Modeling Inheritance with Entity Modeler

An integral aspect of object oriented programming is the class hierarchy. Code is reused by taking very general classes and extending them with sub-classes... a.k.a. Inheritance. Since WebObjects maps database tables into java objects, it should be no surprise then that WO provides a way to map these sorts of class hierarchies into a database as well. Apple's EOModeler User Guide devotes a chapter to the modeling of inheritance. I assume here that you are already familiar with that information. The goal of this document is to explain how to model the three different types of inheritance in the WOLips Entity Modeler.

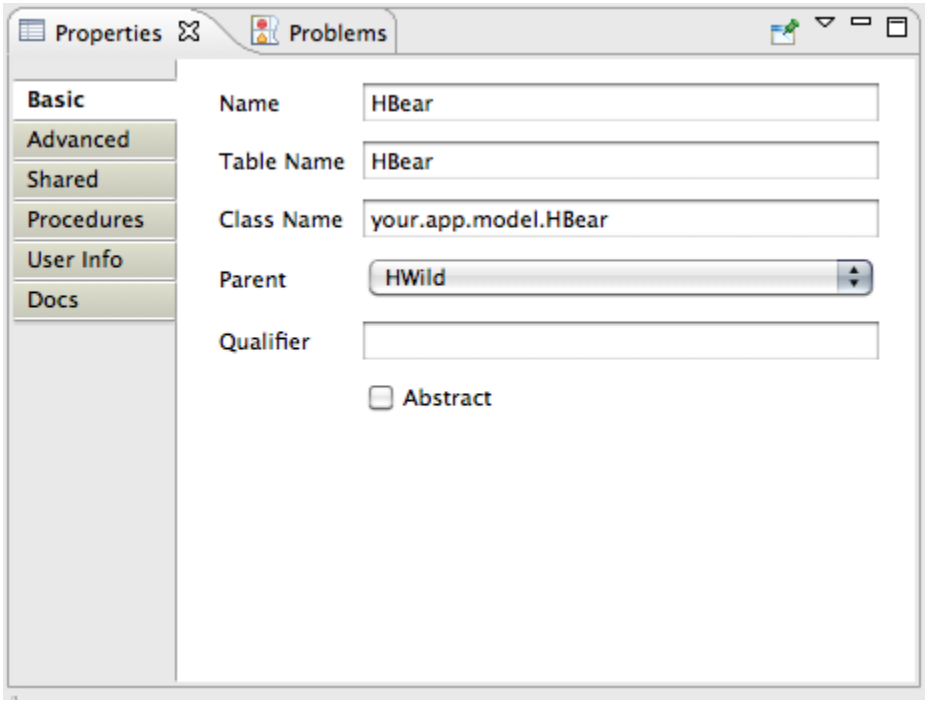
Entity Modeler

If you are modeling inheritance and you don't already have your subentities modeled, then you probably want to use entity modeler's subclass widget. To use it, just select the entity you want to subclass and click the subclass button in the toolbar.

Manual Inheritance Modeling

Horizontal Inheritance

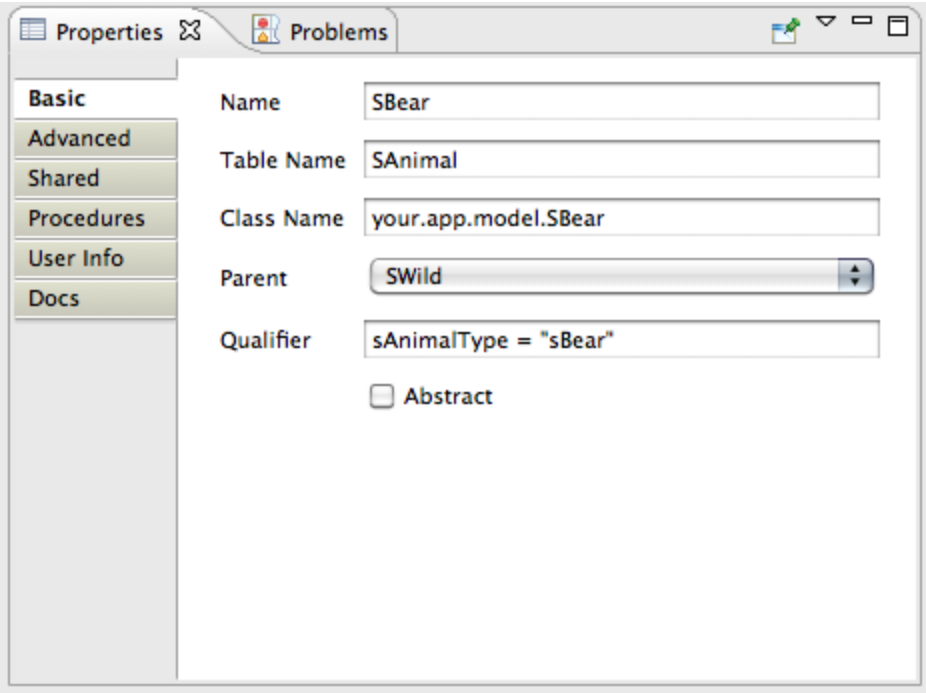
By far the easiest form of inheritance to model is Horizontal Inheritance. If you've already modeled a subentity, just select the entity then in the properties panel, select a parent entity and you're done. Yes, it really is that easy. When you save your model, Entity Modeler warns that you forgot to copy parent relationships and attributes to your new child entity. Then it kindly does the job for you.



Gotchas: Database foreign key constraints don't work for relationships to the superclass. With all forms of EOF inheritance, primary keys must be unique across all inherited tables. Entities generated with horizontal inheritance have a base class table, but it is empty. The table exists to generate unique primary keys for all the subclass tables. Obviously, since the table is empty, there's no foreign key to constrain. At the end of this article you'll find a zip file with a working inheritance example. In it, you will see the rather contrived relationship between HCart and HAnimal. Attempting to create an HCart will not work unless FK constraints were disabled at the time you executed your generated SQL due to this.

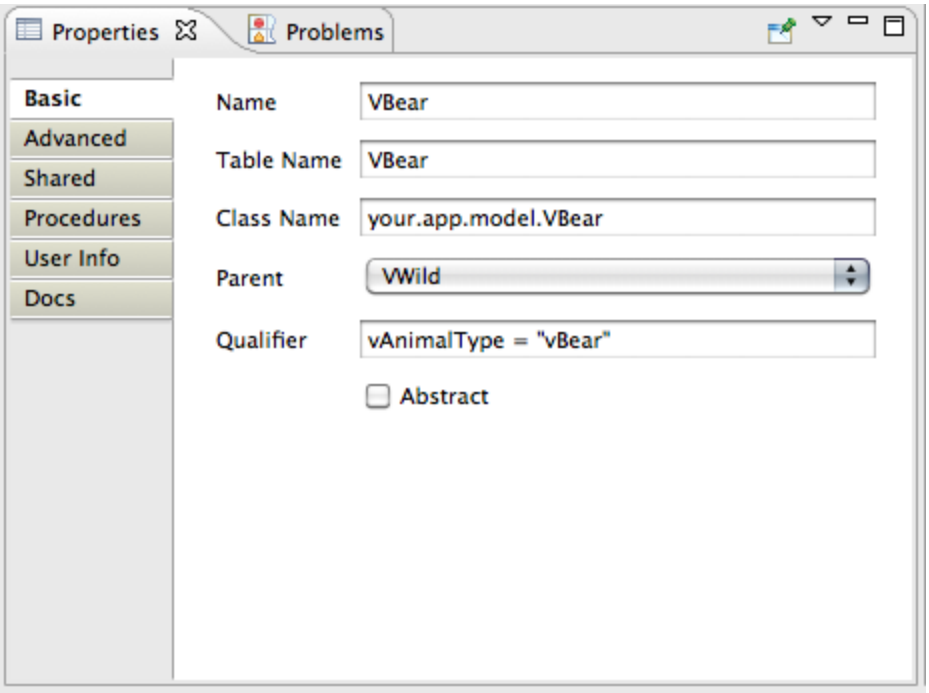
Single Table Inheritance

Single table inheritance is a bit more involved, but not much more. To create an subentity using single table inheritance, you must set three values for concrete subentities. Start by selecting your chosen subentity. Select the parent entity in the properties panel as you did before. This time, you also need to set the table name to match the table name of the root entity class. Finally, if this subclass is not abstract, you need to provide a qualifier that will identify rows in the superclass table that belong to this particular entity. Obviously, to do this, your root entity must have a column to store your unique identifier. You also need to copy the parent attributes and relationships, but I believe entity modeler will take care of that for you on save just like it does with horizontal inheritance.

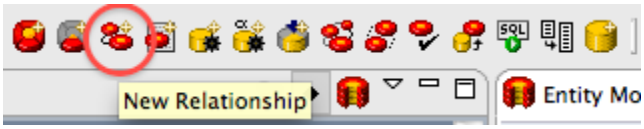


Vertical Inheritance

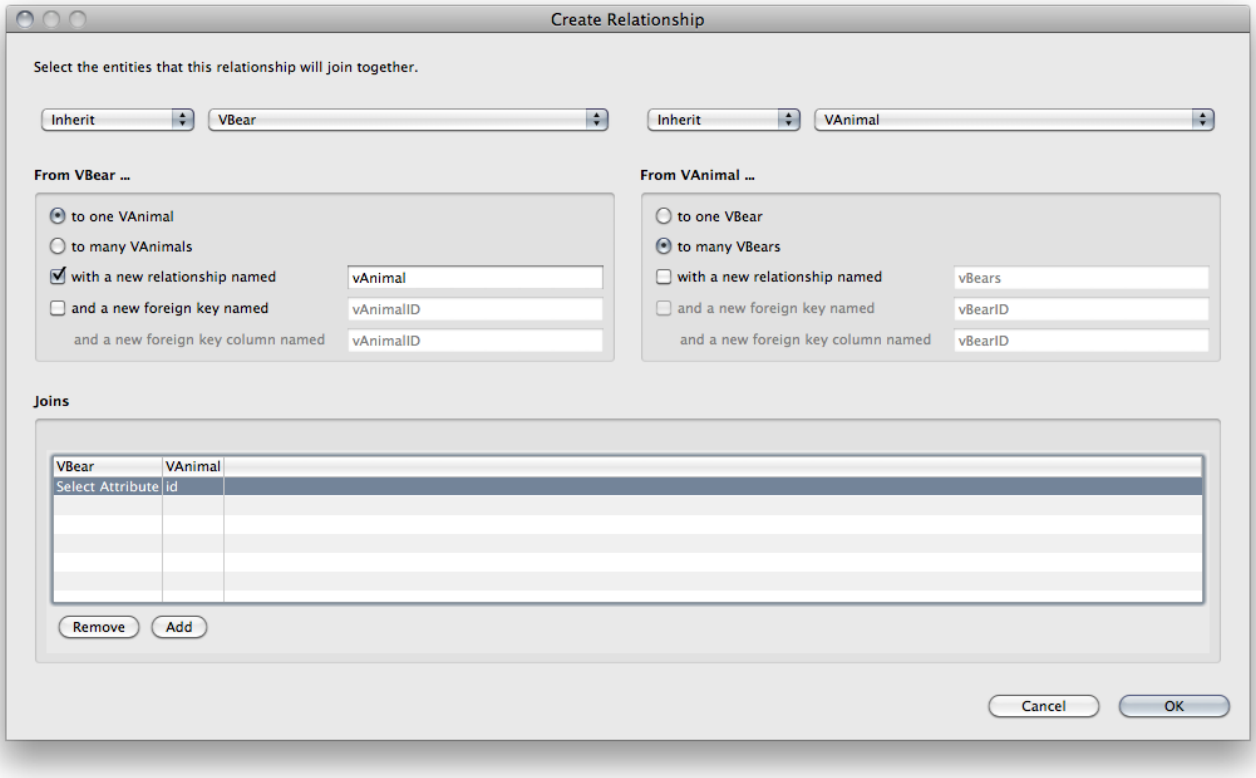
Vertical Inheritance is the most challenging. Like single table, each concrete entity will need a type and qualifier, but the similarity ends there.



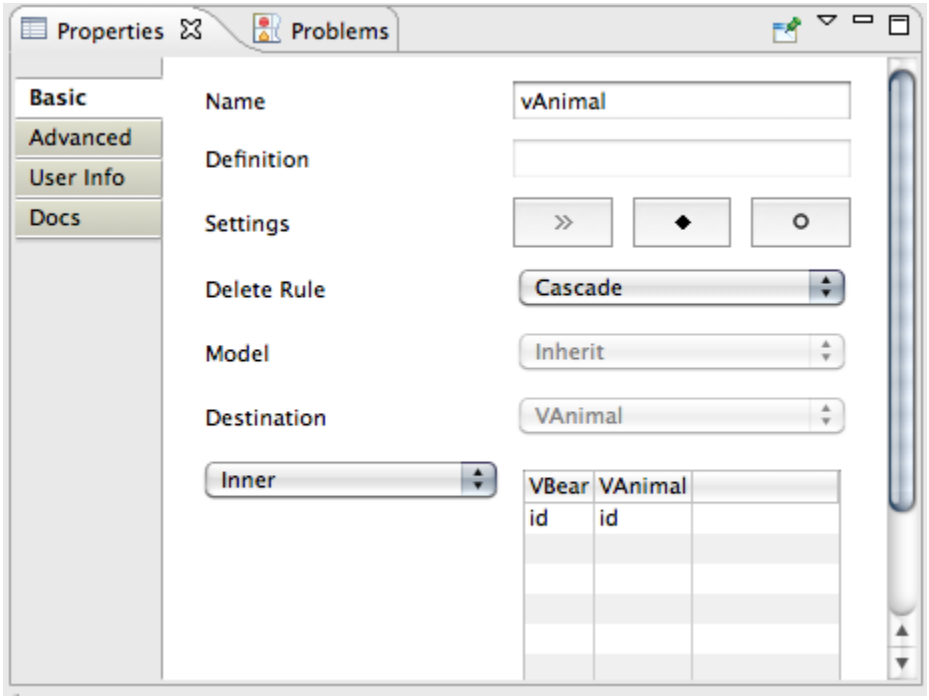
In vertical inheritance, relationships and attributes for vertical subclasses are not copied. They are flattened instead. To do this, you begin by creating a to one relationship to each parent entity table that has class properties. The to-one relationship needs to be joined on the primary key of the destination table. Start by clicking the new relationship button.



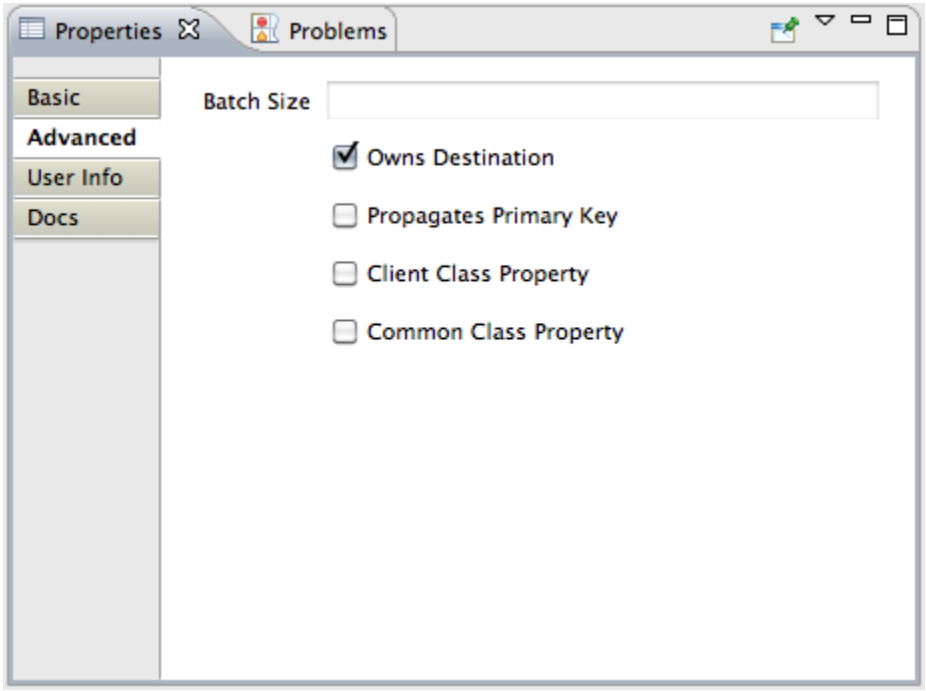
That will open the relationship wizard. Here you create a one way to-one relationship to the parent entity. Don't create a new foreign key though. Instead, uncheck that box and select the primary key of your new subclass in the join table below it.



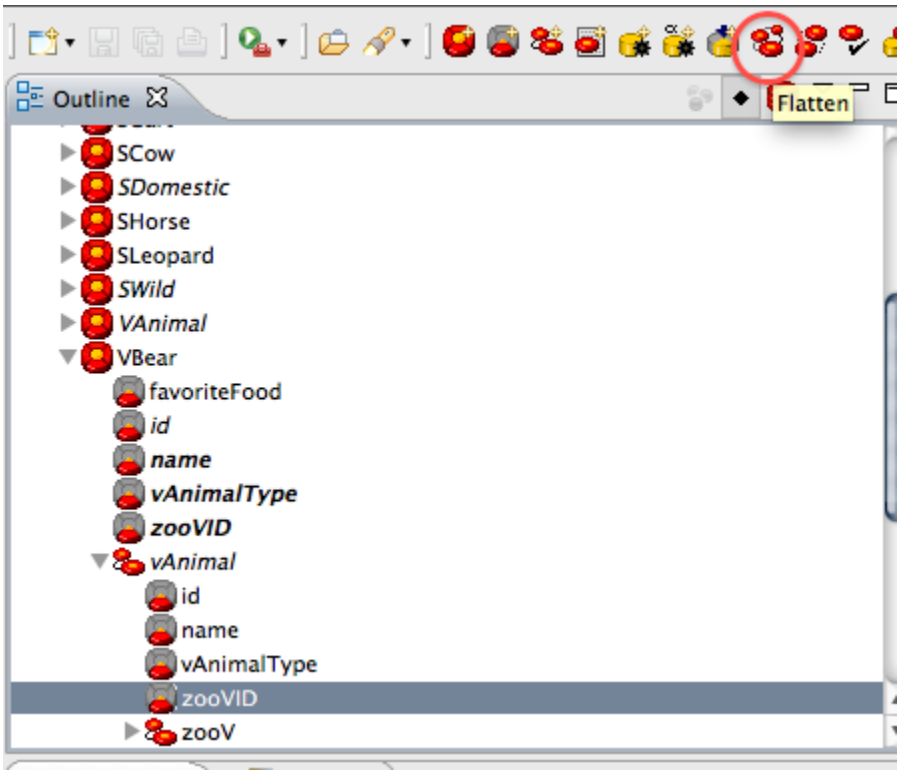
Once created, you'll need to adjust the properties of this relationship. It should be a non-class property. It should own the destination table and it should cascade delete.



You will find the Owns Destination checkbox in the Advanced properties panel of the relationship.



Now that you have your to-one relationship, you can use it to flatten the appropriate attributes and relationships into your subentity. To do that, turn down the arrow beside the to-one relationship, select the attribute or relationship you would like flattened and then click the flatten button on your toolbar. You will need to rename your flattened property so that it matches the property name of the parent's property.



If you are flattening a to-one relationship from the parent into the child entity, you MUST also flatten the foreign key for that relationship as well. You also MUST lock that foreign key. Your relationship will fail unless you do both these things. Furthermore, the flattening MUST be done with the "shortest hop" from your subentity. The reason for this is the way EOF expands the relationship definition vs attribute definition. In the VBear example, if the zooVID were modeled as vWild.zooVID and the zooV relationship definition was vWild.zooV instead of vAnimal.zooVID and vAnimal.zooV, then the expanded key path for the attribute will be vAnimal.zooVID and the expanded key path for the relationship join will be vWild.vAnimal.zooVID.

Gotchas: Due to the way the PKs are modeled, you must have a database that supports deferred constraints, or you must handle the order of the commit operations. I have tried VI in three different databases and it only worked in Postgresql so far. Maybe MySQL would work if the operations were ordered. I didn't try it.

Example

WO:Inherit.zip