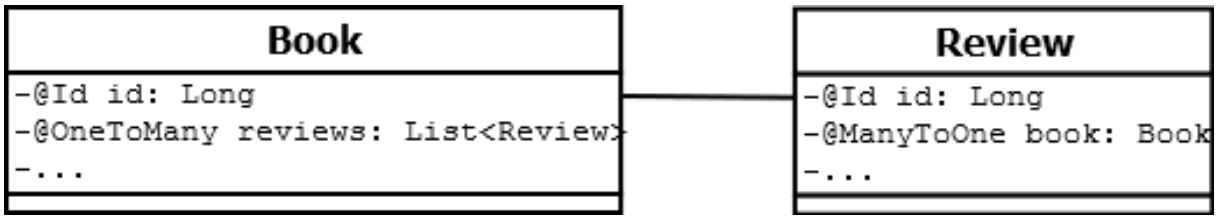
Hibernate Tip: Map a bidirectional many-to-one association

Question:

My table model contains a many-to-one association. How do I model it with Hibernate so that I can navigate it in both directions?

Solution:

You need to model the association on both entities if you want to be able to navigate it in both directions. Consider this example. A book in an online bookstore can have multiple reviews. In your domain model, the *Book* entity has a one-to-many association to the *Review* entity, and the *Review* entity has a many-to-one relationship to the *Book* entity.



Let's begin with the *Review* entity, which is the owning side of the association in this example. That means that it defines the association and the *Book* entity just references it. The relationship consists of two mandatory and one optional part. The entity attribute of type *Book* and the *@ManyToOne* annotation are required. The attribute models the association, and the annotation declares the type of relationship. The *@JoinColumn* annotation is optional. It allows you to define the name of the foreign key column. I use it in the following code snippet to set the column name to *fk_book*.

If you don't define the name yourself, Hibernate generates a name by combining the name of the association mapping attribute and the name of the primary key attribute of the associated entity. In this example, Hibernate would use *book_id* as the default column name.

```
@Entity
public class Review {

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	@Column(name = “id”, updatable = false, nullable =
false)
	private Long id;

	@ManyToOne
	@JoinColumn(name = “fk_book”)
	private Book book;
	…
}
```

You also need to map the one-to-many association on the *Book* entity to make it bidirectional. As you can see in the following code snippet, this is done in a similar way as the many-to-one association.

```
@Entity
public class Book {

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	@Column(name = “id”, updatable = false, nullable =
false)
```

```
        private Long id;

        @OneToMany(mappedBy = “book”)
        private List reviews = new ArrayList();
        …
}
```

You need an attribute that models the association, which is the *List reviews* attribute in this example and a *@OneToMany* annotation. Like in the table model, the bidirectional one-to-many association gets defined on the many side. The table on the many side stores the foreign key and its entity defines the association. It's similar for the entity mapping. You just need to reference the name of the association attribute of the many side as the value of the *mappedBy* attribute and Hibernate has all the information it needs.

That's all you need to do to define a bidirectional many-to-one association. You can now navigate it in both directions in your JPQL or Criteria API queries or on your domain objects.

```
b = em.find(Book.class, 1L);

List reviews = b.getReviews();
Assert.assertEquals(b, reviews.get(0).getBook());
```

Bidirectional associations are easy to use in queries, but they also require an additional step when you persist a new entity. You need to update the association on both sides when you add or remove an entity. You can see an example of it in the following code snippet, in which I first create a new *Review* entity and initialize its association to the *Book* entity. And after that, I also need to add the new *Review* entity to the *List* of reviews on the *Book* entity.

```
Book b = em.find(Book.class, 1L);

Review r = new Review();
r.setComment("This is a comment");
r.setBook(b);

b.getReviews().add(r);

em.persist(r);
```

Updating the associations on both entities is an error-prone task. Therefore, it's a good practice to provide a helper method that adds another entity to the many side of the association.

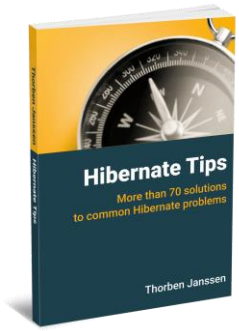
```
@Entity
public class Book {
    …

    public void addReview(Review review) {
```

```
            this.reviews.add(review);
            review.setBook(this);
        }
        …
}
```

Learn more

Bidirectional many-to-one associations are just one way to model relationships between entities. I show you how to model a bi-directional many-to-many association in How to map a bidirectional many-to-many association

Hibernate Tips Book



Get more recipes like this one in my book Hibernate Tips: More than 70 solutions to common Hibernate problems.

It gives you more than 70 ready-to-use recipes for topics like basic and advanced mappings, logging, Java 8 support, caching and statically and dynamically defined queries.