Best Practices for Many-to-Many Associations

Many-to-Many associations are one of the most commonly used associations with JPA and Hibernate. You can find lots of examples for them in the real world, and you can map them with JPA and Hibernate as a uni- or bidirectional association in your domain model.

But you probably also know that these mappings provide several pitfalls. In this article, I will show you 5 best practices that will help you to avoid these pitfalls and to implement efficient mappings.

1. The most efficient data type for your association

Hibernate handles remove operations on Many-to-Many relationships that are mapped to a java.util.List very inefficiently. It first removes all records from the association table before it inserts all remaining ones.

You should instead model a many-to-many association as a java.util.Set.

```
@Entity
public class Book {


    @ManyToMany
    @JoinTable(name = "book_author",
            joinColumns = { @JoinColumn(name = "fk_book") },
            inverseJoinColumns = { @JoinColumn(name = "fk_author") })
    private Set authors = new HashSet();


    ...
}
```

2. Why you need utility methods to manage your association

Bidirectional associations are mapped to an entity attribute on both ends of the relationships. If you add or remove an association, you always need to perform the change on both ends of the association.

Utility methods your entities make updating and removing much easier. Within this methods, you perform the required operations on both entities.

```
@Entity
public class Author {

    @ManyToMany(mappedBy = "authors")
    private List<Book> books = new ArrayList<Book>();

    ...

    public void addBook(Book book) {
        this.books.add(book);
        book.getAuthors().add(this);
    }

    public void removeBook(Book book) {
        this.books.remove(book);
        book.getAuthors().remove(this);
    }
}
```

3. The right FetchType for an efficient mapping

You should always use FetchType.LAZY for your many-to-many associations. It tells your persistence provider not to fetch the associated entities from the database until you use them. That's usually the case when you call its getter method for the first time.

Luckily, that's the default for all to-many associations. So, please make sure that you don't change it.

4. When and how to use query-specific fetching

When you load an entity and use query-specific fetching, you tell Hibernate which mapped associations it shall initialize for each fetched entity. It then extends the SELECT clause of your query so that it includes the columns mapped by these other entities and initializes the associations.

You can implement query-specific fetching in several different ways. The simplest one is a JOIN FETCH clause.

```
Author a = em.createQuery(
        "SELECT a FROM Author a JOIN FETCH a.books “
        + “WHERE a.id = 1", Author.class).getSingleResult();
```

It not only gets translated into a SQL JOIN, as it's the case for a JPQL JOIN clause, it also forces your persistence provider to extend the SELECT clause by all columns that are mapped by the associated entity.

```
16:21:03,046 DEBUG SQL:94 -
    select
        author0_.id as id1_0_0_,
        book2_.id as id1_1_1_,
        author0_.firstName as firstNam2_0_0_,
        author0_.lastName as lastName3_0_0_,
        author0_.version as version4_0_0_,
```

```
        book2_.format as format2_1_1_,
        book2_.publishingDate as publishi3_1_1_,
        book2_.title as title4_1_1_,
        book2_.version as version5_1_1_,
        books1_.author_id as author_i2_2_0__,
        books1_.book_id as book_id1_2_0__
    from
        Author author0_
    inner join
        book_author books1_
            on author0_.id=books1_.author_id
    inner join
        Book book2_
            on books1_.book_id=book2_.id
    where
        author0_.id=1
```

5. The CascadeType you should avoid at all costs

If you activate cascading on an association, your persistence provider applies the operations you perform on the entity to all associated entities. If it does that for all operations or just for a few selected ones depends on the configured CascadeType.

You should avoid the CascadeTypes REMOVE and ALL, which includes REMOVE, for many-to-many associations. In the best case, it only creates performance issues, but in the worst case, it might also remove more records than you intended.

So, better remove associated entities programmatically.