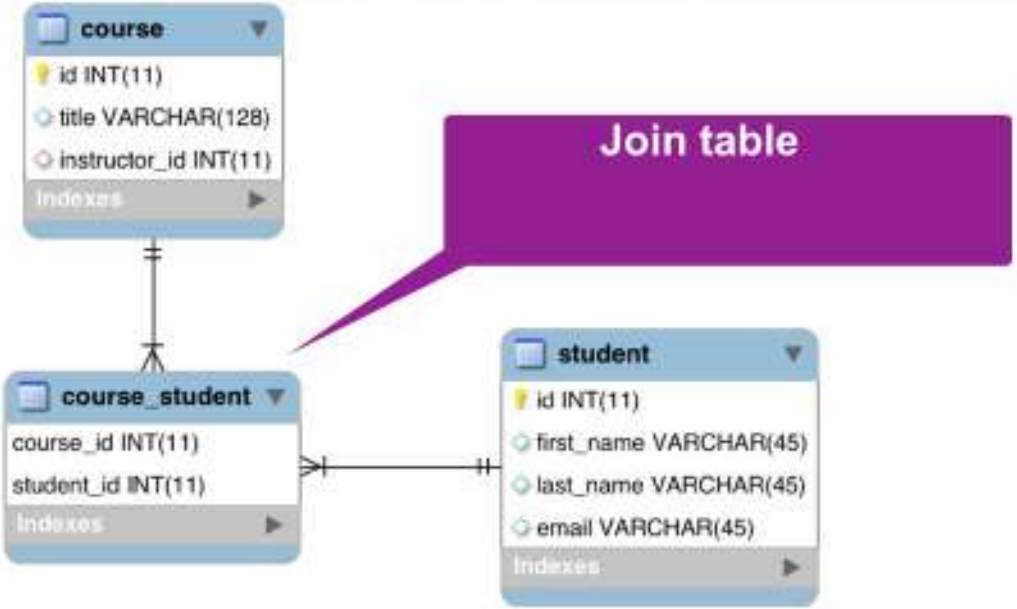
A table that provides a mapping between two tables.

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

Topic 3, Hibernate Advanced Mappings - @ManyToMany

Azzeddine
RIGAT

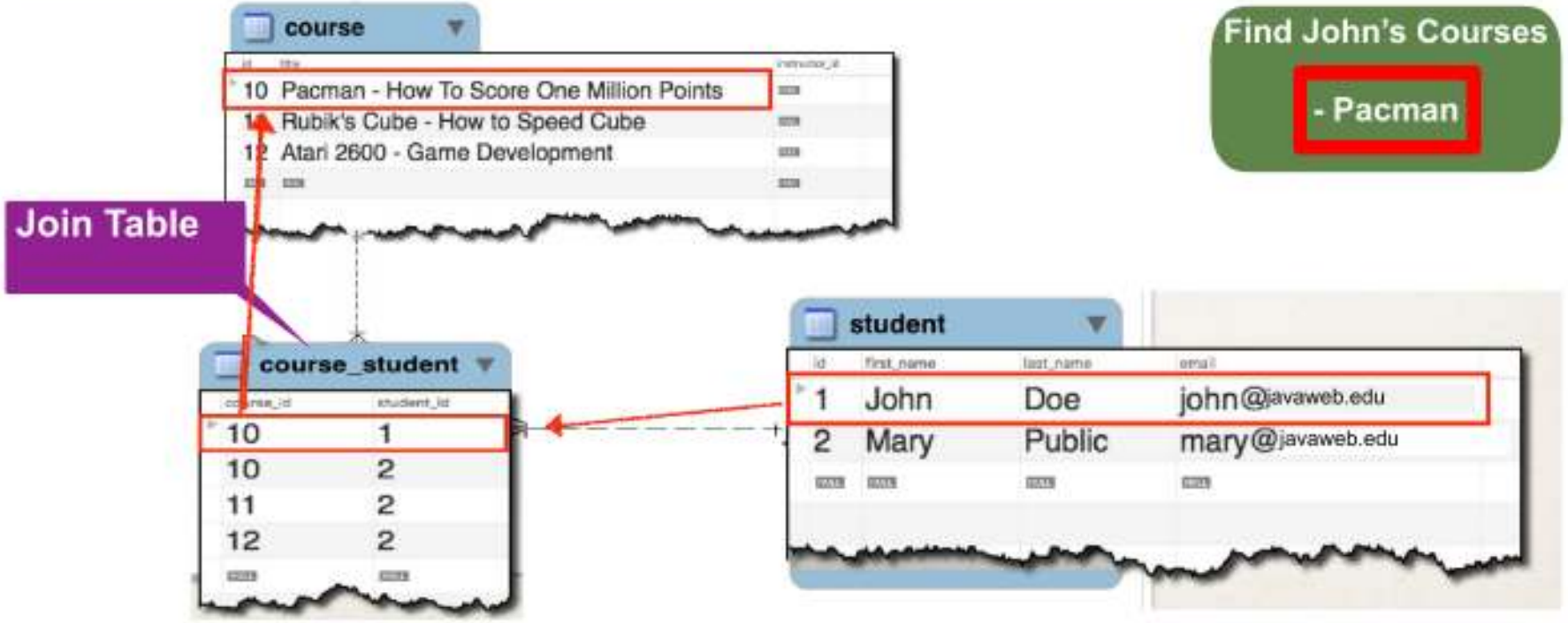
@ManyToMany



Azzeddine
RIGAT

Topic 3, Hibernate Advanced Mappings - @ManyToMany

Join Table Example



Topic 3, Hibernate Advanced Mappings - @ManyToMany

Azzeddine
RIGAT

Development Process: Many-to-Many

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

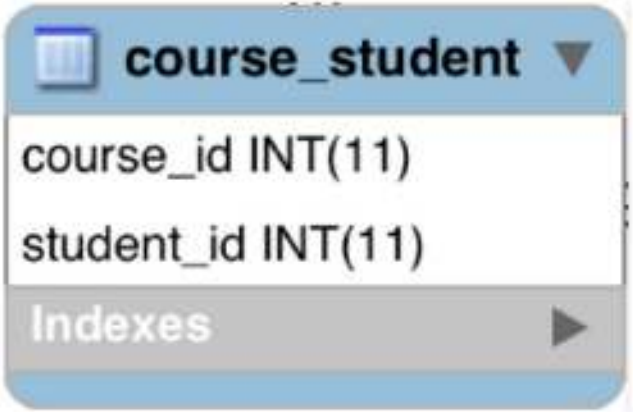
Topic 3, Hibernate Advanced Mappings - @ManyToMany

Azzeddine
RIGAT

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`),
    ...
);
```



Azzeddine
RIGAT

Topic 3, Hibernate Advanced Mappings - @ManyToMany

join table: course_student - foreign keys

File: create-db.sql

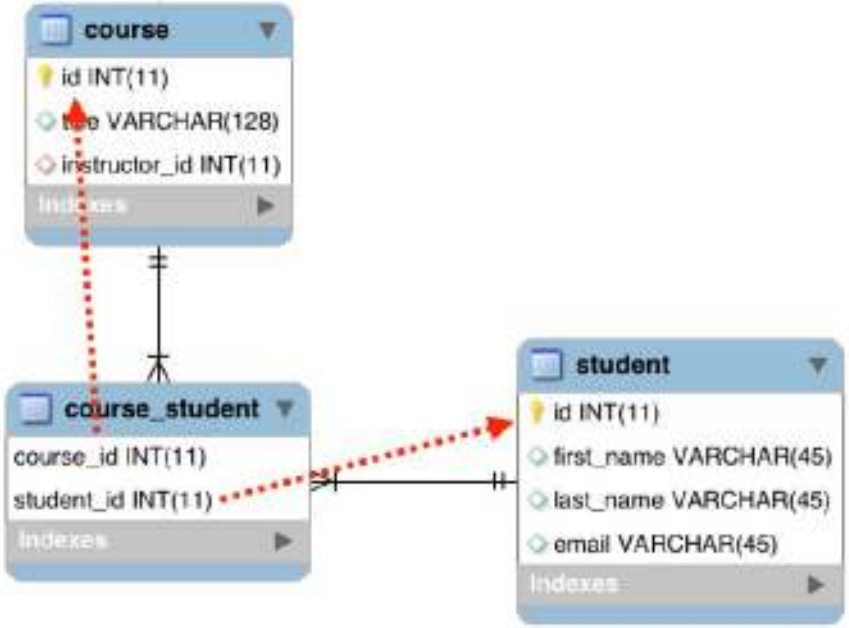
```
CREATE TABLE `course_student` (
    ...
    CONSTRAINT `FK_COURSE_05`
    FOREIGN KEY (`course_id`)
    REFERENCES `course` (`id`),

    CO  `FK_STUDENT`

    FOREIGN KEY (`student_id
    REFERENCES `student` (`id`),
    ...
);
```

Column

Table

Column

Table

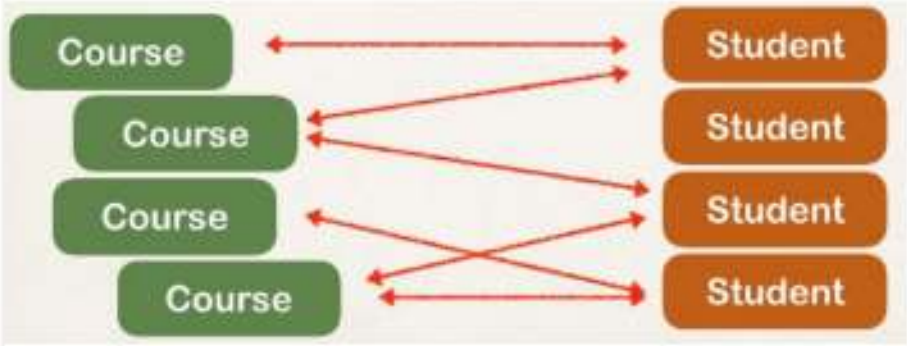


Topic 3, Hibernate Advanced Mappings - @ManyToMany

Azzeddine
RIGAT

Step 2: Update Course - reference students

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

    private List<Student> students;

    // getter / setters
    ...

}
```



Topic 3, Hibernate Advanced Mappings - @ManyToMany

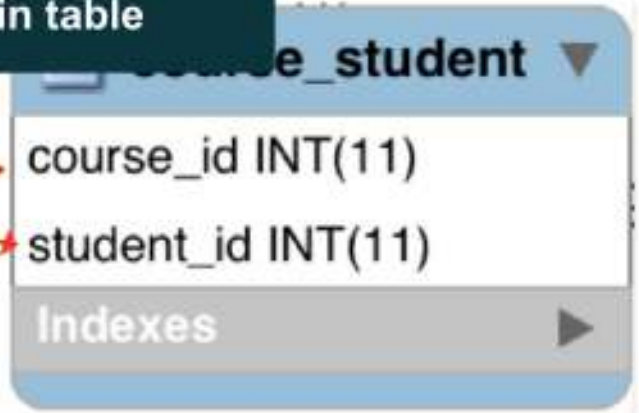
Azzeddine
RIGAT

Add @ManyToMany

```
@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 "course_id" column in "course_

Refers to "student_id" column in "course_student" join table



Topic 3, Hibernate Advanced Mappings - @ManyToMany

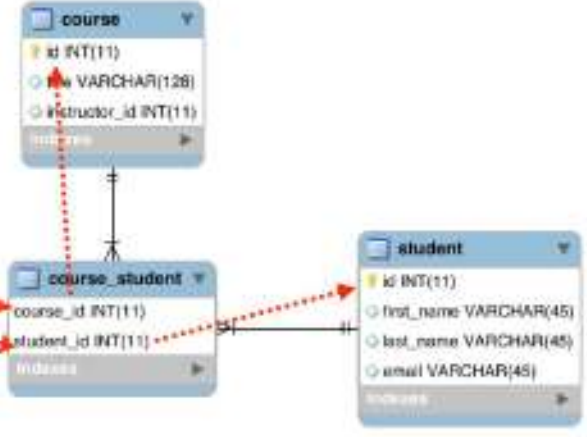
Azzeddine
RIGAT

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;
}
```

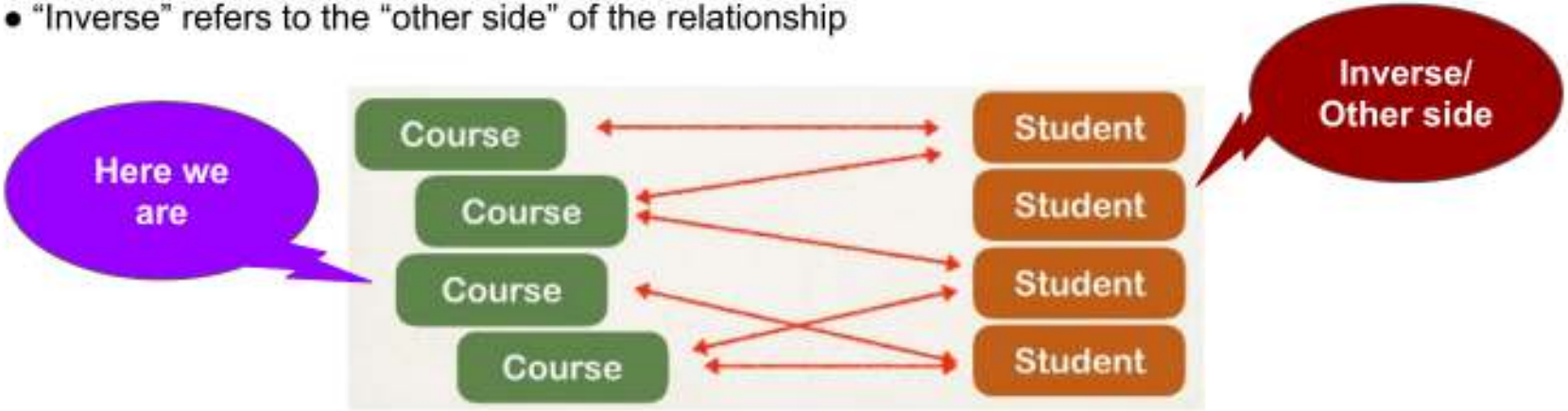


Topic 3, Hibernate Advanced Mappings - @ManyToMany

Azzeddine
RIGAT

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



Topic 3, Hibernate Advanced Mappings - @ManyToMany

Azzeddine
RIGAT

Now, let's do a similar thing for Student

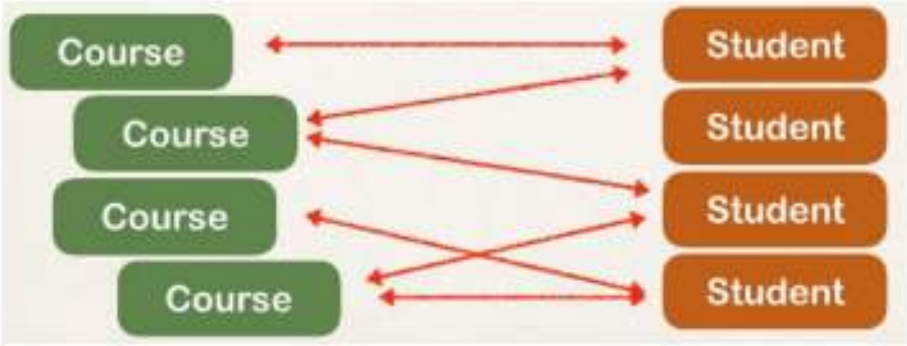
just going the other way …

Topic 3, Hibernate Advanced Mappings - @ManyToMany

Azzeddine
RIGAT

Step 3: Update Student - reference courses

```
@Entity

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

   private List<Course> courses;

   // getter / setters
   …

}
```



Topic 3, Hibernate Advanced Mappings - @ManyToMany

Azzeddine
RIGAT

Add @ManyToMany

```
@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

column in table

_student

course_id INT(11)

student_id INT(11)

Indexes

Topic 3, Hibernate Advanced Mappings - @ManyToMany

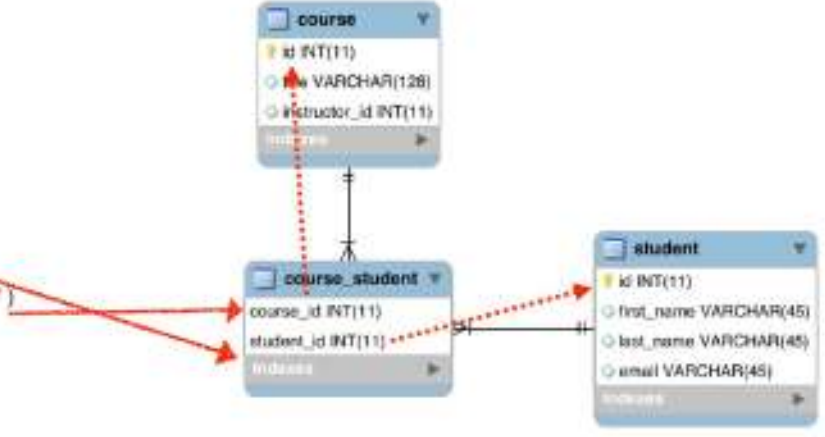
Azzeddine
RIGAT

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 **course** and **students**

```
public class Student {
    @ManyToMany
    @JoinTable(
        name="course_student",
        joinColumns=@JoinColumn(name="student_id"),
        inverseJoinColumns=@JoinColumn(name="course_id")
        )
    private List<Courses> courses;
}
```

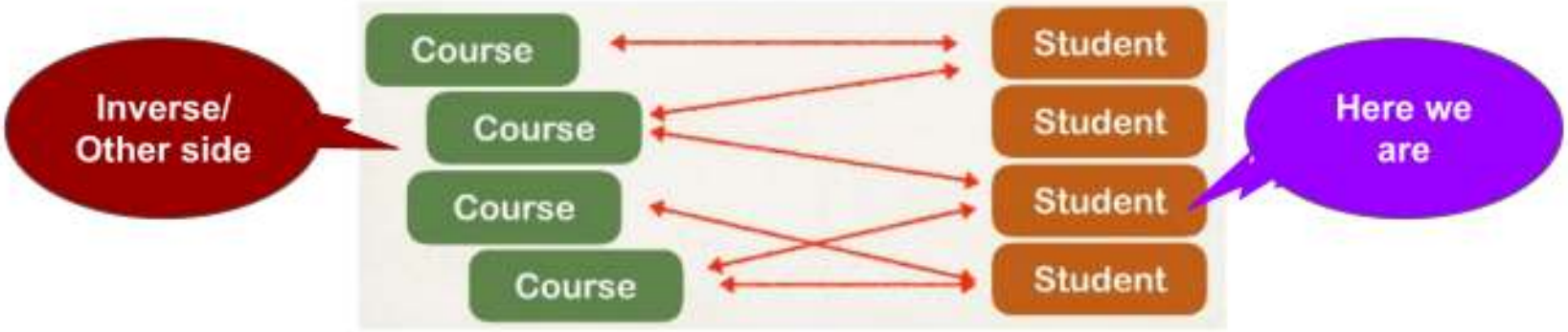


Topic 3, Hibernate Advanced Mappings - @ManyToMany

Azzeddine
RIGAT

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

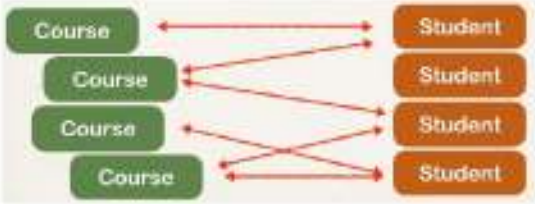


Topic 3, Hibernate Advanced Mappings - @ManyToMany

Azzeddine
RIGAT

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());
    …
}
```



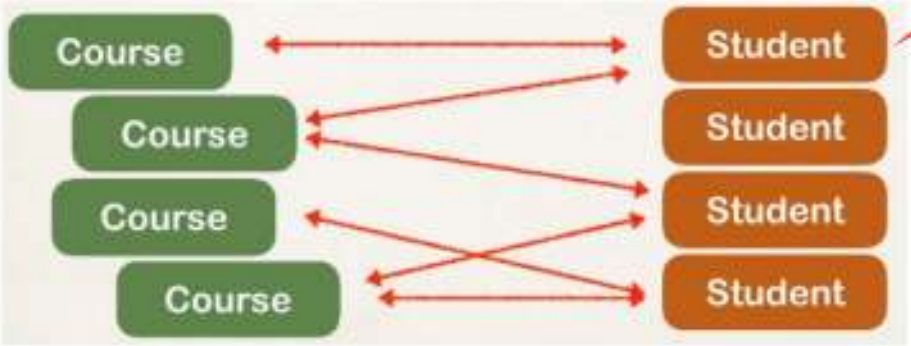
Topic 3, Hibernate Advanced Mappings - @ManyToMany

Azzeddine
RIGAT

Real-World Project Requirement

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

DO NOT apply cascading deletes!



Topic 3, Hibernate Advanced Mappings - @ManyToMany

Azzeddine
RIGAT

Other features

In the next set of lectures, we'll add support for other features

- **Lazy Loading** of students and courses
- **Cascading** to handle cascading saves … but NOT deletes
    a. If we delete a course, DO NOT delete students
    b. If we delete a student, DO NOT delete courses

Topic 3, Hibernate Advanced Mappings - @ManyToMany

Azzeddine
RIGAT

Assignment 19

Make an implementation of @ManyToMany in the bidirectional way using the sides code

Deadline: 2019-November-21