Why @OneToMany does not work with inheritance in Hibernate

Asked 16 years, 3 months ago Modified 4 years, 6 months ago Viewed 31k times

13

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
public class Problem {
    @ManyToOne
    private Person person;
}

@Entity
@DiscriminatorValue("UP")
public class UglyProblem extends Problem {}

@Entity
public class Person {
    @OneToMany(mappedBy="person")
    private List< UglyProblem > problems;
}
```

I think it is pretty clear what I am trying to do. I expect @ManyToOne person to be inherited by UglyProblem class. But there will be an exception saying something like: "There is no such property found in UglyProblem class (mappedBy="person")".

All I found is this. I was not able to find the post by Emmanuel Bernard explaining reasons behind this.

> Unfortunately, according to the Hibernate documentation "Properties from superclasses not mapped as @MappedSuperclass are ignored."

Well I think this means that if I have these two classes:

```
public class A {
    private int foo;
}

@Entity
public class B extens A {
}
```

then field `foo` will not be mapped for class B. Which makes sense. But if I have something like this:

```
@Entity
public class Problem {

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

private String name;

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}
}

@Entity
public class UglyProblem extends Problem {

private int levelOfUgliness;

public int getLevelOfUgliness() {
    return levelOfUgliness;
}

public void setLevelOfUgliness(int levelOfUgliness) {
    this.levelOfUgliness = levelOfUgliness;
}
}
```

I expect the class UglyProblem to have fileds `id` and `name` and both classes to be mapped using same table. (In fact, this is exactly what happens, I have just checked again). I have got this table:

```
CREATE TABLE "problem" (
    "DTYPE" varchar(31) NOT NULL,
    "id" bigint(20) NOT NULL auto_increment,
    "name" varchar(255) default NULL,
    "levelOfUgliness" int(11) default NULL,
    PRIMARY KEY  ("id")
) AUTO_INCREMENT=2;
```

Going back to my question:

> I expect @ManyToOne person to be inherited by UglyProblem class.

I expect that because all other mapped fields are inherited and I do not see any reason to make this exception for ManyToOne relationships.

Yeah, I saw that. In fact, I used Read-Only solution for my case. But my question was "Why..." :). I know that there is an explanation given by a member of hibernate team. I was not able to find it and that is why I asked.

I want to find out the motivation of this design decision.

(if you interested how I have faced this problem: I inherited a project built using hibernate 3. It was Jboss 4.0.something + hibernate was already there (you'd download it all together). I was moving this project to Jboss 4.2.2 and I found out that there are inherited mappings of "@OneToMany mappedBy" and it worked fine on old setup...)

java hibernate inheritance orm

Share
Improve this question
Follow

edited Jul 13, 2013 at 12:52
Bill the Lizard
405k ● 211 ● 572 ● 889

asked Sep 1, 2008 at 15:21
Georgy Bolyuba
8,513 ● 7 ● 30 ● 39

You should add more details about your edit on September 4. Specifying List<UglyProblem> instead of only List in the @OneToMany(mappedBy="person") mapping changes the nature of the problem, since I think we previously assumed you wanted to map a Person to a list of Problems (not Ugl – David Crow Sep 12, 2008 at 1:02

6 Answers

Sorted by: Highest score (default)

11

In my case I wanted to use the SINGLE_TABLE inheritance type, so using @MappedSuperclass wasn't an option.

What works, although not very clean, is to add the Hibernate proprietary @Where clause to the @OneToMany association to force the type in queries:

```
@OneToMany(mappedBy="person")
@Where(clause="DTYPE='UP'")
private List< UglyProblem > problems;
```

Share

Improve this answer

Follow

edited Apr 8, 2015 at 13:51

degr
1,565 ● 1 ● 19 ● 42

answered Sep 29, 2014 at 0:44

Guillaume Carre
140 ● 1 ● 5

Alternatives to `@Where` for eclipseLink or JPA in general would be much appreciated – Odys Dec 1, 2016 at 17:18

Hi Odys, in EclipseLink you simply do not need to specify anything. Hibernate normally forces you to specify in the oneToMany annotation the targetEntity=Person.class. So when you have Entity with field1. UglyPerson uglyPerson; and field2 BeutifulPerfoson beutifulPerson; In eclipselink, he automatically knows to look at your BeuftifulPerson @DescriminatorValue("PRETTY_PERSON") and fetch the right stuff. Hibernate does not know this, and forces you to use the @WHERE clause. Eclipselink is smarted analysing a class hiearchy and resolving the oneToMany mapping. – 99Sono Feb 22, 2017 at 11:01

In Ecliipselink just make sure that your entities that are part of a class hierarhcy each has its own appropriate; @DiscriminatorValue("BEAUTIFUL_PERON") descriminator and add to the base class if your entity something like: @DiscriminatorColumn(name = "ENTITY_CLASS", descriminatorType=string) and you can easily use @Inheritance(SINGLE_TABLE) With Hibernate, you are quite stuck on this, as you need to specify a WHERE clasuse, and continuously maintain the predicate with the ENTITY_CLASS for each child element in the hierarchy. So here eclipselink is way ahead. – 99Sono Feb 22, 2017 at 11:33

@Where helped me out when I was trying to map a collection twice. I did this to separate child entities with a certain property into two collections. Example: `@OneToMany @JoinColumn(name = 'FLEET_ID') @Where(clause = "COLOR='blue'") Set<Car> blueCars;` – andy Jul 2, 2019 at 11:43

8

I think it's a wise decision made by the Hibernate team. They could be less arrogante and make it clear why it was implemented this way, but that's just how Emmanuel, Chris and Gavin works. :)

Let's try to understand the problem. I think your concepts are "lying". First you say that many **Problem**s are associated to **People**. But, then you say that one **Person** have many **UglyProblem**s (and does not relate to other **Problem**s). Something is wrong with that design.

Imagine how it's going to be mapped to the database. You have a single table inheritance, so:

```
 ____________
|__PROBLEMS__|          |__PEOPLE__|
|id <PK>     |          |          |
|person <FK> | -------->|          |
|problemType |          |_________ |
 --------------
```

How is hibernate going to enforce the database to make **Problem** only relate to **People** if its **problemType** is equal UP? That's a very difficult problem to solve. So, if you want this kind of relation, every subclass must be in it's own table. That's what `@MappedSuperclass` does.

PS.: Sorry for the ugly drawing :D

Share

Improve this answer

Follow

edited Dec 4, 2018 at 9:28

SilverNak
3,381 ● 4 ● 31 ● 45

answered Sep 10, 2008 at 5:06

Marcio Aguiar
14.5k ● 6 ● 40 ● 42

5

Unfortunately, according to the Hibernate documentation "Properties from superclasses not mapped as @MappedSuperclass are ignored." I ran up against this too. My solution was to represent the desired inheritance through interfaces rather than the entity beans themselves.

In your case, you could define the following:

```
public interface Problem {
    public Person getPerson();
}

public interface UglyProblem extends Problem {
}
```

Then implement these interfaces using an abstract superclass and two entity subclasses:

```
@MappedSuperclass
public abstract class AbstractProblemImpl implements Problem {
    @ManyToOne
    private Person person;

    public Person getPerson() {
        return person;
    }
}

@Entity
public class ProblemImpl extends AbstractProblemImpl implements Problem {
}

@Entity
public class UglyProblemImpl extends AbstractProblemImpl implements UglyProblem
{
}
```

As an added benefit, if you code using the interfaces rather than the actual entity beans that implement those interfaces, it makes it easier to change the underlying mappings later on (less risk of breaking compatibility).

Share

Improve this answer

Follow

edited Sep 2, 2008 at 8:04

answered Sep 2, 2008 at 7:56

David Crow

16.2k ● 8 ● 43 ● 34

2

I figured out how to do the OneToMany mappedBy problem.

In the derived class UglyProblem from the original post. The callback method needs to be in the derived class not the parent class.

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@ForceDiscriminator
public class Problem {

}

@Entity
@DiscriminatorValue("UP")
public class UglyProblem extends Problem {
    @ManyToOne
    private Person person;
}

@Entity
public class Person {
    @OneToMany(mappedBy="person")
    private List< UglyProblem > problems;
}
```

Found the secret sauce for using Hibernate at least. http://docs.jboss.org/hibernate/stable/annotations/api/org/hibernate/annotations/ForceDiscriminator.html The @ForceDiscriminator makes the @OneToMany honor the discriminator

Requires Hibernate Annotations.

Share

Improve this answer

Follow

edited Dec 21, 2009 at 17:38

answered Dec 17, 2009 at 18:28

Bill Leeper

703 ● 1 ● 10 ● 21

According to the JavaDoc `InheritanceType.SINGLE_TABLE` is the default – alexander Jul 26, 2019 at 15:02

`@ForceDiscriminator` has been replaced with `@DiscriminatorOptions(force = true)` - I needed to use this when I had a `@OneToMany` – anztenney May 10 at 19:24

1

I think you need to annotate your *Problem* super-class with @MappedSuperclass instead of *@Entity*.

Share Improve this answer Follow

answered Sep 1, 2008 at 16:09

Peter Hilton
17.3k ● 6 ● 51 ● 76

-2

In my opinion @JoinColumn should at least provide an option to apply the @DiscriminatorColumn = @DiscriminatorValue to the SQL "where" clause, although I would prefer this behaviour to be a default one.

I am very surprised that in the year 2020 this is still an issue. Since this object design pattern is not so rare, I think it is a disgrace for JPA not yet covering this simple feature in the specs, thus still forcing us to search for ugly workarounds.

Why must this be so difficult? It is just an additional where clause and yes, I do have a db index prepared for @JoinColumn, @DiscriminatorColumn combo.

.i.. JPA

Introduce your own custom annotations and write code that generates native queries. It will be a good exercise.

Share Improve this answer Follow

answered Jun 16, 2020 at 12:03

Sašo Horvat
1