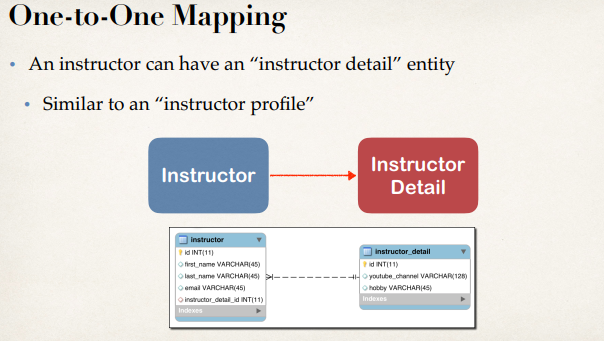
**23.1. @OneToOne – Overview**

**One-to-One Mapping**:

An instructor can have an “instructor detail” entity. That’s a One-to-One relationship. We will model this in the database using two separate tables.



**Uni-Directional**:

The above example is Uni-Directional, so we will start with the instructor and then we will have a one-way relationship with the instructor\_detail.

**Development process One-to-One**:

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

**1) Create table in database**:

* instructor\_detail
* instructor

**Table: instructor\_detail**:

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`)

) ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=latin1;

**Table: instructor**:

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,

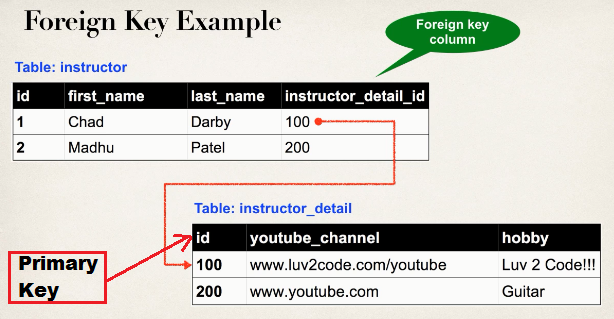
`instructor\_detail\_id` int(11) DEFAULT NULL,

PRIMARY KEY (`id`),

) ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=latin1;

**Foreign Key**:

* Link tables together
* A field in one table that refers to primary key in another table



**Define Foreign Key**:

CREATE TABLE `instructor` (

...

CONSTRAINT `FK\_DETAIL` FOREIGN KEY (`instructor\_detail\_id`)

REFERENCES `instructor\_detail` (`id`)

);

**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



**2) Create InstructorDetail class**:

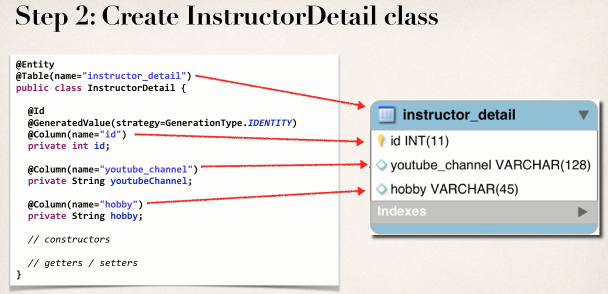
@Entity@Table(name="instructor\_detail")public class InstructorDetail {

@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}



**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;**

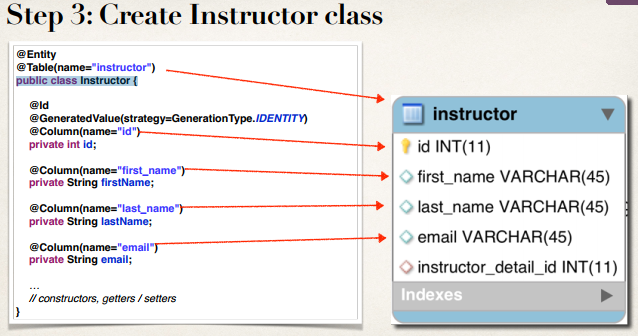
...

*// constructors,*

//*getters*

*// setters*

**}**



**Create Instructor class One-to-One**:

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

**...**

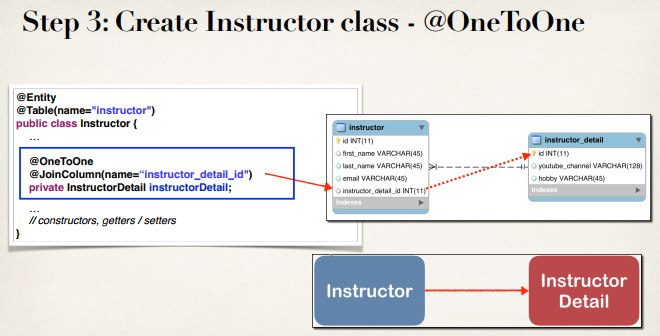
**@OneToOne  
 @JoinColumn(name=“instructor\_detail\_id")  
 private InstructorDetail instructorDetail;**

...

// constructors,

//getters

// setters**}**

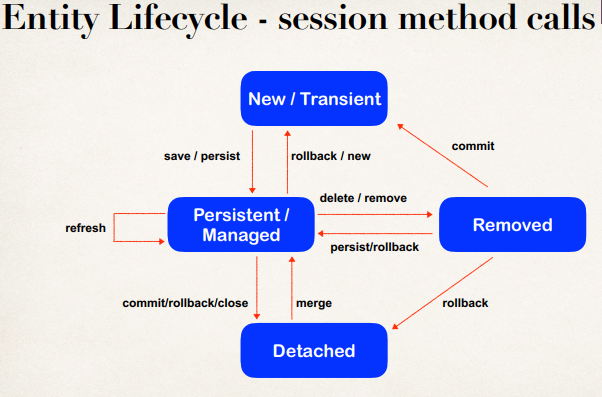


**Entity Lifecycle**:

The Entity lifecycle is basically a set of states that a Hibernate entity can go through when we using it in our application.

|  |  |
| --- | --- |
| 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 |

**Entity Lifecycle - session method calls**:



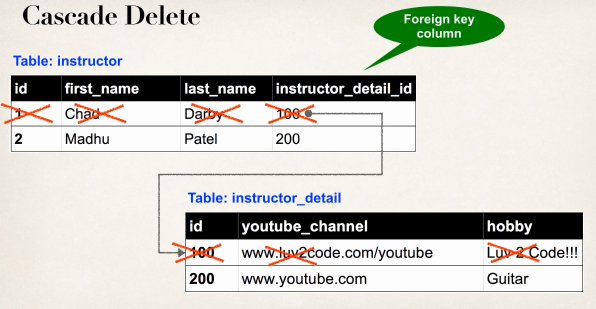
**Cascade**:

Cascade basically means we can apply the same operation to related entity

* We can cascade operations
* Apply the same operation to related entities

**Cascade Delete**:

When we delete the instructor then it will delete the appropriate row from the instructor\_detail. It will cascade the delete, apply the same operations on the given related entity.



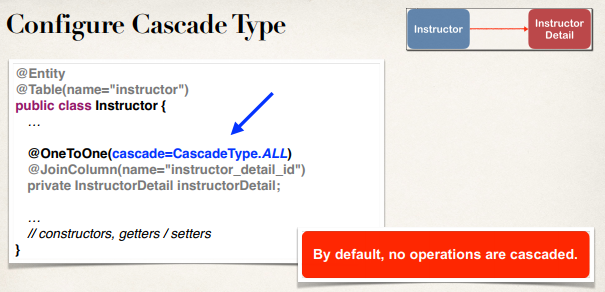
**@OneToOne – Cascade Types**:

As a developer, we can specify the actual cascade type that we want them to use. Here is the list of the cascade types.

|  |  |
| --- | --- |
| **Cascade Type** | **Description** |
| **PERSIST** | If entity is persisted/saved, related entity will also be persisted |
| **REMOVE** | If entity is removed / deleted, related entity will also be deleted |
| **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 |

**Configure Cascade Type**:

Add “**@OneToOne**” and then say “**cascade-cascadeType.ALL**”. That means that all operations we apply to our entity like “**instructor**”, will also cascade to our related entity, “**instructor\_detail**”. By default no operation are cascade. So if we don’t specify cascade then none of the operations will be cascade. So we have to explicitly reference a given cascade type that we would apply for our given relationship.



**Configure Multiple Cascade Types**:

We can configure multiple cascade types. If we want final grant control over which cascade types are applied, because we may not want all.

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

**4) Create Main App**:

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

**...  
// create the objects  
Instructor tempInstructor = new Instructor("Chad", "Darby", "darby@luv2code.com");**

**InstructorDetail tempInstructorDetail =  
new InstructorDetail(“http://www.luv2code.com/youtube", "Luv 2 code!!!");**

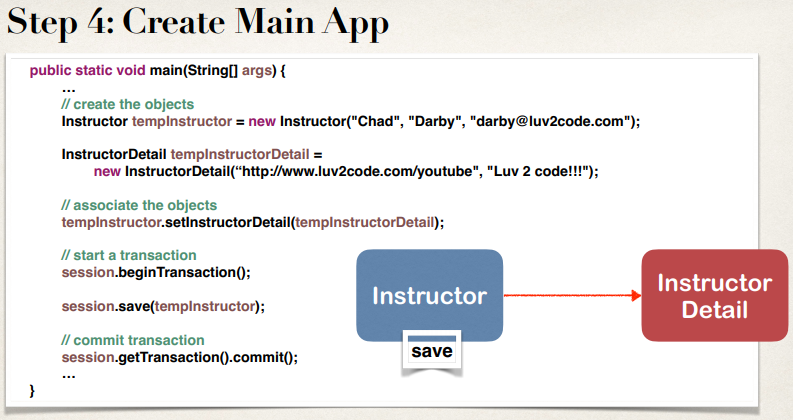
**// associate the objects  
tempInstructor.setInstructorDetail(tempInstructorDetail);**

**// start a transaction  
session.beginTransaction();**

**// save entity   
session.save(tempInstructor);**

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

**}**



23.1. @OneToOne – Overview1