Next Previous Index

# 4. Collection (many-to-many)

**Definition:**
A many-to-many reference is basically a collection. First-rank class A holds a reference to a set of first-rank class B instances (as in the one-to-many case), but B might have multiple A's.

**Scenario:**
We have two first-rank classes, Foo and Bar which are related to each other as follows:

```
Set Foo.getBars() // returns a collection of Bar instances
```

**Hibernate Mapping:**
In Hibernate, this could be mapped as follows:

```
<class name="Foo" table="foo">
     ...
     <set role="bars" table="foo_bar">
          <key column="foo_id"/>
          <many-to-many column="bar_id" class="Bar"/>
     </set>
</class>
```

**Table Schema:**

**Foo**
id

**Bar**
id

**Foo_Bar**
foo_id bar_id

This time we cannot have an extra column on Bar as that would dictate that each Bar has only one Foo. So instead we have an extra table, *foo_bar*, which holds the relationship between instances.

**Bidirectionality:**
This relationship can be declared both ways, with Bar having getFoos(), by suitable code changes to Bar and the following schema change:

```
<class name="Bar" table="bar">
     ...
     <set role="foos" table="foo_bar" readonly="true">
          <key column="bar_id"/>
          <many-to-many column="foo_id" class="Foo"/>
     </set>
</class>
```

Now your Bars will know who their Foos are.

NB: No extra columns are generated for the bidirectionality.
NB: Note that one end of the relationship must be declared "readonly".

If you want independent collections of Foos on Bars and Bars on Foos (i.e. membership one way doesn't imply the other), you need to declare Bar's table to be *bar_foo*. That way an independent table will be used to keep track of the Foo set on Bar.

Next Previous Index