Hibernate Advanced Mapping

In DB

1 Multiple Tables

2 Relationship btw tables

Need to model this with Hibernate

Types

1 one to one

Ex : An instructor can have instructor details , Similar instrctur profile

Instructor(T)-> Instructor Details(T)

2 one to many , many to one

Ex: One to many

instructor can have many courses inverse of this is many to one relation many courses can have single Instrctor

3 many to many

Ex : A course an have many student and student can have many courses

Database concepts

1 primary key and Foreign key

-primary key : identify unique row in a table

-Foreign key : Link table together -> a field in one table refer to primary key in another table

2 cascade

Apply same operation to related entites

If I save Instrctor it will cascade and save instructor\_details also

If we save Instructor it performs same operation to Instructor\_details

If we delete Instructor should delete their instructor details also bcz they no longer have record known as “**CASCADE DELETE**”

We have to be carefull with cascade delete in terms Many to many we should not delete check use case

Fetch Types: Eager VS Lazy Loading => “should we retrieve everything”

Eager will retrieve everything

Lazy will retrieve on request

Uni – Directional Relationship

One way relation using Instrctor get instructor details

Instructor-> Instrctor\_details

Bi –directional

Both ways

Instructor<--> Instrctor\_details

ONE TO ONE

1st uni – directional example

Deveolpment process one to one

1 Define database tables

2 create instructor\_details class

3 create instructor class

4 create main App

Table : instructor\_details

create table instructor\_detail(id serial PRIMARY KEY , youtube\_channel VARCHAR(200) DEFAULT NULL,hobby VARCHAR(100) DEFAULT NULL);

Table : instructor

CREATE TABLE instructor (id serial PRIMARY KEY,first\_name varchar(45) DEFAULT NULL,last\_name varchar(45) DEFAULT NULL,email varchar(45) DEFAULT NULL,

instructor\_detail\_id int DEFAULT null, CONSTRAINT fk\_detail FOREIGN KEY (instructor\_detail\_id) REFERENCES instructor\_detail(id)

)

Need to set relation to the table instructor\_detail\_id

Link the tbale using FK

Definf FK

CONSTRAINT fk\_detail FOREIGN KEY (instructor\_detail\_id) REFERENCES instructor\_detail(id)

Forieng key

Preserve relatiosnship btween tbles

Referential Integrity

Prevents the operations that would destroy relationship

Ensure only valid data is inserted into the foreign key column

Can contain only valid reference to the primary key another table

Refactor and Exception handling

javax.persistenceNullPoniterException: Null Pointer

Bi instructorDetail null

java.lang.NullPointerException

at com.mapping.onetoonedemo.DemoExceptionHandle.main(DemoExceptionHandle.java:29)

javax.persistence.NoResultException: No entity found for query

Cascade Delete

Delete InstrutorDetail along with Instructor

Delete only InstrutorDetail keep Instructor

-Modify Cascade on InstructorDetails Entity

In InstructorDetails in cascade type select Excpet REMOVE

@OneToOne(mappedBy="instructorDetail",cascade= {CascadeType.***DETACH***,CascadeType.***MERGE***,CascadeType.***PERSIST***,CascadeType.***REFRESH***,CascadeType.***REMOVE***})

Perform same delete operation of previous one and c changes

javax.persistence.EntityNotFoundException: deleted object would be re-saved by cascade (remove deleted object from associations): [com.mapping.entity.InstructorDetail#3]

need to remove bi directional in Main app code

// remove the assoicated object reference

// break bi directionl link

instructorDetail.getInstructor().setInstructorDetail(**null**);

session.delete(instructorDetail);

OneToMany

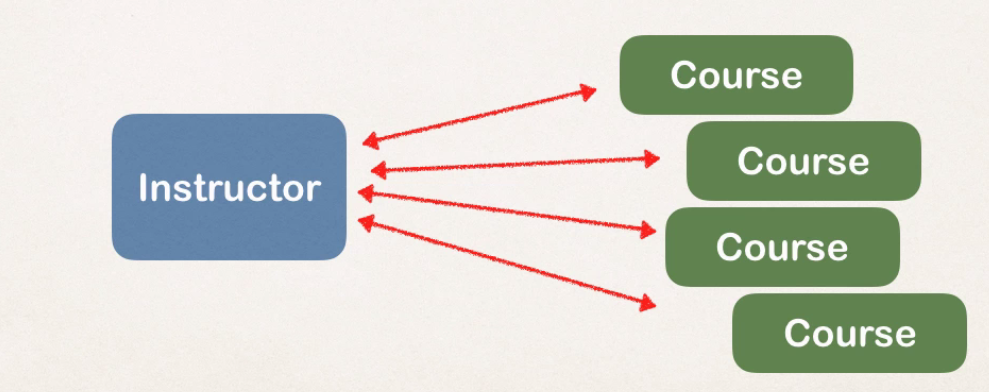
Instructor can have many cource

* Bi directional

Many to One

Many cource can have one intrsrctor

Inverse/ oppsite of one to many



Real project

If delete instructor don’t delete course

If you delete course don’t delete instructor

Means Do not apply cascade DELETE

Development Process

1 define tables

2 create cource class

3 update instructor class

4 main app

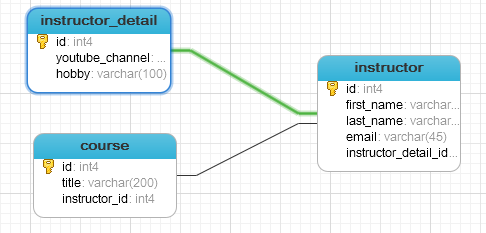
Create Couse table

create table course (id serial PRIMARY KEY , title VARCHAR(200) ,instructor\_id int,

CONSTRAINT FK\_INSTRUCTOR\_MANY FOREIGN KEY (instructor\_id) REFERENCES instructormany(id)

)

Add unique key to title to prevent duplicate



Create Course Class

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Column(name="id")

**private** **int** id;

@Column(name="title")

**private** String title;

// many course has one instructor

// cascade is imp dnt delete so check cascade delete

@ManyToOne(cascade = {CascadeType.***PERSIST***,CascadeType.***DETACH***,CascadeType.***MERGE***,CascadeType.***REFRESH***})

@JoinColumn(name="instructor\_id") // instructor\_id is kay in Course table which points if of Instructor table

**private** Instructor instructor;

Update instructor class

Mapped By

In Instructor entity look into instructor property in Course class use cource column from @joinColumn

Help to find assoicated courses for instructor

Add supprting CASCADE except REMOVE so no CASCADE DELETE in both classes

Add convinecen methhod for bi-driectional

4 Create MainAPP

Fetch Types

When we load data should we retrive everything

1 Eager will retrive everything

- it will load all dependententites ex: load Instructor and all of their cources at once

- one shot to db bring all data and its assoicated data

-its k for small no of instructor and Coursce if its big impact performance of applciation

Easily could turn into performnace nightmare

All of the studnets for the cource will slow applciation and its performance

-in our app if we are searching for a cource by keyword

- only want a list of matching cources

-Eager wuld still load all the studnets for each cource which is not good

2 lazy will retrive on request

- it will load main entity first

- load dependent entites on demand(lazy)

Load Cource 1st then when u need student then u load it on demand

Lazy lodaing is prefered

Best practice : only load data when absolutly needed prefer lazy Loading insted of Eager loading

Real Time

Instructor details if want see that instructor go to its details load its assoaicted class

Real Time use case

- in mastee view, use lazy loading

- in detail view, retrive the entity and nessary dependent entites

Real Time use case -MASTER VIEW

- in master view use lazy loading for search case

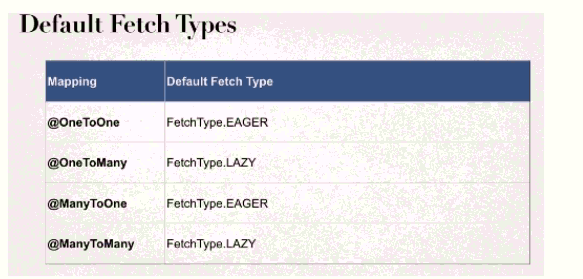
- only load the instructor not their Cources

Real Time use case -Detail VIEW

- in detail view , retrive entity and its nesscry dependent entity

-Load instructor and thier Cources

Fetch Type



When we define mapping relationship we can specify fetch type Eager / lazy

In Intstructor Class

Where we call course there we can aply with mapping relation

Override Default Fetch type

Specifying the fetch type , overrides the defaults

More on Lazy Loading

- when you lazy load, data is only retrived on demand

- require open hibernate session

-need an connection to retrive data from db

- if hbiernate session session I sclosed and trying to retrive lazy data hibernate will throw exception

Retrtive lazt data using

Option 2: HQL

Test Lazy Loading

1 lets break it on purpose close session

session.beginTransaction();

// get Instructor from DB

Instructor instructor=session.get(Instructor.**class**, 5);

System.***err***.println("Instructor object "+ instructor );

session.getTransaction().commit();

// break the session by closing session close since Lazy Fetch reqiure hibernate session

session.close();

// get Course from instructor

System.***err***.println( "Cources : "+ instructor.getCourses());

// .getCourses() this our lazy data

org.hibernate.LazyInitializationException: failed to lazily initialize a collection of role: com.mapping.entity.Instructor.courses, could not initialize proxy - no Session

Resolve the issue:

System.***err***.println( "Cources : "+ instructor.getCourses()); use this line before session.getTransaction().commit(); ,session.close();

session.beginTransaction();

// get Instructor from DB

Instructor instructor=session.get(Instructor.**class**, 5);

System.***err***.println("Instructor object "+ instructor );

// get Course from instructor

System.***err***.println( "Cources : "+ instructor.getCourses());

// .getCourses() this our lazy data

session.getTransaction().commit();

session.close();

Excuted before closing session : reolved this issues using Option 1 : session.get call appricate getter method

Option 2 : using HQL

session.beginTransaction();

// get Instructor from DB

//when excuted will load instructor and course all at once

~~Query~~<Instructor> query=session.createQuery("select i from Instructor i "

+ " JOIN FETCH i.courses where i.id=:theInstructorId ",Instructor.**class** );

// set paratmeter on query

query.~~setParameter~~("theInstructorId", 5);

// excute query

Instructor instructor=query.getSingleResult();

// load instructor and course all at once

System.***err***.println("Instructor : "+ instructor);

session.getTransaction().commit();

session.close();

**How load the courses at a later time in the application?**

getting related courses before closing session and using JOIN FETCH seem to be negating of lazy loading (using those solutions we completely resign of lazy loading.

**sloution**Yes, you can load it later with using a new session, just make use of HQL

get course later

   // create session factory  
        SessionFactory factory = new Configuration()  
                                .configure("hibernate.cfg.xml")  
                                .addAnnotatedClass(Instructor.class)  
                                .addAnnotatedClass(InstructorDetail.class)  
                                .addAnnotatedClass(Course.class)  
                                .buildSessionFactory();  
          
        // create session  
        Session session = factory.getCurrentSession();  
          
        try {              
              
            // start a transaction  
            session.beginTransaction();  
                          
            // get the instructor from db  
            int theId = 1;  
            Instructor tempInstructor = session.get(Instructor.class, theId);                      
              
            System.out.println("luv2code: Instructor: " + tempInstructor);      
              
            // commit transaction  
            session.getTransaction().commit();  
              
            // close the session  
            session.close();

            System.out.println("\nluv2code: The session is now closed!\n");

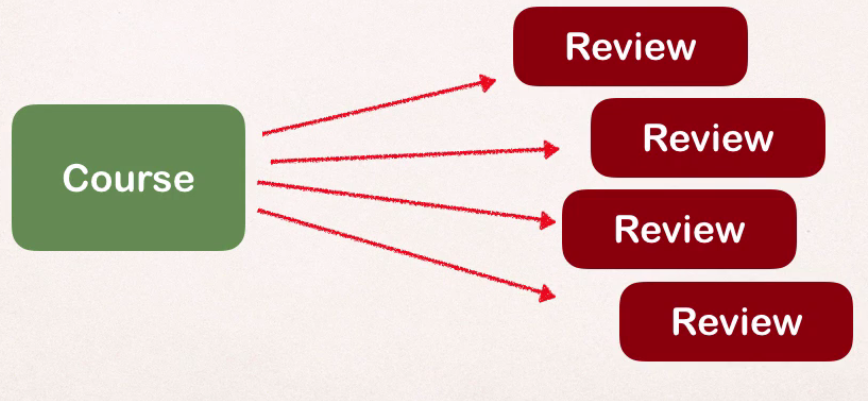
            //  
            // THIS HAPPENS SOMEWHERE ELSE / LATER IN THE PROGRAM

            // YOU NEED TO GET A NEW SESSION  
            //  
              
            System.out.println("\n\nluv2code: Opening a NEW session \n");

            session = factory.getCurrentSession();  
              
            session.beginTransaction();  
              
            // get courses for a given instructor  
            Query<Course> query = session.createQuery("select c from Course c "  
                                                    + "where c.instructor.id=:theInstructorId",      
                                                    Course.class);  
              
            query.setParameter("theInstructorId", theId);  
              
            List<Course> tempCourses = query.getResultList();  
              
            System.out.println("tempCourses: " + tempCourses);  
              
            // now assign to instructor object in memory  
            tempInstructor.setCourses(tempCourses);  
              
            System.out.println("luv2code: Courses: " + tempInstructor.getCourses());  
              
            session.getTransaction().commit();  
              
            System.out.println("luv2code: Done!");  
        }  
        finally {  
              
            // add clean up code  
            session.close();  
              
            factory.close();  
        }

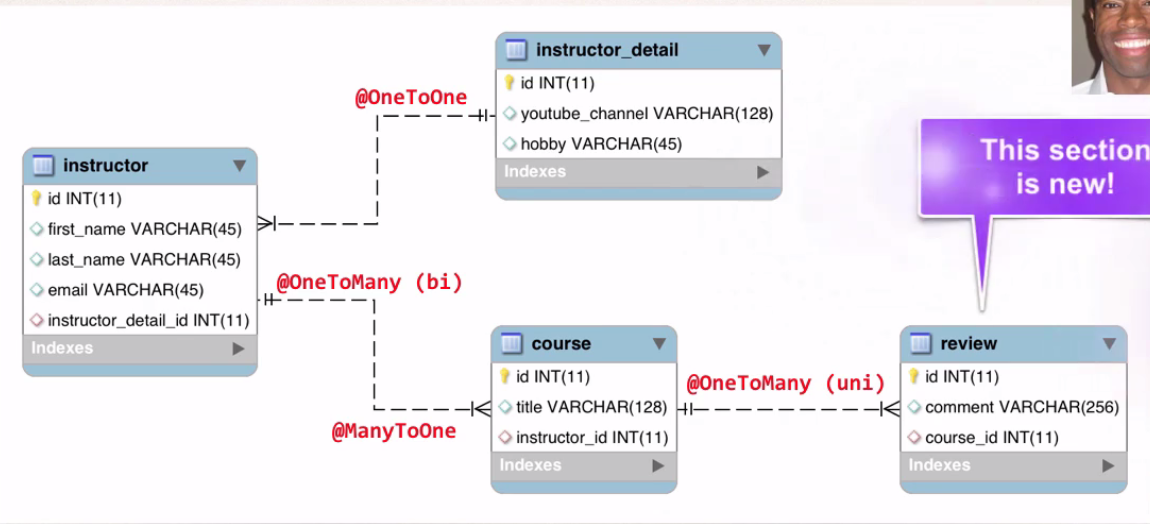
Hibernate OneTOMany UNi directional

* Course can have many reviews-uni directional
* Uni direction



Real world project requirement

* If u delete course also delete reviews
* Review with out course no meaing



Development process One to Many uni

1 define table

2 create review class

3 update Course

4 create main app

1 Defien Table

CREATE TABLE review (id serial PRIMARY KEY,comments VARCHAR(200), course\_id int , CONSTRAINT FK\_REVIEW FOREIGN KEY (course\_id) REFERENCES course(id))

2 create review Class

3 update Course class

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

@JoinColumn(name="course\_id") // refers to course\_id in review table

**private** List<Review> reviews;

More @JoinColumns

Look at the course\_id column in review table and find assoaicte review for a course

Add support for cascading

Cascade ALL bcz uni direction need to delete

Add lasy load support

Add convinece method

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

@JoinColumn(name="course\_id") // refers to course\_id in review table

**private** List<Review> reviews;

// add convein method to add reviews

**public** **void** add(Review review)

{

**if**(reviews==**null**)

{

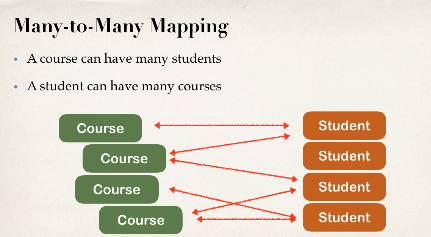
reviews=**new** ArrayList<Review>();

}

reviews.add(review);

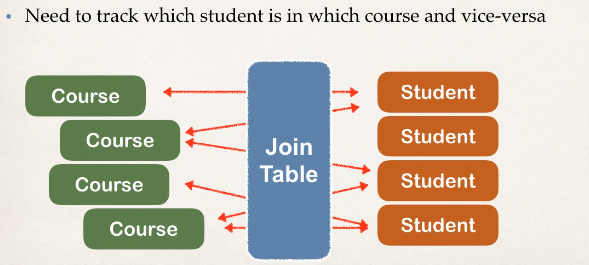
}

Many To Many



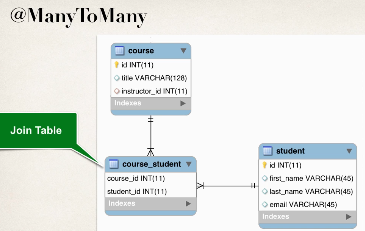
- need to track which studnet is in which course

Make use of join table



Join Table

Table that provide mapping between two tables , it has a foreign key for each table to define mapping relationship



Join table examples

Develeopment Process

1 define table

2 update couse table

3 update studnet table

4 main app

Join Table : course\_student;

CREATE table course\_student(course\_id int , student\_id int , PRIMARY KEY(course\_id,student\_id))

Join Table: course\_student - foreign keys

CREATE table course\_student(course\_id int , student\_id int , PRIMARY KEY(course\_id,student\_id),

CONSTRAINT FK\_COURSE\_STUDENT FOREIGN KEY (course\_id) REFERENCES course(id),

CONSTRAINT FK\_STUDENT\_COURSE FOREIGN KEY (student\_id) REFERENCES student(id)

)

2 update couse Entity

@ManyToMany(fetch=FetchType.***LAZY***,cascade= {CascadeType.***DETACH***,CascadeType.***MERGE***,CascadeType.***PERSIST***,CascadeType.***REFRESH***})

@JoinTable(name="course\_student",

joinColumns=@JoinColumn(name="course\_id"), // refere to course\_id column in course\_student join table

inverseJoinColumns=@JoinColumn(name="student\_id") // refere to student\_id column in course\_student join table

)

**private** List<Student> students;

// gettter and setter

// add convein method to add students

**public** **void** add(Student student)

{

**if**(students==**null**)

{

students=**new** ArrayList<Student>();

}

students.add(student);

}

@JoinTable

1 look at the course\_id column in course\_student join table

2 for the other side(inverse) look the student\_id column in the course\_student table

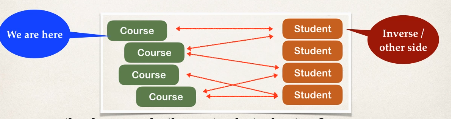
Use the information to find relation between course and student

Inverse:

In context defining relationship in the course class

The student class in another side so it is consdiered inverse

Invers refer to other side of the relationship



Lets do same way update student

3 Update Student

@ManyToMany(Student)

@JoinTable(name="course\_student",

joinColumns=@JoinColumn(name="student\_id"), // refere to student\_id column in course\_student join table

inverseJoinColumns=@JoinColumn(name="course\_id") // refere to course\_id column in course\_student join table

)

**private** List<Course> courses;

// gettet and setter

4 main APp

Real World porject

-If delete a course don’t delete student NO CASCADE DELETE

-lazy loadign of student classs

- cascading saves but not deletes

select courses0\_.student\_id as student\_1\_5\_0\_, courses0\_.course\_id as course\_i2\_5\_0\_, course1\_.id as id1\_0\_1\_, course1\_.instructor\_id as instruct3\_0\_1\_, course1\_.title as title2\_0\_1\_, instructor2\_.id as id1\_1\_2\_, instructor2\_.email as email2\_1\_2\_, instructor2\_.first\_name as first\_na3\_1\_2\_, instructor2\_.instructor\_detail\_id as instruct5\_1\_2\_, instructor2\_.last\_name as last\_nam4\_1\_2\_, instructor3\_.id as id1\_2\_3\_, instructor3\_.hobby as hobby2\_2\_3\_, instructor3\_.youtube\_channel as youtube\_3\_2\_3\_ from course\_student courses0\_ inner join basicdb.course course1\_ on courses0\_.course\_id=course1\_.id left outer join basicdb.instructor instructor2\_ on course1\_.instructor\_id=instructor2\_.id left outer join basicdb.instructor\_detail instructor3\_ on instructor2\_.instructor\_detail\_id=instructor3\_.id where courses0\_.student\_id=5

Delete Course

- confirm student not deleted

-only delete course\_student relationship

Delete Student

- confirm Course not deleted

-only delete course\_student relationship