Best way to model Many-To-One Relationships in NHibernate When Dealing With a Legacy DB?

Asked 16 years, 4 months ago Modified 12 years, 4 months ago

Viewed 15k times

3

Warning - I am very new to NHibernate. I know this question seems simple - and I'm sure there's a simple answer, but I've been spinning my wheels for some time on this one. I am dealing with a legacy db which really can't be altered structurally. I have a details table which lists payment plans that have been accepted by a customer. Each payment plan has an ID which links back to a reference table to get the plan's terms, conditions, etc. In my object model, I have an AcceptedPlan class, and a Plan class. Originally, I used a many-to-one relationship from the detail table back to the ref table to model this relationship in NHibernate. I also created a one-to-many relationship going in the opposite direction from the Plan class over to the AcceptedPlan class. This was fine while I was simply reading data. I could go to my Plan object, which was a property of my AcceptedPlan class to read the plan's details. My problem arose when I had to start inserting new rows to the details table. From my reading, it seems the only way to create a new child object is to add it to the parent object and then save the session. But I don't want to have to create a new parent

Plan object every time I want to create a new detail record. This seems like unnecessary overhead. Does anyone know if I am going about this in the wrong way?

c# nhibernate

Share

Improve this question

Follow

asked Aug 14, 2008 at 11:56

Mark Struzinski

33.5k ● 35 ● 108 ● 137

6 Answers

Sorted by: Highest score (default)

3

I'd steer away from having child object containing their logical parent, it can get very messy and very recursive pretty quickly when you do that. I'd take a look at how you're intending to use the domain model before you do that sort of thing. You can easily still have the ID references in the tables and just leave them unmapped.

Here are two example mappings that might nudge you in the right direction, I've had to adlib table names etc but it could possibly help. I'd probably also suggest mapping the StatusId to an enumeration.

Pay attention to the way the bag effectivly maps the details table into a collection.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping default-cascade="save-update" xmlns
```

```
2.2">
    <class lazy="false" name="Namespace.Customer, Name
        <id name="Id" type="Int32" unsaved-value="0">
            <column name="CustomerAccountId" length="4
null="true" unique="true" index="CustomerPK"/>
            <generator class="native" />
        </id>

        <bag name="AcceptedOffers" inverse="false" laz
delete-orphan" table="details">
          <key column="CustomerAccountId" foreign-key=
          <many-to-many
            class="Namespace.AcceptedOffer, Namespace"
            column="AcceptedOfferFK"
            foreign-key="AcceptedOfferID"
            lazy="false"
           />
        </bag>

  </class>
</hibernate-mapping>


<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping default-cascade="save-update" xmlns
2.2">
    <class lazy="false" name="Namespace.AcceptedOffer,
table="AcceptedOffer">
        <id name="Id" type="Int32" unsaved-value="0">
            <column name="AcceptedOfferId" length="4"
null="true" unique="true" index="AcceptedOfferPK"/>
            <generator class="native" />
        </id>

        <many-to-one
          name="Plan"
          class="Namespace.Plan, Namespace"
          lazy="false"
          cascade="save-update"
        >
        <column name="PlanFK" length="4" sql-type="int
        </many-to-one>

        <property name="StatusId" type="Int32">
```

```
            <column name="StatusId" length="4" sql-typ
        </property>

    </class>
</hibernate-mapping>
```

Share Improve this answer

Follow

answered Aug 14, 2008 at 15:04

DavidWhitney

4,378 ● 4 ● 25 ● 30

1

Didn't see your database diagram whilst I was writing.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping default-cascade="save-update" xmlns
2.2">
    <class lazy="false" name="Namespace.Customer, Name
        <id name="Id" type="Int32" unsaved-value="0">
            <column name="customer_id" length="4" sql-
null="true" unique="true" index="CustomerPK"/>
            <generator class="native" />
        </id>

        <bag name="AcceptedOffers" inverse="false" laz
delete-orphan">
            <key column="accepted_offer_id"/>
            <one-to-many class="Namespace.AcceptedOffe
        </bag>

    </class>
</hibernate-mapping>


<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping default-cascade="save-update" xmlns
2.2">
    <class lazy="false" name="Namespace.AcceptedOffer,
table="Accepted_Offer">
        <id name="Id" type="Int32" unsaved-value="0">
            <column name="accepted_offer_id" length="4
```

```
null="true" unique="true" />
            <generator class="native" />
        </id>

        <many-to-one name="Plan" class="Namespace.Plan
cascade="save-update">
            <column name="plan_id" length="4" sql-type
        </many-to-one>

  </class>
</hibernate-mapping>
```

Should probably do the trick (I've only done example mappings for the collections, you'll have to add other properties).

Share Improve this answer

Follow

answered Aug 14, 2008 at 15:12

DavidWhitney

4,378 ● 4 ● 25 ● 30

0

The approach I'd take to model this is as follows:

Customer object contains an ICollection <PaymentPlan> PaymentPlans which represent the plans that customer has accepted.

The PaymentPlan to the Customer would be mapped using a bag which uses the details table to establish which customer id's mapped to which PaymentPlans. Using cascade all-delete-orphan, if the customer was deleted, both the entries from details and the PaymentPlans that customer owned would be deleted.

The PaymentPlan object contains a PlanTerms object which represented the terms of the payment plan.

The PlanTerms would be mapped to a PaymentPlan using a many-to-one mapping cascading save-update which would just insert a reference to the relevant PlanTerms object in to the PaymentPlan.

Using this model, you could create PlanTerms independantly and then when you add a new PaymentPlan to a customer, you'd create a new PaymentPlan object passing in the relevant PlanTerms object and then add it to the collection on the relevant Customer. Finally you'd save the Customer and let nhibernate cascade the save operation.

You'd end up with a Customer object, a PaymentPlan object and a PlanTerms object with the Customer (customer table) owning instances of PaymentPlans (the details table) which all adhear to specific PlanTerms (the plan table).

I've got some more concrete examples of the mapping syntax if required but it's probably best to work it through with your own model and I don't have enough information on the database tables to provide any specific examples.

Share Improve this answer

Follow

answered Aug 14, 2008 at 13:23

DavidWhitney

4,378 ● 4 ● 25 ● 30

0

I don't know if this is possibly because my NHibernate experience is limited, but could you create a BaseDetail class which has just the properties for the Details as they map directly to the Detail table.

Then create a second class that inherits from the BaseDetail class that has the additional Parent Plan object so you can create a BaseDetail class when you want to just create a Detail row and assign the PlanId to it, but if you need to populate a full Detail record with the Parent plan object you can use the inherited Detail class.

I don't know if that makes a whole lot of sense, but let me know and I'll clarify further.

Share Improve this answer

Follow

answered Aug 14, 2008 at 13:25

lomaxx

116k ● 58 ● 147 ● 180

0

I think the problem you have here is that your AcceptedOffer object contains a Plan object, and then your Plan object appears to contain an AcceptedOffers collection that contains AcceptedOffer objects. Same thing with Customers. The fact that the objects are a child of each other is what causes your problem, I think.

Likewise, what makes your AcceptedOffer complex is it has a two responsibilities: it indicates offers included in a plan, it indicates acceptance by a customer. That violates the Single Responsibility Principle.

You may have to differentiate between an Offer that is under a Plan, and an Offer that is accepted by customers. So here's what I'm going to do:

1. Create a separate Offer object which does not have a state, e.g., it does not have a customer and it does not have a status -- it only has an OfferId and the Plan it belongs to as its attributes.
2. Modify your Plan object to have an Offers collection (it does not have to have accepted offer in its context).
3. Finally, modify your AcceptedOffer object so that it contains an Offer, the Customer, and a Status. Customer remains the same.

I think this will sufficiently untangle your NHibernate mappings and object saving problems. :)

Share Improve this answer Follow

edited Aug 14, 2008 at 15:00

answered Aug 14, 2008 at 14:56

Jon Limjap

95.3k ● 15 ● 103 ● 153

0

A tip that may (or may not) be helpful in NHibernate: you can map your objects against Views as though the View was a table. Just specify the view name as a table name; as long as all NOT NULL fields are included in the view and the mapping it will work fine.

Share Improve this answer

Follow

edited Aug 11, 2012 at 15:51

burning_LEGION

13.4k ● 8 ● 41 ● 52

answered Aug 14, 2008 at 15:04

Jon Limjap

95.3k ● 15 ● 103 ● 153