Hibernate Tip: Map an entity attribute to an Optional<T>

Question:

Some of my entity attributes are optional. Is there any way to map them to an *Optional<T>*?

Solution:

Since the release of Java 8, I get this question quite often. Unfortunately, Hibernate and JPA 2.2 don't support *Optional* as an attribute type.

But with a little trick, you can still use *Optional* as the return type of your getter methods. If you annotate the primary key attribute with an *@Id* annotation, you tell Hibernate to use field access to set and retrieve the entity attribute values. That means that Hibernate doesn't call the getter or setter methods and that you can implement them in any way you like. So, you can wrap the optional attribute into an *Optional* instead of returning it directly.

But please be aware, that this doesn't include any lazy loading and just wraps the already selected value of the database column into an *Optional*. Hibernate requires bytecode enhancement to enable lazy loading for basic attributes. That's far beyond the scope of this article but I explain it in more details in my Hibernate Performance Tuning Online Training.

Example:

OK, let's take a look at a simple example. A book might have been published, or the author is still working on it and will announce the publishing date soon. So, the *publishingDate* attribute of a *Book* entity can be null. The following code snippet shows the definition of the *Book* entity.

Hibernate Tip: Map an entity attribute to an Optional<T>

```
@Entity
public class Book {

    @Id
    @GeneratedValue(strategy =
GenerationType.SEQUENCE)
    @Column(name = "id", updatable = false, nullable = false)
    private Long id;

    @Column(nullable = true)
    private LocalDate publishingDate;

    ...

    public Optional getPublishingDate() {
        return Optional.ofNullable(publishingDate);
    }

    public void setPublishingDate(LocalDate publishingDate) {
        this.publishingDate = publishingDate;
    }
}
```

I annotated the *id* attribute with *@Id* so that Hibernate uses field access. I also use a *@GeneratedValue* value annotation to tell Hibernate to use a sequence to generate the primary key value.

The *publishingDate* attribute is of type *LocalDate*. Since JPA 2.2 and Hibernate 5 support *LocalDate* as a basic type, there is no need to use the old *java.util.Date* anymore. And the *getPublishingDate* method wraps that attribute into an *Optional*.

Learn more

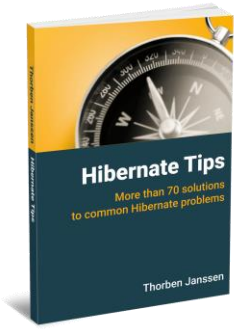
You can use the same approach to map optional to-one associations to an *Optional*: Hibernate Tip: How to map an association to an *Optional*

And if you want to use more Java 8 features in your persistence layer, take a look at the following posts:

- JPA 2.2
  - JPA 2.2's *getResultStream()* method and how you should NOT use it
  - How to Map the Date and Time API with JPA 2.2
  - JPA 2.2 Introduces *@Repeatable* Annotations

- Hibernate 5
  - How to persist LocalDateTime & Co with Hibernate 5
  - How to get query results as a *Stream* with Hibernate 5.2
  - Benefits of *@Repeatable* annotations in Hibernate 5.2

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.