Hibernate Tip: Map Multiple Entities to the Same Table

Question:

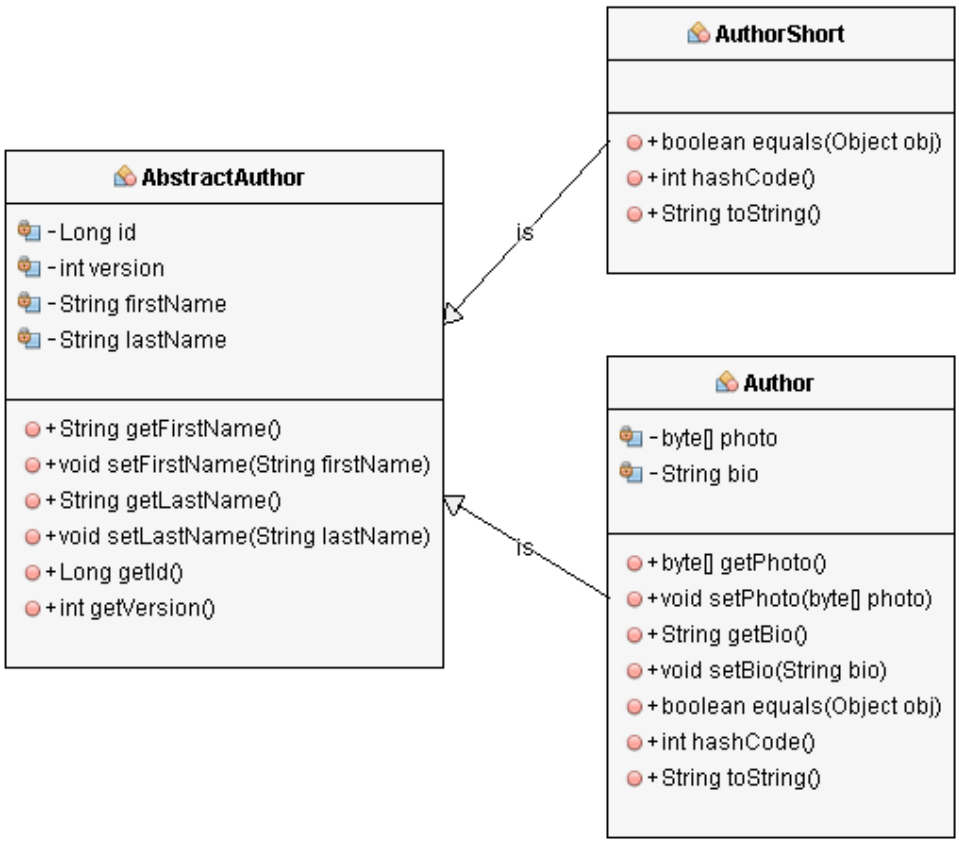
Some of our entities have a lot of attributes and we don't need all of them in all use cases. Can I create a second entity that only maps a subset of the most commonly used attributes? Is there anything I should consider?

Solution:

Yes, you can map two or more entities to the same database table. But you should only do that if you will use all of these entities to perform write operations or to define associations between entities. If that's not the case, you should better use a DTO projection.

Entity Mappings

If you want to map the same database table to two entities, you should create a simple inheritance hierarchy.



The superclass should be abstract and contain all attributes that are shared by both entities. You should map it as a mapped superclass so that it is not an entity itself.

```
@MappedSuperclass
public class AbstractAuthor {

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

    @Version
    private int version;

    private String firstName;

    private String lastName;

    ...
}
```

The class *AuthorShort* maps only the most commonly used attributes. It extends the superclass and doesn't add any attributes itself.

By default, Hibernate would map this entity to the database table *AuthorShort*. You can override the default table mapping with the *@Table* annotation and specify the table name in the *name* attribute.

```
@Entity
@Table(name = "author")
public class AuthorShort extends AbstractAuthor {}
```

And the *Author* entity maps all columns of the *author* table. So, it needs to provide the mapping information of the not commonly used attributes that are not already defined by the *AbstractAuthor* class.

```
@Entity
public class Author extends AbstractAuthor {

    private byte[] photo;

    private String bio;

    ...
}
```

That's all you need to do to map 2 entities to the same database table. But there are a few things you need to keep in mind before you implement your use cases.

Potential Problems

Even so, you told Hibernate to map both entities to the same database table; Hibernate doesn't know which *Author* and *AuthorShort* entities represent the same database record. That has a few implications:

- Hibernate doesn't refresh any of these entities if you update the other one. So, you need to either take care of that yourself or make one of the entities immutable.
- Hibernate stores all managed entities in the 1st level cache and delays all write operations until it flushes the persistence context. This happens transparently so that you normally don't recognize it.
  But that changes when you map 2 entities to the same database

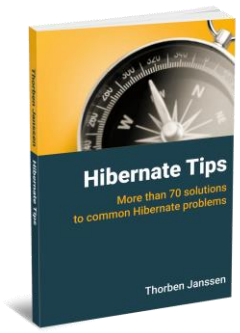
table. A call of the *EntityManager.find* method to load an *AuthorShort* entity doesn't trigger a flush of the *Author* entities. So, Hibernate doesn't write the pending changes to the database and the *find* method might return an outdated entity. You can avoid that by using a JPQL query.

- Hibernate handles the *Author* and *AuthorShort* entities independently of each other. Hibernate will throw an *OptimisticLockException* if you update an *AuthorShort* and an *Author* entity that map the same database record within the same transaction.

Learn more

You can learn more about advanced entity mappings in my Hibernate Advanced Online Training.

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.