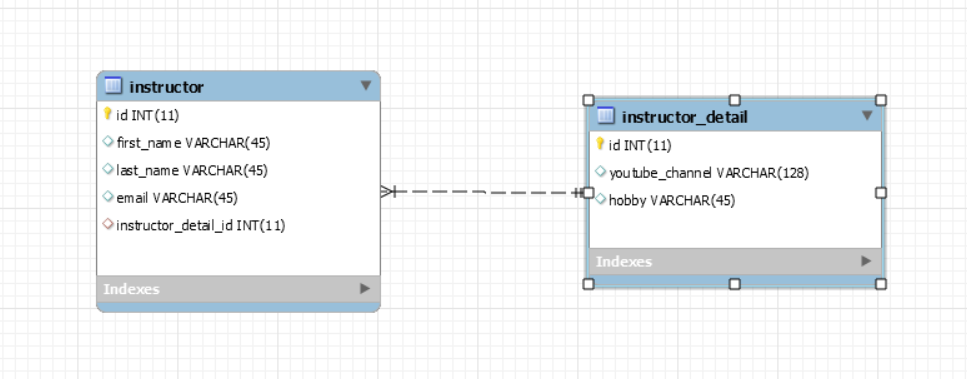
**Mapping**

1. **One to One Mapping**



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`),

KEY `FK\_DETAIL\_idx` (`instructor\_detail\_id`),

**CONSTRAINT `FK\_DETAIL` FOREIGN KEY (`instructor\_detail\_id`) REFERENCES `instructor\_detail` (`id`)** ON DELETE NO ACTION ON UPDATE NO ACTION

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

* **OneToOne UniDirectional**

1. Instructor.java

* @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;
* @OneToOne(cascade=CascadeType.ALL)
* @JoinColumn(name="instructor\_detail\_id")
* private InstructorDetail instructorDetail;
* //Create default constructor and parameterized constructor...generate getters and setters
* }

1. InstructorDetail.java

* @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 youtubeChanner;
* @Column(name="hobby")
* private String hobby;
* //Create default and parametized Constructor
* //generate getters and setters
* }

=> Here in OneToOne Uni directional , if we delete Instructor then Instructor details will also delete, but reverse is not possible. Reverse is possible through bi-directional mapping.

* **OneToOne Bi – Directional**

1. Instructor.java

* @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;
* @OneToOne(cascade=CascadeType.ALL)
* @JoinColumn(name="instructor\_detail\_id")
* private InstructorDetail instructorDetail;
* //Create default constructor and parameterized constructor...generate getters and setters
* }

1. **InstructorDetail.java**

* **@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 youtubeChanner;
* @Column(name="hobby")
* private String hobby;
* **//Create Object for Instructor.. //mappedBy="instructorDetail" refer to the instructorDetail property of the Instructor class**
* **@OneToOne(mappedBy="instructorDetail" , cascade=CascadeType.ALL)**
* private Instructor instructor;
* //Create default and parametized Constructor
* //generate getters and setters
* }
* @OneToOne(mappedBy="instructorDetail",
* **cascade={CascadeType.DETACH, CascadeType.MERGE, CascadeType.PERSIST,**
* **CascadeType.REFRESH}**)
* private Instructor instructor;

1. **One To Many Mapping**

* In OneToMany, an Instructor can have multiple courses.

1. Course.java

* @Entity
* @Table(name="course")
* public class Course {
* @Id
* @GeneratedValue(strategy=GenerationType.IDENTITY)
* @Column(name="id")
* private int id;
* @Column(name="title")
* private String title;
* //Many courses are mapped to one Instructor
* //In course table "instructor\_id" is the Foreign key
* @ManyToOne(cascade = {CascadeType.PERSIST , CascadeType.MERGE , CascadeType.DETACH ,CascadeType.REFRESH})
* @JoinColumn(name="instructor\_id")
* private Instructor instructor;
* }

1. Instructor.java

* @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;
* //OneToMany - One Instructor is having multiple Courses
* //mappedBy="instructor" refers instructor field in Course class
* @OneToMany(mappedBy="instructor" , cascade = {CascadeType.PERSIST ,CascadeType.DETACH ,CascadeType.MERGE , CascadeType.REFRESH})
* private List<Course> courses;
* **Eager Vs Lazy Loading**
* Eager = Retrieve everything
* Lazy = Retrieve on demand

1. **Eager Loading**

* Suppose an Instructor is having multiple courses. So, in Eager loading it will fetch all the Instructor and its courses at one short
* But Eager Loading will impact the system when we have huge number of data in Database. So we should avoid Eager loading
* Best practice is to use Lazy loading as it will fetch data on demand

1. **Lazy Loading**

* Lazy loading will load main entity first and dependent entity on demand
* Default Fetch Type

|  |  |  |
| --- | --- | --- |
| SL No | Mapping | Default Fetch Type |
| 1 | @OneToOne | FetchType.**Eager** |
| 2 | @OneToMany | FetchType.**Lazy** |
| 3 | @ManyToOne | FetchType.**Eager** |
| 4 | @ManyToMany | FetchType.**Lazy** |

* + To retrieve Lazy data , we need to open the Hibernate Session
  + Retrieve Lazy data using

I) session.get and then call appropriate getters method

Ii) Hibernate query with HQL

**Many to Many :-**