Hibernate Tip: Map the latest element of an association

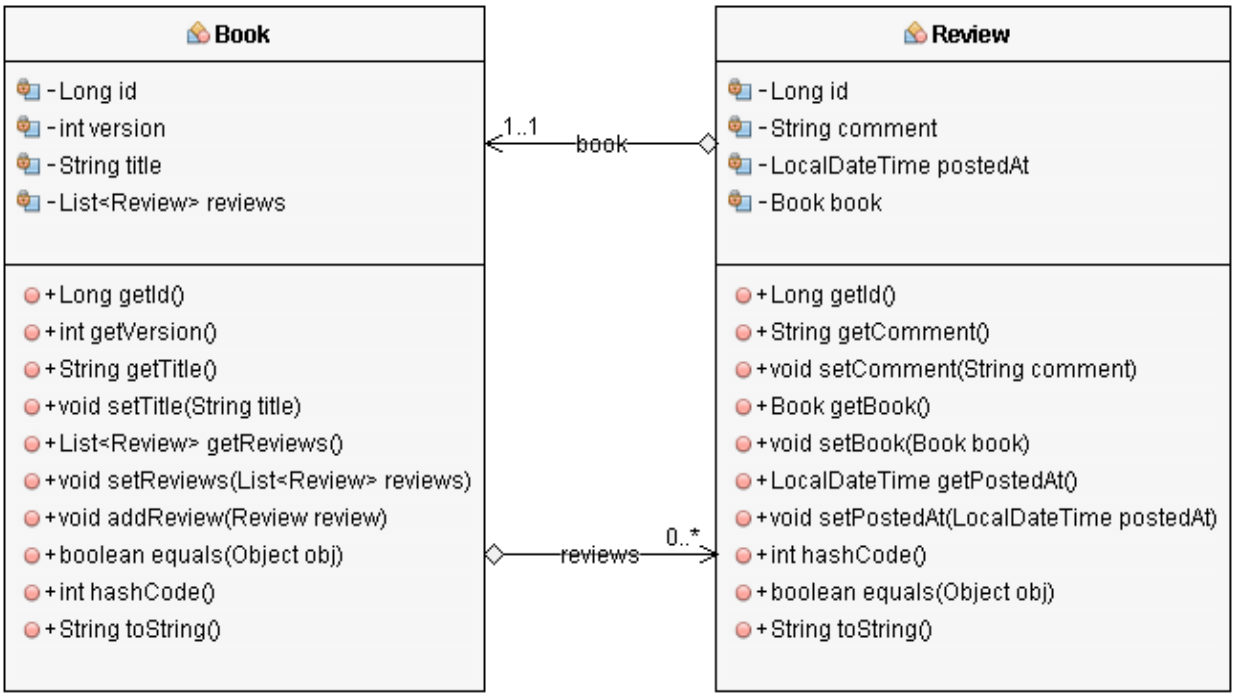
Question:

I want to map the latest element of a to-many association as an entity attribute. How can I map that with Hibernate?

Solution:

Mapping the latest element of an association to an entity attribute is a little bit tricky and creates an overhead which you should avoid in most situations. If you don't use that specific element in most of your use cases, you should prefer a JPQL query to get the latest associated element from the database.

Let's take a look at both options. I will show them based on the following, simple model which you probably know from a typical online store. It consists of a *Book* entity, a *Review* entity and a one-to-many association between them.



Entity Mappings:

The mappings of both entities are relatively simple. The *Review*entity maps the primary key column *id*, the *comment* as a simple *String*, the date and time when the review was published as a *LocalDateTime*, and it also defines a many-to-one association to the *Book* entity.
The *@CreationTimestamp* annotation, which I use with the *postedAt* attribute, is Hibernate-specific. It tells Hibernate to set the value of this attribute to the current timestamp when it persists a new *Review* entity. So, you don't need to set the creation timestamp yourself.

```
@Entity
public class Review {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private String comment;

    @CreationTimestamp
    private LocalDateTime postedAt;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "book_id")
    private Book book;

    ...
}
```

The mapping of the *Book* entity is even simpler. It maps the *id*attribute as the primary key, the *title* as a simple *String* and the other end of the association to the *Review* entity.

```
@Entity
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private String title;

    @OneToMany(mappedBy = "book")
    private List<Review> reviews = new ArrayList<Review>();

    ...
}
```

OK, so let's take a look at a JPQL query, which returns the latest *Review* of a given *Book*.

Getting the Latest Element as a JPQL Query:

This query is relatively simple and should be the preferred approach if only a few of your use cases use that *Review*. You can either use a function to select the entity with the highest *postedAt* timestamp, or you simply order the associated reviews by their *postedAt* date and limit the result list to the first element. I chose the latter approach in this example.

```
TypedQuery<Review> q = em.createQuery("SELECT r
FROM Review r WHERE r.book.id = :bid ORDER BY
postedAt DESC", Review.class);
q.setMaxResults(1);
q.setParameter("bid", 1L);
Review r = q.getSingleResult();
```

Mapping the latest element to an attribute:

The mapping to an attribute is more complicated and will affect how Hibernate loads all *Book* entities. You should only use it if all of your uses cases require the latest *Review*. Otherwise, you should select it with a JPQL query.

You probably expected to model this as a one-to-one association with additional join criteria. That's what I intended to do, but unfortunately, there is a bug in Hibernate 5.2 which prevents you from using a *@JoinFormula* annotation with a one-to-one association. You need to model it as a many-to-one association, instead.

The *@JoinFormula* annotation is Hibernate-specific and enables you to provide a native SQL snippet as the JOIN criteria of an association. Hibernate includes this snippet into the generated query and compares its return value with the primary key of the associated entity.

So, in this example, my SQL snippet needs to return the *id* of a *Review*. As you can see in the following code snippet, I'm using the same idea as in the previous example. I select the *id* of all *review*records which *book_id* is equal to the *id* of the selected *Book*, order the records by their *postedAt* value and limit the result set to the first record.

```
@Entity
public class Book {

    @ManyToOne
    @JoinFormula("(SELECT r.id FROM review r WHERE
r.book_id = id ORDER BY r.postedAt DESC LIMIT 1)")
    private Review latestReview;

    ...
}
```

When I now select a *Book* entity from the database, Hibernate includes the SQL snippet defined in the *@JoinFormula* annotation and maps the latest, associated *Review* entity to the *latestReview*attribute.

Here you can see the generated SQL statement. Unfortunately, Hibernate includes the SQL snippet in the SELECT and the FROM clause of the query. Depending on the SQL snippet and the optimization performed by your database, this might slow down your application significantly.

```
10:50:54,400 DEBUG [org.hibernate.SQL] -
    select
        book0_.id as id1_0_0_,
        book0_.title as title2_0_0_,
        book0_.version as version3_0_0_,
        (SELECT
            r.id
        FROM
            review r
        where
            r.book_id = book0_.id
        ORDER BY
            r.postedAt DESC LIMIT 1) as formula1_0_,
        review1_.id as id1_1_1_,
        review1_.book_id as book_id4_1_1_,
        review1_.comment as comment2_1_1_,
        review1_.postedAt as postedAt3_1_1_
    from
        Book book0_
    left outer join
        Review review1_
            on (
                SELECT
                    r.id
            FROM
```

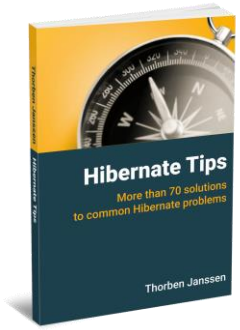
```
review r
            where
                r.book_id = book0_.id
            ORDER BY
                r.postedAt DESC LIMIT 1
        )=review1_.id
    where
        book0_.id=?
```

Learn more

If you enjoyed this post, you might also be interested in the following posts about association mappings:

- Ultimate Guide to Association Mappings with JPA and Hibernate
- Best Practices for Many-To-One and One-To-Many Association Mappings
- How to map an association as a java.util.Map
- How to Choose the Most Efficient Data Type for To-Many Associations – Bag vs. List vs. Set

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.