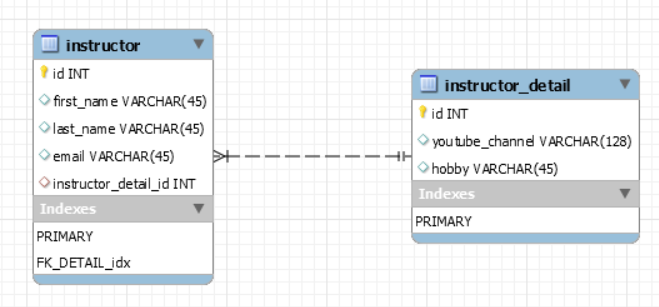
One-to-one Mapping

**Unidirectional Mapping**

1. Using a Foreign key association.



In this kind of association, a foreign key column is created in the **owner entity**. For example, if we have made **Instructor** owner, then an extra column "***instructor\_detail\_id***" will be created in the **instructor** table. This column will store the foreign key for the **instructor\_detail** table.

To make such an association, refer to the **Instructor\_Detail** entity in an **Instructor** entity class

@Entity

@Table(name = "instructor")

public class Instructor {

// annotate a class as an entity and map to db table

// define the fields

// annotate the fields with table column names

// \*\* Setup mapping to InstructorDetails entiry

// create constructors

// generate setters and getters

// generate to string method

@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 InstructorDetails instructorDetails;

}

Also, **we need to place the *@JoinColumn*annotation** to configure the name of the column in the *instructor* table that maps to the primary key in the *instructor\_details* table. If we don't provide a name, hibernate will follow some rules to select a default one.

Finally, note in the next entity that we won't use the *@JoinColumn*annotation there. This is because we only need it on the owning side of the foreign key relationship. **Simply put, whoever owns the foreign key column gets the *@JoinColumn* annotation.**

**If no @JoinColumn is declared on the owner side, the defaults apply.**A join column(s) will be created in the owner table and its name will be the concatenation of the name of the relationship in the owner side, \_ (underscore), and the name of the primary key column(s) in the owned side.

@Entity

@Table(name = "instructor\_detail")

public class InstructorDetails {

// annotate a class as an entity and map to DB table

// define the fields

// annotate the fields with table column names

// create constructors

// generate setters and getters

// generate to string method

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id")

private int id;

@Column(name = "youtube\_channel")

private String youtubeChannel;

@Column(name = "hobby")

private String hobby;

}

1. Using a shared Primary key.
2. Using a Join Table.

Bidirectional

To make it bidirectional, We also need to place the *@OneToOne*annotation here too. **The InstructorDetails side of the relationship is called the non-owning side.**

MappedBy signals hibernate that the key for the relationship is on the other side.

This means that although you link 2 tables together, only 1 of those tables has a foreign key constraint to the other one. MappedBy allows you to still link from the table not containing the constraint to the other table.

**Note**:- Mapped by can be used only in one class because one table must contain foreign key constraint. if mapped by can be applied on both side then it remove the foreign key from both tables and without a foreign key there is no relation b/w two tables.

@Entity

@Table(name = "instructor\_detail")

public class InstructorDetails {

// annotate a class as an entity and map to db table

// define the fields

// annotate the fields with table column names

// create constructors

// generate setters and getters

// generate to string method

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id")

private int id;

@Column(name = "youtube\_channel")

private String youtubeChannel;

@Column(name = "hobby")

private String hobby;

// add new field for instructor

// add one to one

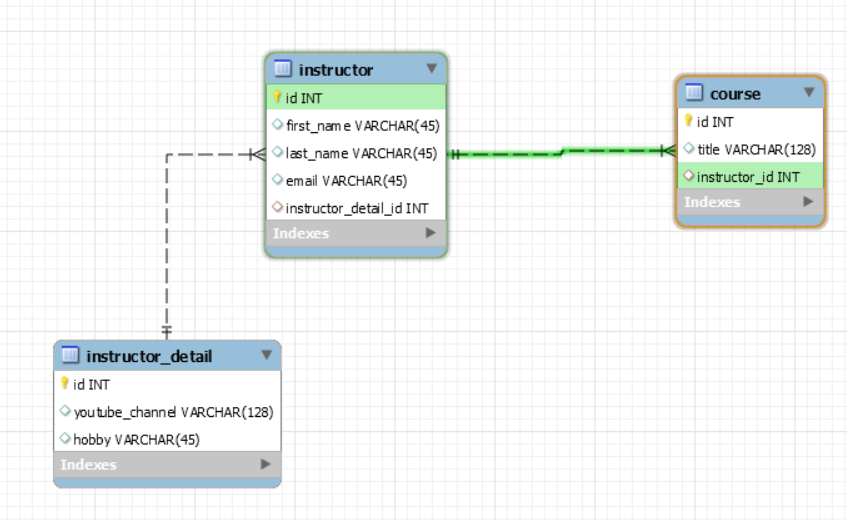
**@OneToOne(mappedBy = "instructorDetails", cascade = CascadeType.ALL)**

**private Instructor instructor;**

}

**One-to-many / many-to-one**

**Bidirectional**



@Entity

@Table(name = "instructor")

public class Instructor {

// annotate a class as an entity and map to db table

// define the fields

// annotate the fields with table column names

// \*\* Setup mapping to InstructorDetails entiry

// create constructors

// generate setters and getters

// generate to string method

@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 InstructorDetails instructorDetails;

**@OneToMany(mappedBy = "instructor", cascade = { CascadeType.DETACH, CascadeType.PERSIST, CascadeType.MERGE,CascadeType.REFRESH })**

**private List<Course> courses;**

}

@Entity

@Table(name = "course")

public class Course {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id")

private int id;

@Column(name = "title")

private String title;

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

**@JoinColumn(name = "instructor\_id")**

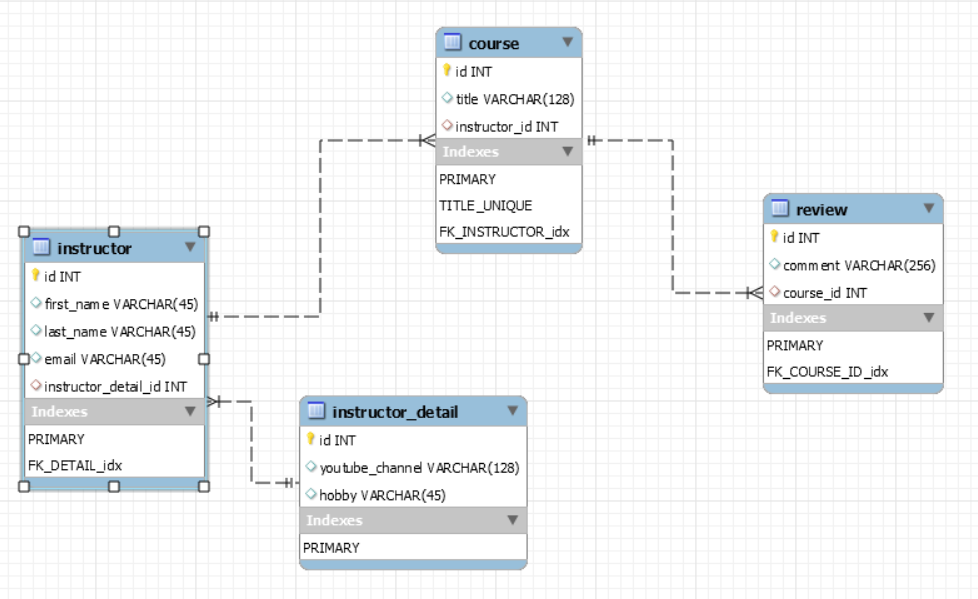
private Instructor instructor;

}

One instructor may have multiple courses. So, in the ***course******entity*** class, we have ***Instructor*** property for which we are adding ***@JoinColumn annotation as it’s a foreign key*** hence course entity is owning the relationship (Many courses point to a single entity that is the reason **@ManyToOne** annotation).

On the other side, the ***Instructor***, entity contains the list of courses (many courses may be thought by a single instructor that’s the reason for the **@OneToMany** annotation). The mappedBy attribute characterizes a bidirectional association and must be set on the parent-side. In other words, for a bidirectional @OneToMany association, **set mappedBy to @OneToMany on the parent-side and add @ManyToOne on the child-side referenced by mappedBy**. Via mappedBy, the bidirectional @OneToMany association signals that it mirrors the @ManyToOne child-side mapping.

One-to-Many Unidirectional



@Entity

@Table(name = "course")

public class Course {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id")

private int id;

@Column(name = "title")

private String title;

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

@JoinColumn(name = "instructor\_id")

private Instructor instructor;

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

**@JoinColumn(name = "course\_id")**

**private List<Review> reviews;**

}

@Entity

@Table(name = "review")

public class Review {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id")

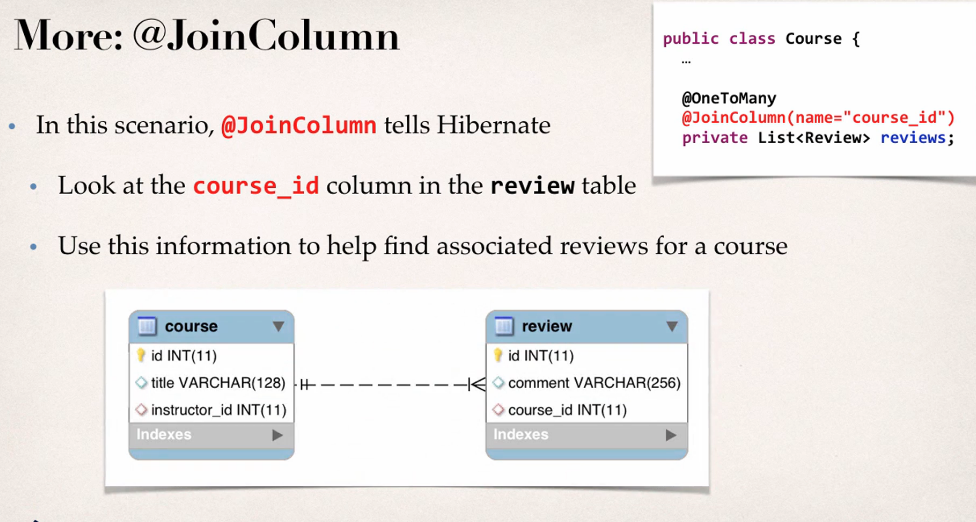
private int id;

@Column(name = "comment")

private String comment;

}

We have introduced a new entity Review. One course may have many reviews. Hence the @OneToMany mapping on list of courses on course entity.



Many to Many