Table of Contents

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

7 entity relationships

When Beans Relate

Entity beans need relationships. An Order needs a Customer. A LineItem needs an Order. An Order needs LineItems. A Movie needs a Director. A Director needs Movies. A Movie needs Actors. An Actor needs *talent*... Entity beans can have container-managed relationships (CMR) and the Container takes care of virtually everything. Make a new Movie and give it a Director? That Director automatically has one more Movie in his Movie collection. Make a new LineItem that's related to an Order? If you ask the Customer to show you his Orders, his Orders will show the new LineItem. Best of all, you can use EJB-QL to write portable (think: vendor/databse independent) queries.

this is a new chapter 373

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

***exam** objectives*

Entity Relationships: CMP and CMR

Official:

6.2 Identify correct and incorrect statements or examples about persistent relationships, remove protocols, and about the abstract schema type, of a CMP entity bean.

6.3 Identify correct and incorrect statements or examples about rules and semantics for relationship assignment updating, in a CMP bean.

6.4 Match the name with a description of purpose or functionality for each of the following: <ejb-name>, <abstract-schema-name>, <ejb-relationship-role>, <cmr-field>, <cmr-field-type>, and <relationship-role-source>

6.5 Identify correctly implemented deployment descriptor elements for a CMP bean (including container-managed relationships).

9.1 Identify correct and incorrect syntax for an EJB-QL query including the SELECT, FROM, and WHERE clause.

9.2 Identify correct and incorrect statements or examples about the purpose and use of EJB-QL.

9.3 Identify correct and incorrect conditional expressions, between expressions, like expressions, and comparison expressions.

What it really means:

You have to know that the container manages relationships between two entity beans through container-managed persistent (CMR) fields, and that CMR fields must be described in the deployment descriptor using the <relationships> section.

You have to understand the implications of multiplicity in a relationship, and that the Container cares whether the relationship is one-to-one, one-to-many, or many-to-many. You must know that the Container maintains the referential integrity of the database by using the multiplicity when doing assignments. For example, if there can be only one Customer per Order, if you assign an Order to a Customer, that Order cannot exist in any other Customer's Order collection. And if you reassign that Order to a different Customer, the Order is moved from one Customer's collection to the other. But if you assign a Customer with three existing Orders, to another Order, the reference to the Customer is copied, not moved, so that the reference to a Customer that the other three existing Orders have, is not affected.

You have to know that EJB-QL queries are defined in the deployment descriptor, and are for Finder and Select methods only. You have to know the basic syntax of EJB-QL, and that FROM and SELECT are mandatory, but WHERE is optional.

You have to know that if you use path navigation to, say, get the title of a Movie, you can say m.title (assuming "m" is declared as the abstract schema type of Movie, and "title" is a CMP field of the MovieBean). But that you can't say d.movies.title where "d" is a Director, and "movies" is a CMR field that is a Collection of movies. For that, you must use the IN operator within a FROM clause. You'll see... it's all in here. And much simpler than it sounds.

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Beanifying your movie database

Imagine you have a movie application (it isn't making the folks at imdb.com nervous, but it works for you).

You can use it to look up a movie.

Once you have a movie, you can use it to launch the movie trailer.

You can do all kinds of searches to, say, *find all sci-fi movies*, or *find all action movies by a specific director.*

We can make beans for Movie, Trailer, and Director. But how do they map to the database? And how do they relate to one another?

a foreign key into the Director table

Movie Table

MovieID	Title	Genre	DirectorID	Year
12	Crouching Pixels, Hidden Mouse	Action	42	2000
5	The Fifth Array Element	Action	27	2003
22	The Return of the Bean Queen	Fantasy	27	2001
11	Lord of the Loops	Sci-fi	42	2002

Director Table

DirectorID	OscarWinner	Degrees	Name
27	TRUE	3	Jim Yingst
42	FALSE	6	Jessica Sant
56	FALSE	24	Skyler Safford
17	FALSE	5	John Dawson

Director doesn't have a reference to his movies.

Trailer Table

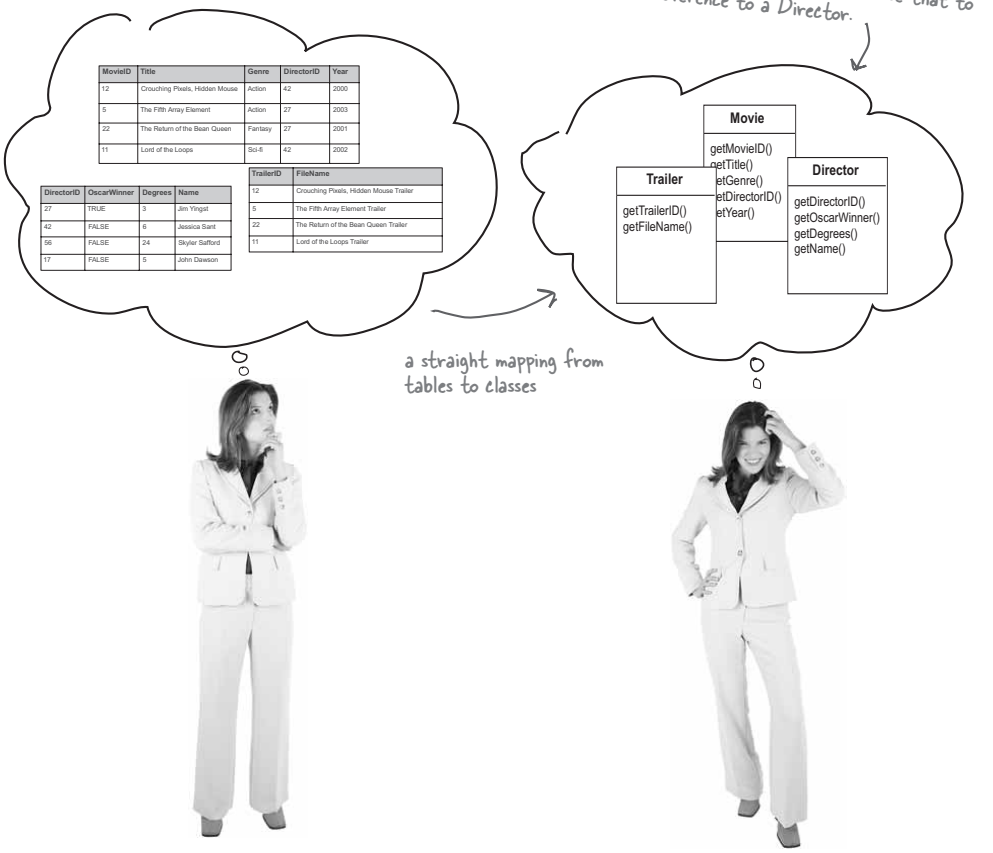
TrailerID	FileName
12	Crouching Pixels, Hidden Mouse Trailer
5	The Fifth Array Element Trailer
22	The Return of the Bean Queen Trailer
11	Lord of the Loops Trailer

The trailer's primary key always matches the MovieID

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

But we don't want to think in TABLES
We want to think in CLASSES

We know we can map tables to bean classes, no problem. All the columns become persistent fields in the bean class, represented by abstract getters and setters, and we're good to go. Except... if this were a class and not a table, we wouldn't have designed it that way. We'd probably make the movie class a lot more friendly and useful, for example, so that a client could work with just the movie bean, rather than having to get references to all three beans.



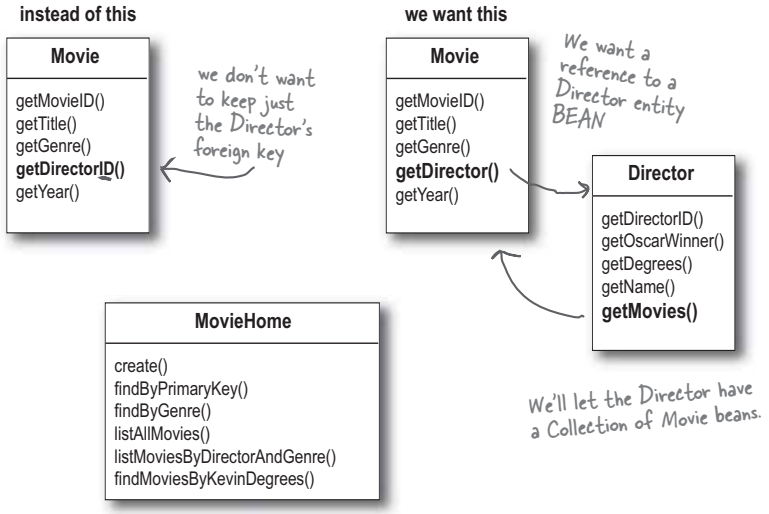
Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

We need relationships between the Movie bean and the Director bean

We want the Movie bean to have a reference to its matching Director bean and we want to do all sorts of searches against the Movie bean and have the queries use the Director bean's data as well.

In other words, we want to make it easy on the client (and on the developer) to think in a more natural way, rather than in the database-efficient way that was used to design the schema of the database. Who wants that? Remember, this is the OO world and relational databases, while crucial to your business, are so 1999. We want to *use* databases, we just don't want to *think* like databases. (If you're one of the lucky ones who gets to use an OO database, and assuming that database still somehow manages to perform well enough for your needs, then you can just smile smugly during this first section.)



We're not going to use the Trailer table and bean after this (although you'll see it in code), to keep the example cleaner. Adding the Trailer bean wouldn't add any new complexity to the application, though. If you can set-up one bean-to-bean relationship, you can set up others.

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Why should the Director be a bean? Why can't it just be data?

Why not have the Movie bean simply go to the database, using the Director's foreign key (stored as a persistent field in the Movie bean), and get the Director's data? But if you make the director a bean as well, you get to think ONLY in objects (except during deployment, when you do have to map from your beans to your tables). And you get all the benefits of container-managed persistence and synchronization. Imagine if someone calls a setter method to change the director's oscar winning status. If you're managing this, you'd have to get a JDBC connection and synchronize the database yourself. And of course, you'd have to know when is the right time, etc. But if its director is also a bean, you call a setWinner() method, and you're done.

Oh, and there's another cool thing. You might decide, for example, to banish a specific director from your database, because you think his movies suck in an unrecoverable way. No problem deleting that director, obviously. But then what happens to the movies? How can you have movies in the database that don't have a director? What is the database is already set up in such a way that you aren't allowed to have a null value for the director column? In other words, what if a movie cannot exist in the database without a director?

You can handle this brainlessly, by setting up the Movie-to-Director bean relationship in such a way that when you delete a director from the database, all of his movies are automatically deleted! You do this with one simple tag in the deployment descriptor and POOF! Somebody calls remove() on a director bean, and remove() will automatically be called on all of the movie entity beans that have a reference to this director (as the value of their director persistent field).

If both the Director and the Movie are entity beans, you never have to worry about synchronizing any of the Movie's related data; if somebody uses the Movie bean to change something about the Director, everything's taken care of by the Container.

You can even set things up for a cascade-delete... so if you remove a Director, all the Director's movies will be removed automatically!

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Relationships and multiplicity

Ⓐ One-to-One

Each Movie —has→ One Trailer

One Movie ←has— Each Trailer

Ⓑ One-to-Many

One Director —has→ Many Movies

One Director ←has— Each Movie

③ Many-to-Many

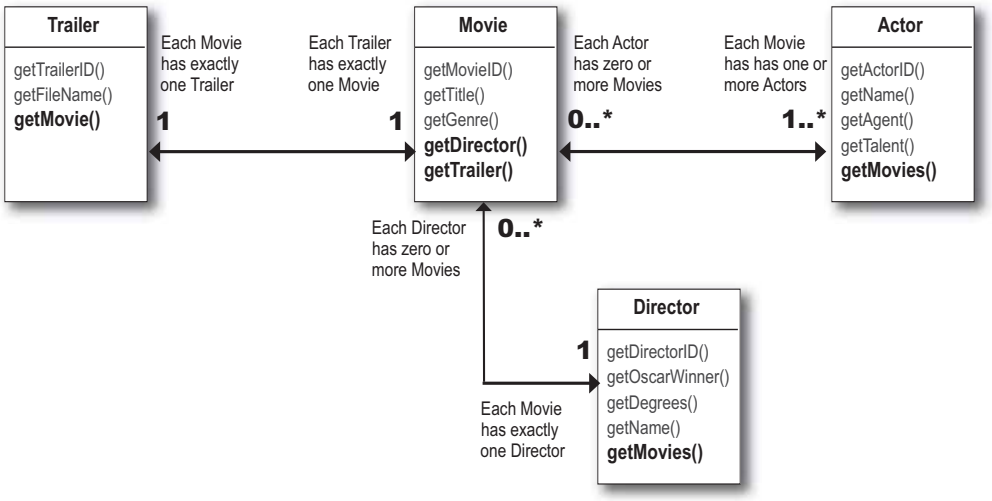
An Actor —has made→ Many Movies

Many Actors ←has— A Movie

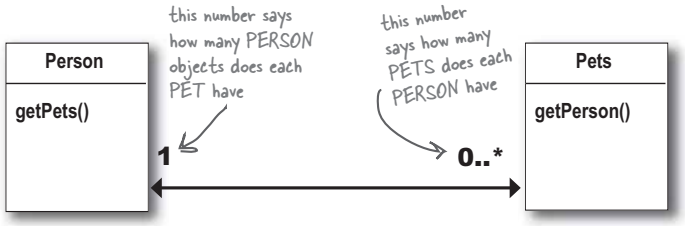
Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Multiplicity in Bean Classes

Here's how it works with our bean code. This multiplicity notation is the only UML-like thing you have to know on the exam.



To read this, you have to follow the arrow to its destination. In the Person to Pets relationship, Person has a multiplicity of ONE, but Pets has a multiplicity of MANY (which could be zero). To find out how many Pets a Person can have, you have to follow the arrow out of Person and into Pets. The number closest to the class is the multiplicity of that class to others. In other words, how many of that type does the other type have.

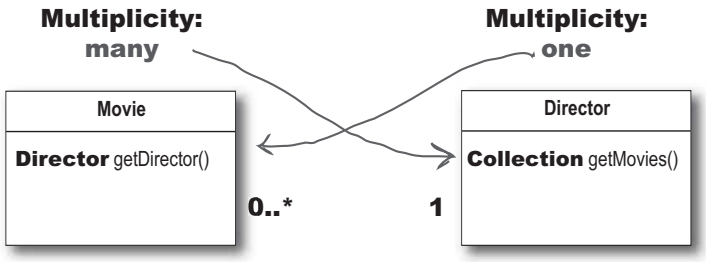


Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Multiplicity affects return type!

A Movie has one Director. When you call getDirector(), you get back one Director. So the return type of getDirector() is a Director().

But a Director has many movies, so when you call getMovies(), you get back a Collection of Movies.



Movie has a multiplicity of "many" in its relationship with Director. That does NOT mean that Movie has "many" Directors... it means that Director has "many" Movies!

So Director has a getMovies() that returns a Collection of Movies.

And Director has a multiplicity of "one", which means that Movie will return just one Director.

In the Movie-to-Director relationship, the multiplicity of Movie is *many* and the multiplicity of Director is *one*.

A multiplicity of *one* means the object you're related to holds just *one* of you.

Director has a multiplicity of *one*, so the Movie object related to Director returns just *a single Director* bean (OK, technically the component interface of Director, but you know what we mean by now).

A multiplicity of *many* means the object you're related to holds a *Collection* of you.

Movie has a multiplicity of *many*, so the Director object related to Movie returns a *Collection of Movie beans*.

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Defining virtual fields for persistent data fields and relationship fields

In the previous chapter we looked at defining container-managed fields—you put in a pair of abstract getters and setters. We said ***a container-managed field exists simply because you have a getter and setter for it.*** It works the same way with container-managed relationship (CMR) fields. You define a pair of abstract getters and setters, but rather than setting and returning a value that maps to a column in a table, you set and return a reference to another entity bean, or a Collection that will hold references to the entity bean. The restrictions for CMR fields are that they can refer only to the ***local*** component interface of the entity bean, and that if the method returns a Collection, the declared type can be only Collection or Set (not Map, List, etc.)

The terminology is a little confusing because both CMP and CMR fields are container-managed persistent fields. They might have called it CMPCD (container-managed persistent *column data*) fields and CMPR (container-managed persistent *relationship*) fields. Of course, persistence isn't restricted to just relational database, so we're using the term "column" a little loosely. But in EJB 1.1, *all* container-managed persistent fields were CMP fields, because there was no concept of persistent relationships in EJB 1.1 CMP.

The only difference between a CMP field and a CMR field is the TYPE.

A CMR field is always another entity bean's local interface type, or a Collection of them.

If it's a Collection, it must be either java.util.Collection or java.util.Set.

You need a pair of abstract getters and setters for each CMP field (column values) and each CMR field (relationship with another entity).

① CMP field

```
public abstract String getTitle();
public abstract void setTitle(String g);
```

make sure the argument and return type match

② CMR (relationship) field

```
public abstract Director getDirector();
public abstract void setDirector(Director d);
```

Director must be the local component interface for the Director bean.

The argument and return type of a CMR "virtual field" is always another Entity bean's local type, or a Collection.

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

the abstract persistence schema

Defining your "abstract persistence schema" (virtual fields aren't enough)

To define persistent fields and relationships, you need to create an ***abstract persistence schema***. Defining the virtual fields isn't enough. Your abstract persistence schema is a combination of your virtual fields in your bean class, plus some things you write in the deployment descriptor.

The way you describe CMP fields in the DD is simple and straightforward. With CMR, you'll have to do a little more to set things up, including describing the multiplicity for each of the two participants in a relationship.

But we'll get to CMR fields in just a minute. For now, we'll start with just CMP fields.

An abstract persistent schema is a combination of some stuff you put in the deployment descriptor, plus your abstract getters and setters.

Together, they tell the Container how to manage your bean's persistence, including both persistent FIELDS (called CMP fields) and persistent RELATIONSHIPS (called CMR fields).

The Container needs these TWO things to know you have a CMP field "title"

① In the DD

```
<entity>
   <ejb-name>MovieBean</ejb-name>
   <local-home>headfirst.MovieHome</local-home>
   <local>headfirst.Movie</local>
   <ejb-class>headfirst.MovieBean</ejb-class>
   <abstract-schema-name>MovieSchema</abstract-schema-name>
   <cmp-field>
        <field-name>title</field-name>
    </cmp-field>
      ...
<entity>
```

② In the bean class

```
public abstract String getTitle();
public abstract void setTitle(String g);
```


Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Persistent CMP fields in the DD

```
<entity>
   <display-name>MovieBean</display-name>
   <ejb-name>MovieBean</ejb-name>
   <local-home>headfirst.MovieHome</local-home>
   <local>headfirst.Movie</local>
   <ejb-class>headfirst.MovieBean</ejb-class>
   <persistence-type>Container</persistence-type>
   <prim-key-class>java.lang.String</prim-key-class>
   <reentrant>False</reentrant>
   <cmp-version>2.x</cmp-version>
   <abstract-schema-name>MovieSchema</abstract-schema-name>
   <cmp-field>
     <field-name>genre</field-name>
   </cmp-field>
   <cmp-field>
     <field-name>movieID</field-name>
   </cmp-field>
   <cmp-field>
     <field-name>year</field-name>
   </cmp-field>
   <cmp-field>
     <field-name>title</field-name>
   </cmp-field>
   <primkey-field>movieID</primkey-field>
  ...
<entity>
```

tell the Container that IT is managing your persistence (as opposed to "Bean")

fully-qualified class name for the field that represents the primary key

Name your schema so that you know which entity type it represents. This name must be unique in the DD!!

Now that you've told the Container WHICH fields to manage, tell it which of those fields is the Special One... the primary key field.

The naming convention is REQUIRED!

You must have a pair of abstract getters and setters that have the first character after the "get" and "set" capitalized, but in the CMP field definition in the DD, you start with a lowercase letter, leaving off the "get" and "set". In other words, you define the CMP field in the DD as if it were a real instance variable, following the old pre-EJB JavaBeans naming convention.

there are no Dumb Questions

Q: **How does the Container know the type of the persistent field? I see a definition for the primary key type (<prim-key-class>, but I don't see anything for the other CMP fields.**

A: The Container figures it out based on the return type and argument of the abstract getters and setters in the bean class. And they had better match! If you have a getLimitNum() that returns an int, you better have a setLimitNum(int i) that takes an int.

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Using relationships in your code

In your bean code, you use your virtual relationship fields just as you would any other getter and setter. The only difference between using a *real* field (i.e. an instance variable declared in your class) and a *virtual* field is that you can access your virtual fields only by calling the abstract getters and setters (which the Container implements at deploy-time).

The only new restriction is that a relationship field can be ***only*** a local component interface type! You can't have a CMR relationship field that uses a bean's Remote interface!

You can use your virtual relationship fields just as you would any other virtual field... by calling the abstract getters that YOU defined, but which the Container implements.

The type of a relationship field MUST be the local component interface of the entity!

A CMR field will ALWAYS be a local component interface type.

Virtual field in the bean class

```
public abstract Director getDirector();
public abstract void setDirector(Director d);
```

The argument and return type of a CMR "virtual field" is always another Entity bean type, and it MUST be a local interface!! You can't have a virtual field to the entity's Remote interface.

Exposed business method in the bean class

```
public Director getMovieDirector() {
    return getDirector();
}
```

Now we can return the Director to a client, by calling the abstract getter.

```
public String getMovieDirectorName() {
    return getDirector().getDirectorName();
}
```

We can also do the work on behalf of the client, if we don't want to expose the real Director reference to the client. In fact, that's how we're going to do it, because the Director is a local interface, but the Movie is going to have Remote clients.

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Defining relationships in your abstract persistence schema (in the DD)

CMP field definitions are so easy. You declare a <cmp-field> and give it a name that matches one of your getter/setter pairs (following the naming rule of dropping the "get" and "set" and beginning the CMP field name with a lower-case letter).

But CMR is more involved. You can't be in a relationship with only yourself, so **a relationship always includes two beans!**

1 Define an ejb-relation between two beans

Every relationship has exactly two beans.

A Define an ejb-relationship-role for one bean

1. **multiplicity for this bean** — Multiplicity of this bean (i.e. how many of this type will the OTHER participant have)
2. **source for this bean** — Which bean are we really talking about?
3. **cmr-field for this bean** — Must match the "virtual field" in the bean class
4. **cascade-delete for this bean** — Only if THIS bean wants to be deleted when its partner is deleted.

A Define an ejb-relationship-role for the second bean

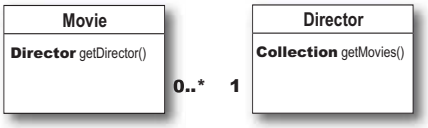
1. **multiplicity for this bean**
2. **source for this bean**
3. **cmr-field for this bean**
4. **cascade-delete for this bean**

Same stuff for the second participant in the relationship

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

relationships in the deployment descriptor

Relationship definition for the Director-to-Movie relationship



① Define an ejb-relation between two beans

```
<ejb-relation>
```

Each <ejb-relation> describes one pair of beans... one relationship. They REALLY should have named this <ejb-relationship>

Ⓐ Define an ejb-relationship-role for one bean

```
<ejb-relationship-role>
    <ejb-relationship-role-name>DirectorBean</ejb-relationship-role-name>
```

This is a made-up name! It's simply what you choose to call the ROLE. We could have called it "DirectorRole" or anything that tells us which participant we're talking about.

① multiplicity for this bean

```
<multiplicity>One</multiplicity>
```

Remember, this does NOT say how many of the other participant the Director bean will have... it says how many of THIS bean the OTHER partner will have. Movie will have only ONE Director.

② source for this bean

```
<relationship-role-source>
    <ejb-name>DirectorBean</ejb-name>
</relationship-role-source>
```

We can use any role-name we want, but sooner or later the Container must know EXACTLY which bean we're talking about. Put the bean's <ejb-name> which is just a label telling the Container where to find this bean's real definition in the <enterprise-beans> section of the DD. This MUST match an <ejb-name> in the DD.

③ cmr-field for this bean

```
<cmr-field>
    <cmr-field-name>movies</cmr-field-name>
```

This must match a "virtual field" in the bean class for getMovies() / setMovies()

```
    <cmr-field-type>java.util.Collection</cmr-field-type>
</cmr-field>
```

Use <cmr-field-type> ONLY if the other partner has a MANY relationship to this bean, which means you will hold more than one, so you need a Collection. You can also say java.util.Set (your only two choices)

④ cascade-delete for this bean

```
(not specified for this bean)
```

```
</ejb-relationship-role>
```


Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Ⓐ Define an ejb-relationship-role for the second bean

```
<ejb-relationship-role>
     <ejb-relationship-role-name>MovieBean</ejb-relationship-role-name>
```

Remember, this is just a name for your own (or the App Assembler or a tool's) use. It means nothing in your Java code.

① multiplicity for this bean

```
<multiplicity>Many</multiplicity>
```

This means Director will have MANY of this type (Movie). So we expect that Director will have a CMR field with a Collection type rather than a single Movie

② source for this bean

```
<relationship-role-source>
    <ejb-name>MovieBean</ejb-name>
</relationship-role-source>
```

This MUST match the <ejb-name> for where you defined the Movie entity bean in the <enterprise-beans> section of the DD.

③ cmr-field for this bean

```
<cmr-field>
    <cmr-field-name>director</cmr-field-name>
</cmr-field>
```

Look... no type! We don't have a <cmr-field-type> because it isn't a Collection. In fact, the only reason there's a <cmr-field-type> tag is for when the field type is a Collection, and you have to distinguish between Set or Collection. <cmr-field-type>

④ cascade-delete for this bean

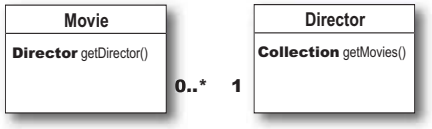
```
    <cascade-delete />
</ejb-relationship-role>
</ejb-relation>
```

Put this in if you want THIS bean to be deleted whenever its partner is deleted.

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Sharpen your pencil

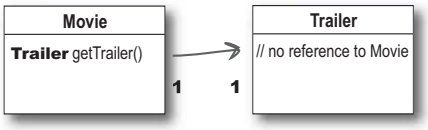
Here's the relationship DD for Director-to-Movie, but we've left a few things out. See if you can fill them in correctly *without* looking on the previous pages!



Director-to-Movie relationship

```
<relationships>
   <ejb-relation>
     <ejb-relationship-role>
       <ejb-relationship-role-name>_____________ </ejb-relationship-role-name>
       <multiplicity>________</multiplicity>
       <relationship-role-source>
         <ejb-name>DirectorBean</ejb-name>
       </relationship-role-source>
       <cmr-field>
         <cmr-field-name>__________ </cmr-field-name>
          ____________________________________
       </cmr-field>
     </ejb-relationship-role>
     <ejb-relationship-role>
       <ejb-relationship-role-name>MovieBean</ejb-relationship-role-name>
       <multiplicity>_________</multiplicity>
       <cascade-delete />
       <relationship-role-source>
         <ejb-name> ___________</ejb-name>
       </relationship-role-source>
       ____________________________________
       ____________________________________
       ____________________________________
     </ejb-relationship-role>
   </ejb-relation>
</relationships>
```


Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01



Relationships can be one-way (unidirectional)

You can have a relationship between two beans, but have a CMR field in only one of the two beans. For example, if you set up a relationship between a Movie and its Trailer (a one-to-one relationship), and you don't want clients to use a Trailer to get to a Movie, just leave the CMR field for Movie out of the Trailer bean. Simple as that.

In that case, the TrailerBean won't know anything about the MovieBean, even though they're both partners in a relationship.

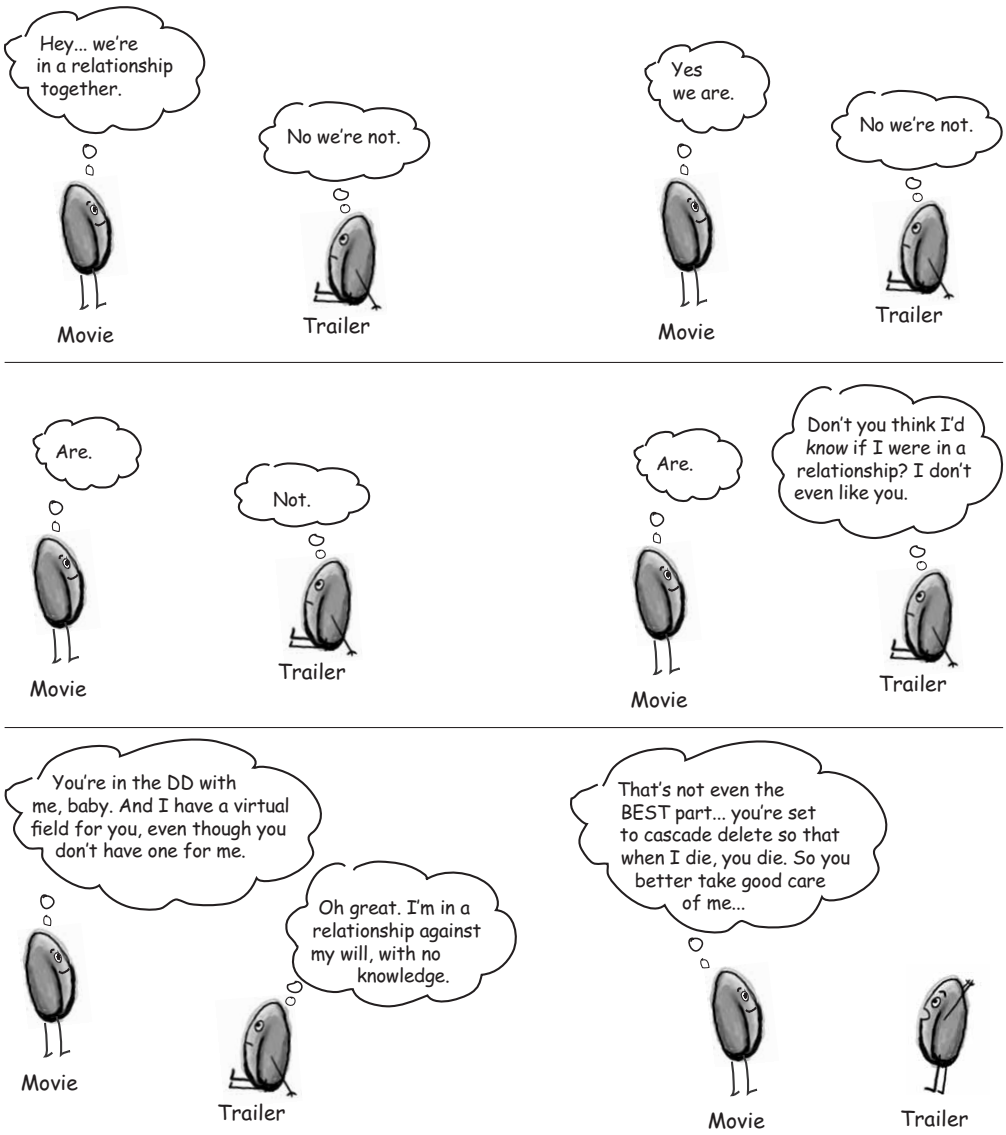
```
<ejb-relationship-role>
    <ejb-relationship-role-name>TrailerBean</ejb-relationship-role-name>
    <multiplicity>One</multiplicity>
    <cascade-delete />
    <relationship-role-source>
      <ejb-name>TrailerBean</ejb-name>
    </relationship-role-source>
    <cmr-field>
      <cmr-field-name></cmr-field-name>
    </cmr-field>
 </ejb-relationship-role>
```

Leave the cmr-field out of a relationship role if you don't want this bean to have a reference to its partner bean.

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

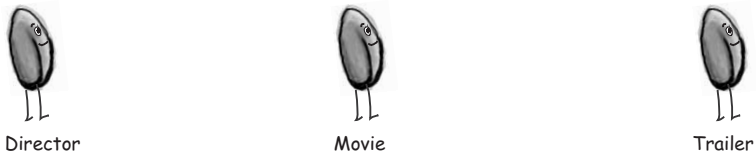
directionality in relationships

The dark side of a unidirectional (one-way) relationship



Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Cascade delete can propagate



Director-to-Movie relationship

Director (can't cascade-delete)

Movie - <cascade-delete/>

You can't set cascade-delete for the Director bean in the Director-to-Movie relationship, because Movie has a multiplicity of Many. Think about it... if someone deletes a Movie, should that Director be deleted? Of course not! The Director could still be the Director of other movies that still exist.

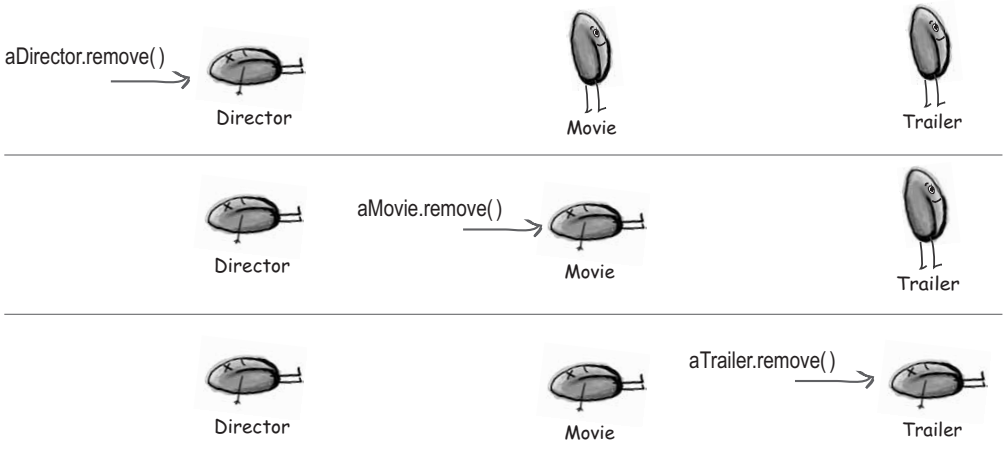
But... the Director has a multiplicity of One with the Movie, so the Movie bean can say, "I can have only one Director, so if he goes, I go."

Movie-to-Trailer relationship

Movie - (could cascade delete, but doesn't want to)

Trailer- <cascade-delete/>

Because this is a one-to-one relationship, both partners could set a cascade-delete. But we want it to go in only one direction, where deleting the Movie deletes the Trailer, but not the reverse. Just because the Trailer is deleted does not mean we want the Movie deleted, but that's determined by our business logic, and not EJB rules.



relationships in the deployment descriptor

The abstract schema is in the <entity> element and the <ejb-relation> is in the <relationships> element

```
<ejb-jar>
  <display-name>MovieJar</display-name>
  <enterprise-beans>
    <entity>
      <display-name>MovieBean</display-name>
      <ejb-name>MovieBean</ejb-name>
      <local-home>headfirst.MovieHome</local-home>
      <local>headfirst.Movie</local>
      <ejb-class>headfirst.MovieBean</ejb-class>
      <persistence-type>Container</persistence-type>
      <prim-key-class>java.lang.String</prim-key-class>
      <reentrant>False</reentrant>
      <cmp-version>2.x</cmp-version>
      <abstract-schema-name>MovieSchema</abstract-schema-name>
      <cmp-field>
        <field-name>genre</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>movieID</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>year</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>title</field-name>
      </cmp-field>
      <primkey-field>movieID</primkey-field>
      <security-identity>
           <use-caller-identity></use-caller-identity>
      </security-identity>
    </entity>

    <entity>
      <display-name>DirectorBean</display-name>
      <ejb-name>DirectorBean</ejb-name>
      <local-home>headfirst.DirectorHome</local-home>
      <local>headfirst.Director</local>
      <ejb-class>headfirst.DirectorBean</ejb-class>
      <persistence-type>Container</persistence-type>
      <prim-key-class>java.lang.String</prim-key-class>
      <reentrant>False</reentrant>
      <cmp-version>2.x</cmp-version>
      <abstract-schema-name>DirectorSchema</abstract-schema-name>
      <cmp-field>
        <field-name>name</field-name>
      </cmp-field>
```

The abstract schema for persistent fields is defined HERE in the <entity> element under <enterprise-beans>, where all beans are described.

WHOA! Notice what's missing!! There is NO <cmp-field> for Director. You don't list <cmr-field> elements inside the <entity> section. They're defined ONLY in the <relationships> section (next page)

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

```
    <cmp-field>
      <field-name>winner</field-name>
    </cmp-field>
    <cmp-field>
      <field-name>directorID</field-name>
    </cmp-field>
    <cmp-field>
      <field-name>degrees</field-name>
    </cmp-field>
    <primkey-field>directorID</primkey-field>
    <security-identity>
      <use-caller-identity></use-caller-identity>
    </security-identity>
  </entity>
</enterprise-beans>

<relationships>
  <ejb-relation>

    <ejb-relationship-role>
      <ejb-relationship-role-name>DirectorBean</ejb-relationship-role-name>
      <multiplicity>One</multiplicity>
      <relationship-role-source>
        <ejb-name>DirectorBean</ejb-name>
      </relationship-role-source>
      <cmr-field>
        <cmr-field-name>movies</cmr-field-name>
        <cmr-field-type>java.util.Collection</cmr-field-type>
      </cmr-field>
    </ejb-relationship-role>


    <ejb-relationship-role>
      <ejb-relationship-role-name>MovieBean</ejb-relationship-role-name>
      <multiplicity>Many</multiplicity>
      <cascade-delete />
      <relationship-role-source>
        <ejb-name>MovieBean</ejb-name>
      </relationship-role-source>
      <cmr-field>
        <cmr-field-name>director</cmr-field-name>
      </cmr-field>
    </ejb-relationship-role>

  </ejb-relation>
</relationships>
```

AFTER the <enterprise-beans> section (and all the beans have been described), we have the <relationships> section that uses <ejb-name> elements to refer BACK to the <enterprise-beans> section

This name MUST match something in this DD, described in an <entity> element in the <enterprise-beans> section

<cmr-field-type> is only when your partner (not necessarily YOU) has a multiplicity of Many. This can be ONLY Collection or Set (NOT List, Map, etc.)

MovieBean says, "You can delete me when you delete my Director"

No <cmr-field-type> because the Director has a multiplicity of One in this relationship, so we return only one Director, not a Collection of Directors.

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

the MovieBean code

MovieBean code with a CMR field

```
package headfirst;

import javax.ejb.*;
import javax.naming.*;
import java.util.*;

public abstract class MovieBean implements EntityBean {

   private EntityContext context;

   public String ejbCreate(String movieID, String title, int year,String genre,
                                   String directorID) throws CreateException {

     setMovieID(movieID);
     setTitle(title);
     setYear(year);
     setGenre(genre);
     return null;
   }

   public void ejbPostCreate(String movieID, String title, int year, String genre,
                                      String directorID) throws CreateException {
     try {
      InitialContext ctx = new InitialContext();
      DirectorHome dirHome = (DirectorHome) ctx.lookup("java:comp/env/ejb/DirectorHome");
      Director dir = dirHome.findByPrimaryKey(directorID);
      setDirector(dir);
     } catch(Exception ex)
       {//handle exception}
   }

   public String ejbHomeListMoviesByDirectorAndGenre(String dirID, String genre) {
       String list = null;
       try {
          Collection c = ejbSelectGetMoviesByDirectorAndGenre(dirID, genre);
          Iterator it = c.iterator();
          while (it.hasNext()) {
             list += it.next();
           }
    } catch (Exception ex) { ex.printStackTrace();}
     return list;
  }
```

Here we set our persistent fields, by calling our abstract setters. We must NOT try to reference our own persistent RELATIONSHIPS. That has to wait until WE have been inserted, and that happens only AFTER ejbCreate() completes.

Now it's safe to assign our persistent relationship field for Director. We lookup the Director (using the directorID parameter from create) and call our abstract setter to assign this Director.

A home business method for the client to get a list of all the movies that fit the query. We use an ejbSelect method that returns a Collection of String objects. By using a select method, we let the Container write all the database code!

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

```
public String ejbHomeListAllMovies() {
      String list = null;
        try {
           Collection c = ejbSelectGetAllMovies();
           Iterator ita = c.iterator();
              while (ita.hasNext()) {
                 Movie movie = (Movie) ita.next();
                list += " " + movie.getMovieTitle();
              }

        } catch (Exception ex) { // handle exception}
      return list;
  }

   public String getMovieTitle() {
      return getTitle();
   }

   public String getMovieDirectorName() {
       return getDirector().getDirectorName();
   }

   public abstract String getMovieID();
   public abstract void setMovieID(String i);

   public abstract String getTitle();
   public abstract void setTitle(String t);

   public abstract int getYear();
   public abstract void setYear(int y);

   public abstract String getGenre();
   public abstract void setGenre(String g);

   public abstract Director getDirector();
   public abstract void setDirector(Director d);

   public abstract Collection ejbSelectGetAllMovies() throws FinderException;

   public abstract Collection ejbSelectGetMoviesByDirectorAndGenre(String dir, String
genre) throws FinderException;


   public void unsetEntityContext() { }
   public void ejbLoad() { }
   public void ejbStore() { }
   public void ejbActivate() { }
   public void ejbPassivate() { }
   public void ejbRemove() { }

   public void setEntityContext(EntityContext ctx) {
      context = ctx;
   }
}
```

Another home business method that uses a select method to find all movies. This select returns a Collection of actual Movie objects, so we walk through it and pull out the titles, so that we can return a big String to the client.

Two exposed business methods from the component interface...

We have FIVE virtual fields: four CMP fields and one CMR field for Director.

The two select methods are declared as abstract methods. The Collection doesn't say WHAT the methods will return... but we'll tell the Container that one will return a Collection of Movies and one a Collection of Strings.

The rest are regular old callback methods.

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

the MovieHomeRemote interface

The MovieBean's home interface

```
import javax.ejb.*;
import java.util.*;
import java.rmi.*;


public interface MovieHomeRemote extends EJBHome {

    public MovieRemote create(String movieID, String title, int year, String genre,
                              String directorID) throws CreateException, RemoteException;

    public MovieRemote findByPrimaryKey(String key) throws FinderException,
                                                           RemoteException;

    public Collection findByGenre(String genre) throws FinderException, RemoteException;

    public String listAllMovies() throws FinderException, RemoteException;

    public String listMoviesByDirectorAndGenre(String directorID, String genre)
                                            throws FinderException, RemoteException;

    public Collection findMoviesByKevinDegrees(int degrees) throws FinderException,
                                                                   RemoteException;
}
```

Whoa! There are three finder methods that weren't in the bean class... remember, the Container implements your finder methods--you don't put ANYTHING about finders in your bean class code. You put them ONLY in the home.

(well, that's not completely true...somewhere you have to tell the Container HOW to implement the finders, which brings us to the next page...)

The home business methods all start with the prefix "ejbHome" when they're in the bean class. In the home, they don't have that prefix.

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

***database to entity** mapping*

Mapping from abstract schema to a real database

Sooner or later, you have to tell the Container how and where to manage your persistent data. This happens outside the deployment descriptor, in a vendor-specific way. That means, of course, that you're not tested on it in the exam. Every vendor has their own mechanism, but you'll usually get some sort of a GUI where, if you're lucky, you can tell the Container to connect to a specific database, and then you'll drag and drop or draw connections from CMP fields to columns in one or more tables.

Because this mapping happens outside of EJB, in the server, there's not much we can say about it. The part we're most interested in is mapping from abstract queries to REAL queries, and that happens in a very cool way, using a vendor-independent query language just for your entity bean queries, EJB-QL. We'll look at that on the next page.

```
<abstract-schema-name>MovieSchema</abstract-schema-name>

<cmp-field>
  <field-name>genre</field-name>
</cmp-field>

<cmp-field>
  <field-name>movieID</field-name>
</cmp-field>

<cmp-field>
  <field-name>year</field-name>
</cmp-field>

<cmp-field>
  <field-name>title</field-name>
</cmp-field>
<primkey-field>movieID</primkey-field>
```

Notice that we're not mapping the Director ID column (the one that keeps the foreign key into the Director table) because we don't HAVE a persistent CMP field for the Director ID. Why do that, when we can have the real Director, as an entity bean, instead? That's where our CMR field comes in, but we don't map that to a database. Relationships happen by matching up two BEANS in the <relationships> part of the DD.

MovieID	Title	Genre	DirectorID	Year
12	Crouching Pixels, Hidden Mouse	Action	42	2000
5	The Fifth Array Element	Action	27	2003
22	The Return of the Bean Queen	Fantasy	27	2001
11	Lord of the Loops	Sci-fi	42	2002

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Writing your portable queries, for select and finder methods

EJB-QL is a portable query language that lets you write SQL-like statements write into the deployment descriptor! You put them in the XML and without having to know anything about the real database.

As a Bean Provider, you can happily write all the queries you want, trusting that the Deployer will, in the end, map your CMP fields to real tables and data. All we care about here is writing our queries to match the methods we've chosen to expose to the client in the home and component interfaces.

Select methods are here simply to make your life easier, by letting the Container build the real database access code from your queries.

So the process is usually something like:

1. **Write a home business method**
2. **Implement the home business method to call the ejbSelect method that does the *real* data access.**
3. **Declare the abstract select method in your bean class.**
4. **Write the EJB-QL for the select query.**

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

EJB-QL for the MovieBean

Let's look at the EJB-QL for all the queries the MovieBean needs. Don't worry if you're not understanding all of it; after we walk through the examples, we'll explore the different pieces in detail, and work out the syntax. If you know SQL, most of this will be familiar (although there are few differences you'll see). If you don't know SQL, that's OK. We'll get you started writing queries, and you can use the spec to learn more about the EJB-QL language. ***The exam expects you to know just the basics of EJB-QL.***

In the bean class

The method returns a Collection of Movie objects

```
public abstract Collection ejbSelectGetAllMovies() throws FinderException;
```

In the the deployment descriptor

```
SELECT OBJECT (m) FROM MovieSchema m
```

This says "Return whatever 'm' is representing." In this example, 'm' is said to represent MovieSchema which really means the Movie entity bean type. So this first part says, "Give me Movie beans".

This is just like declaring a variable... "Let 'm' represent the abstract schema from the Movie bean. But you can REALLY think of it like, "Let 'm' represent the Movie component interface type. In other words, Movie beans. So it REALLY means, "Let 'm' represent a Movie bean.

What it really says

"Give me all the Movie beans"

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Using an optional WHERE clause

This example constrains the objects returned from the search, by adding a WHERE clause that says what extra criteria the objects have to meet before they qualify for "things that can be returned from the query." The WHERE clause is your way of saying, "Don't just give me ANY old Movie... I want ONLY the movies that..."

In the bean class

The method returns a Collection of Movie objects

```
public abstract Collection ejbSelectGetAllMovies() throws FinderException;
```

In the the deployment descriptor

"genre" must be CMP field in the Movie bean, which means you can find a getGenre() / setGenre() method pair in the bean class

```
SELECT OBJECT (m)   FROM MovieSchema m   WHERE m.genre = ?1
```

this means the first parameter to the method

This says we want the bean type of whatever 'm' turns out to be...

Let 'm' represent the Movie bean type

restrict the Movies returned to only those where the CMP field "genre" matches the first parameter of the select method The "1" means the first parameter. The "m.genre" uses the dot navigation operator to make a "path" that navigates in an object-oriented way. It's simple—the left hand operand is a type, and the right hand is a member field of that type. It's almost exactly like using regular Java syntax to say, "myMovie.genre", where genre is an instance variable (in this case, a virtual persistent field.)

What it really says

"Give me all the Movie beans that are in the genre matching the first parameter to the ejbSelectGetAllMovies method."

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

path *navigation*

Navigating to another related bean

This example is for a finder method, so you won't find it in the bean class (remember, you declare abstract methods for only the select methods, not the finders. The Container uses your home interface to figure out what the finders are, and then it's up to you to write the EJB-QL for the finders).

You can use the dot operator to navigate not just to a CMP field in the bean, but to another field within a bean referenced in a CMR field. In other words, If Movie has a reference to a Director (and they're in a relationship), you can use the Movie to get to the Director to ask for a CMP value from the Director.

In the home interface

```
public Collection findMoviesByKevinDegrees(int degrees) throws FinderException,
                                                               RemoteException;
```

In the the deployment descriptor

m.director means "the CMR field 'director' in the Movie bean"

"degrees" is a persistent field in the Director bean.

```
SELECT OBJECT (m)   FROM MovieSchema m    WHERE m.director.degrees = ?1
```

This says that we want objects returned whose type is whatever 'm' turns out to be representing...

... and what do you know? It turns out that 'm' means the Movie bean

restrict the Movies returned to only those where the Movie's Director has a CMP "degrees" field with a value that matches the first parameter of the finder.

What it really says

"Give me all the Movie beans that have a Director whose degrees match what the client sent in to the finder method."

OR

"I want to get Movies back, but only those Movies directed by someone who is a specific number of degrees away from Kevin Bacon."*

*Six degrees from Kevin Bacon is a popular game where you try to take a Hollywood type and see how many connections it takes to get that person linked to Kevin Bacon."

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Selecting a value rather than the whole bean

This example is for a home business method that takes two parameters and returns not Movies, but *Strings*.

In the home interface

This method returns a single String, with all the Movie titles that match the query.

```
public String ejbHomeListMoviesByDirectorAndGenre(String dirID, String genre)
```

In the the deployment descriptor

this is different! We're returning just Movie titles, not Movie objects

m.genre means the "genre" CMP field in the Movie bean

```
SELECT m.title  FROM MovieSchema m

WHERE m.director.directorID = ?1  AND m.genre = ?2
```

This restricts the matches to only those Movies with a Director who has an ID equal to the first parameter to the method

AND whose Movie genre value equals the second parameter to the method

What it really says

> ***"Give me back Movie titles (Strings) that have the director and genre I specified in the method arguments."***

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

EJB-QL SELECT

Now that you've seen some queries, we'll look at some of the details and rules.

Query Domains

The domain of a query is a single deployment descriptor. So that means a single ejb-jar file. However many beans you can squish in the ejb-jar, that's how many beans you can search across.

SELECT

The SELECT clause says what kind of thing this query will return.
It can be only one of these two:

① **an abstract schema type for an entity bean**

```
<abstract-schema-name>MovieSchema</abstract-schema-name>

SELECT OBJECT (m) FROM MovieSchema m
```

In formal EJB terms (and in the spec) the abstract schema type as a SELECT variable is called a RANGE variable, because it ranges over the entire entity bean type.

OR

② **a <cmp-field> single value type for an entity bean**

```
<cmp-field>
   <field-name>title</field-name>
</cmp-field>

SELECT m.title  FROM MovieSchema m
```

When you use a <cmp-field> instead of a range variable, we call this a single-valued path expression.

You can't use dot notation to return a CMR field!

As a SELECT type, these are OK:
m.title *or* ***m.genre***

But this is NOT OK:
m.director

You can use a CMP field, but not a CMR field with the dot notation, as the SELECT type.

NOTE: we're deviating from standard spec naming conventions in our abstract schema name, because we want it to be OBVIOUS that we're using the schema name and not, say, the bean name. In the spec, the abstract schema name is the same as the component interface name.

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

EJB-QL SELECT — when to use Object ( )

What's with the OBJECT (m) thing?

It's a structure the J2EE designers put in to be consistent with a version of SQL.

It's annoying.

But you have to do it ***whenever you return a bean type instead of a CMP field type.*** And you must NOT do it ***whenever you return a CMP field type.***

If you return the bean's abstract schema type, use OBJECT( ).

```
<abstract-schema-name>MovieSchema</abstract-schema-name>

SELECT OBJECT (m)
```

If you return a <cmp-field> ***don't*** use OBJECT ( )

```
<cmp-field>
   <field-name>title</field-name>
</cmp-field>

SELECT m.title
```

If you're using the dot, to navigate a path, you don't use the OBJECT ( )

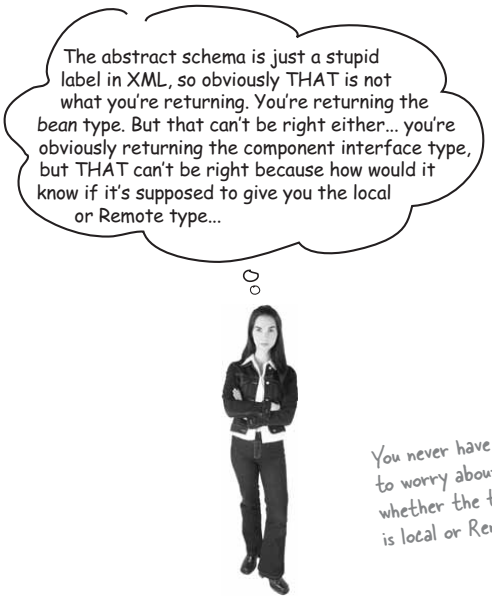
Sharpen your pencil

Circle the EJB-QL statement if it has a legal SELECT clause

```
SELECT OBJECT (m) FROM MovieSchema m

SELECT m.title  FROM MovieSchema m

SELECT m FROM MovieSchema m

SELECT OBJECT (m.title)  FROM MovieSchema m
```

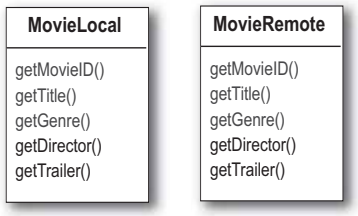

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

What does it MEAN to return an abstract schema type?



The MovieSchema abstract schema name represents the bean type.

As if by magic, the Container knows which interface view to return, local or Remote, depending on whether the invocation of the query came from a home or Remote interface.



When you see this:

MovieSchema m

Think this:

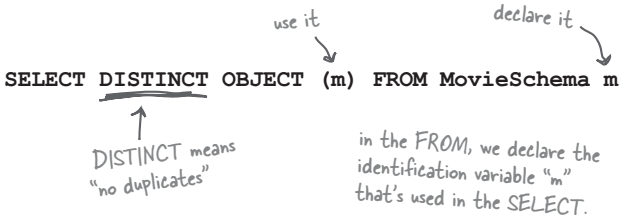
m = the component interface

(and we'll let the Container sort out the Remote vs. local issue)

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

SELECT and FROM are mandatory!

You gotta have SELECT and FROM no matter what, but the WHERE clause is optional. With SELECT, you can also use the keyword DISTINCT to say that you don't want any duplicates returned in your Collection.



The FROM clause declares the identification variable (like the "m" in our example query). This also defines the domain of the query by saying to the Container, "Here's where I want you to be looking..." The order in which the FROM and SELECT might feel a little strange if you're not used to SQL. But think of it like someone saying, "I want you to go get my *things*, and the *things* I'm talking about are FROM my top drawer."

If you need to declare more identification variables to use later in the query, you can separate them with a comma:

```
SELECT OBJECT (m)
FROM MovieSchema m, DirectorSchema d
```

We declared two different identifiers, and defined the domain of the query to those two abstract schemas.

You can also use the AS keyword. It's optional, but you might see it on the exam or in someone else's code, or your company style guide might insist on using it. We're too lazy to keep typing it.

```
SELECT OBJECT (m) FROM MovieSchema AS m
```


Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Identifiers

When you make an identifier, you have to follow the Java programming language guidelines for what you can name things. You can't use the other Java identifiers

Identifiers

Must be valid Java identifiers

Must NOT be same as an <abstract-schema-name> or <ejb-name> anywhere in the DD

Must NOT be one of the EJB-QL reserved words

EJB-QL reserved words

SELECT	*OR*
FROM	*BETWEEN*
WHERE	*LIKE*
DISTINCT	*IN*
OBJECT	*AS*
NULL	*UNKNOWN**
TRUE	*EMPTY*
FALSE	*MEMBER*
NOT	*OF*
AND	*IS*

Watch out for questions with name conflicts!

Remember that an <ejb-name> must be unique for each bean in the ejb-jar, which means unique for one complete DD. The same is true for the <abstract-schema-name>; it must be unique in the DD.

You already know that you can't use an identification variable in your query that matches an <ejb-name> or <abstract-schema-name> in the DD, but keep in mind that EJB-QL is not case-sensitive!

So, you must NOT do this:

```
SELECT OBJECT (movie)
FROM Movie movie
```

In EJB-QL, "Movie" is the same as "movie", and you can't have conflicting names.

* not used in the current version; it's future use is unknown...

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

The WHERE clause

A WHERE clause is optional, and let's you put extra conditions on what's returned. Remember that without one, we got everything of the type used in the SELECT.

NOT using WHERE

```
SELECT OBJECT (m) FROM MovieSchema m
```

Existing Entities

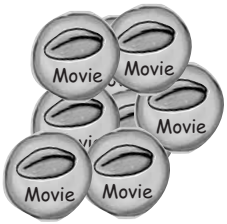
Entities returned from the query

Using WHERE

Enclose a String literal in single quotes

```
SELECT OBJECT (m)  FROM MovieSchema m   WHERE m.genre = 'Horror'
```

Existing Entities



Entities returned from the query

Using WHERE, we get only the movies that have the value "Horror" in their CMP field "genre". (In other words, those movies whose getGenre() method would return "Horror".)

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

The WHERE clause

You can use literals

```
WHERE m.director.degrees > 3
WHERE m.genre = 'Horror'
```

You can use input parameters

```
WHERE m.director.degrees > ?1
```

And the parameters don't have to be in order... that's why they have numbers!

```
WHERE m.genre = ?2 AND m.director.degrees < ?1
```

```
public String ejbHomeListMoviesByDirectorAndGenre(String dirID, String genre)
```

You can do comparisons

```
WHERE m.director.degrees < 5
```

But you better compare apples to apples

```
WHERE m.director < 5
```

No!!

You can't compare a Director bean to a numeric type!

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

The problem with using Collection types

Imagine you want to use the DirectorSchema to return all the Directors that have made horror movies. Sounds pretty simple... Director has a CMR field for movies, and each movie has a CMP field for genre. So we come up with something like this:

```
SELECT OBJECT (d)
FROM DirectorSchema d
WHERE d.movies.genre = 'Horror'
```

Looks good on first glance, but WATCH OUT! This is NOT LEGAL.

Why can't you do this?

Let's back up and imagine this is Java code. How do you normally use the dot operator in Java? You might do something like:

```
Owner o = new Owner();
o.getDog().bark();
```

method of Owner

method of Dog

This is fine... we can go through the Owner, get its Dog, and call bark() on the Owner's dog.

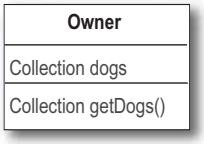
The Owner has a getDog() method, and the Dog has a bark(). No problem.

But what if we changed the Owner to allow one Owner to have multiple Dog objects? Then what happens...

```
Owner o = new Owner();
o.getDogs().bark();
```

method of Owner

YIKES!! Now who are we really calling this method on? A Dog? Or a Collection...



Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

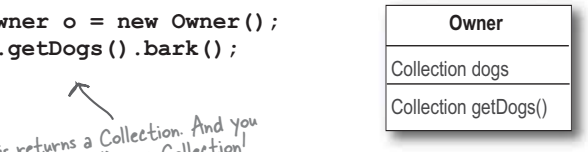
using collections in a path

Collections don't bark()!

You can't use the dot navigation when something you're using *in* the path is a Collection. For example you can't do this:

```
Owner o = new Owner();
o.getDogs().bark();
```

This returns a Collection. And you can't call bark() on a Collection!



It's the Dog objects IN the collection that can bark()! So, in Java, you'd have to first access the individual Dog elements in the collection, and one at a time ask each Dog to bark(). The Dog code above is almost exactly like this WHERE clause we had before:

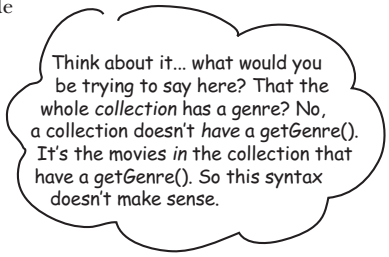
```
WHERE d.movies.genre = 'Horror'
```

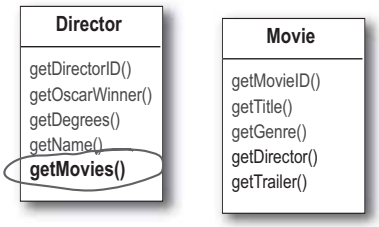
No!!

And since "movies" and "genre" are virtual fields, with accessors, you can think of it just like this:

```
d.getMovies().getGenre() = 'Horror'
```

No!!





Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

The IN operator lets you say "For an individual element IN the Collection..."

The IN operator for the FROM clause let's you refer to a Collection, but name an identifier as representing ONE member of the Collection.

```
SELECT DISTINCT OBJECT (d)
FROM DirectorSchema d, IN(d.movies) m
WHERE m.genre = 'Horror'
```

Now "m" represents an individual Movie IN the movies Collection (a CMR field of Director)

Let "m" represent an individual Movie IN the Collection. So, "m" does NOT represent the Collection type, but rather the Movie type (the type the Collection is holding.)

This query looks through all of the Directors, and looks through each of the Movies of each Director, and looks for a Movie with a genre that matches 'Horror'. The query returns a set (no duplicates because we said "DISTINCT" of Directors

The IN operator is overloaded in EJB-QL!

The IN operator in the FROM clause is different from the IN comparison operator used in a WHERE clause.

In the WHERE clause, you can say:

d.genre IN ('Horror', 'Fantasy')
and it is just a shortcut for saying:
d.genre = 'Horror' OR d.genre = 'Fantasy'

But it has nothing to do with the IN operator used in a FROM clause to designate individual members IN a Collection. So just remember:

FROM clause: IN says, "Look at the individual objects in the Collection, and rather than using the Collection as the type, use the type of the items IN the Collection."

WHERE clause: IN says, "See if the field value matches one of the literals in this set..."

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

The BETWEEN expression

Let's say you want to find movies in the database made in the 60's and 70's, but you want to exclude any movies made in 1975 or 1976, really bad movie years.

You could say:

```
SELECT DISTINCT OBJECT (m)
FROM MovieSchema m
WHERE (m.year > 1959 AND m.year < 1975) OR
       (m.year > 1976 AND m.year < 1980)
```

Or, you could do it the cooler way:

```
SELECT DISTINCT OBJECT (m)
FROM MovieSchema m
WHERE (m.year BETWEEN 1960 AND 1979) AND
       (m.year NOT BETWEEN 1975 AND 1976)
```

In this example notice that BETWEEN is INclusive (i.e. the years 1960 and 1979 WILL be included), and that NOT BETWEEN is EXclusive

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

The other "IN"

We've used IN with a FROM clause to help navigate through collections, but IN has another personality, you might say IN has been overloaded for use in the WHERE clause. Let's say that you want to select movies from a subset of all the movie genres in your database:

```
SELECT DISTINCT OBJECT (m)
FROM MovieSchema m
WHERE m.genre IN('horror', 'mystery')
```

This query would return only those movies whose genre is either "horror" or "mystery". This flavor of IN lets you created something akin to an enumerated list.

And there's always NOT IN for those times when you need some action:

```
SELECT DISTINCT OBJECT (m)
FROM MovieSchema m
WHERE m.genre NOT IN('romance', 'comedy')
```

This query would return only those movies whose genre is NOT "romance" or "comedy"

"!=" is NOT the same as "< >"

When comparing values in a WHERE clause, "< >" means "not equals". You use THAT instead of "!=". But coming from Java, that "!=" can look so right... when you see it on a question. Don't be tempted.

String literals are in SINGLE quotes!

While we're on the topic of inconsistencies between Java syntax and EJB-QL, don't be tricked by a String in double quotes. This is WRONG in EJB-QL:
WHERE m.foo IN ("bar", "baz")
It must be ('bar', 'baz')

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

The IS EMPTY comparison expression

This baby will let you know whether the collection in question is empty:


```
SELECT DISTINCT OBJECT (d)
FROM DirectorSchema d
WHERE d.movies IS EMPTY
```

This query returns only those directors who have not made any movies.

The IS NOT EMPTY comparison expression

IS NOT EMPTY lets you say, "Don't give me anything where this field is empty..."

```
SELECT DISTINCT OBJECT (d)
FROM DirectorSchema d
WHERE d.movies IS NOT EMPTY
```

This query returns only those directors who have at least one movie in their movies collection.

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

The LIKE expression

Like expressions are used to compare single value path expressions with a String literal. The power of LIKE is that the String literal can have wildcard elements, giving you a simple pattern matcher. The "%" wildcard matches against 0 to many characters; the "_" matches against a single character only.

```
SELECT DISTINCT OBJECT (d)
FROM DirectorSchema d
WHERE d.phone LIKE '719%'
```

The "%" symbol is a wildcard for 0 to many characters

This query would return only those directors who have phone numbers beginning with the digits '719'. (Hey, phone numbers can to be Strings, they have those dashes...)

```
SELECT DISTINCT OBJECT (m)
FROM MovieSchema m
WHERE m.filmcode LIKE '7_mm'
```

This query searches for movies whose film codes are in the 7Xmm family.

The "_" symbol is a wildcard for a single character

The NOT LIKE expression

```
SELECT DISTINCT OBJECT (m)
FROM MovieSchema m
WHERE m.filmcode NOT LIKE '%mm'
```

You can probably guess what this one does... returns all movies whose film codes DON'T end in "mm".

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

BULLET POINTS

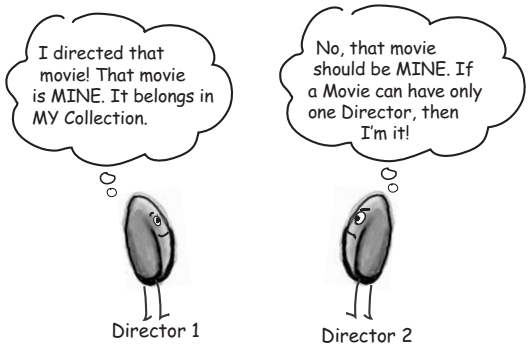
- Entity beans can have persistent relationships with other entity beans.
- A container-managed relationship (CMR) field is defined in the bean class just as a CMP field is—with a pair of abstract getters and setters that return either the local component interface of the bean, or a Collection.
- If the virtual field is a Collection, it must be declared as either java.util.Collection or java.util.Set. No other Collection type is allowed as the declared return type of a CMR field.
- Relationships have multiplicities—they can be one-to-one, one-to-many, or many-to-many. (Many-to-one works in the same way that one-to-many works; it just depends on whose point of view you're using.)
- For example, a Movie bean has a many-to-one relationship with a Director bean. Each Movie has only one Director, but a Director can have many Movies.
- Multiplicity affects the return type of the virtual field. Movie has only one Director, so the CMR field in the bean is getDirector(), that returns a reference to a Director's local component interface. But a Director has many movies, so a Director's virtual field is getMovies(), which returns a Collection.
- A CMP bean must define an "abstract-persistence-schema in the DD, that lists each of the bean's CMP fields, and also identifies which of the fields is the primary key (unless it's a compound primary key). The DD must always define the type of the primary key, even if the primary key is not a field of the bean. (Remember, if the primary key is not a field of the bean, it must be a primary key class *composed* of CMP fields from the bean.)
- Relationship (CMR) fields are not defined in the <enterprise-bean> portion of the DD (where you define your CMP fields), but are instead defined in the <relationships> section of the DD.
- Each relationship must have two partners, with each partner described in an <ejb-relationship-role> element that includes the CMR field name, the source (<ejb-name>) for the partner, the multiplicity (i.e. how many of this bean will the *other* partner have), and an optional <cascade-delete/> (which says, "Delete me if my partner is deleted."
- Relationships can be one-way (unidirectional) or two-way (bi-directional). A unidirectional relationship is described in the DD in the same way as a bi-directional relationship, except that only one of the two partners has a CMR field. This means bean A has a reference to bean B (they're in a relationship), but bean B does not have a reference back to A. Directionality does not affect multiplicity or cascade-delete; it just means that only one of the two partners has a reference to the other. To the Container, they're still in a relationship.
- Cascade-deletes tell the Container to delete the bean with the <cascade-delete/> tag when the bean's partner is deleted. This works only if the bean has a multiplicity of one. (You wouldn't want to delete a Director just because one of this Movies was deleted; but you might want to delete a Movie if its sole Director was deleted.)
- A bean cannot access its CMR fields in ejbCreate(); it must wait until ejbPostCreate() before, say, assigning the CMR virtual field an actual value. In other words, you can't use your abstract getters and setters for your CMR fields until after ejbCreate() completes.
- EJB-QL is a portable query language you can use to specify your queries for both finder and select methods.
- In an EJB-QL query, the SELECT and FROM clauses are required; the WHERE clause is optional.
- A SELECT can return either an <abstract-schema-type> (which really means the bean's local component interface type), or a single-value field.
- If you use the dot notation for navigating a path (for example, m.genre), you can't use a Collection type as part of the path. To use a Collection type as part of a FROM clause, you must enclose it using the IN(d.movies) operator.

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Relationship assignments

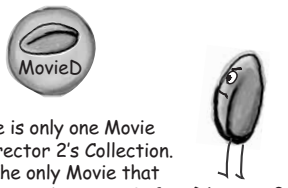
A Movie can have only one Director. If you set a Movie's Director field to Director 1, the Movie will be in Director 1's collection of Movies, but in no other Director's collection. If you later reassign the Movie's Director field so that the Movie has a *different* Director—Director 2—the previous Director (Director 1) will no longer have that Movie in its Collection.

This all happens automatically! To maintain the integrity of the database, the Container manages both sides of the relationship at all times.



Before:

Three Movies in Director 1's Movie Collection. If you call getDirector() on any of these Movies, they will return Director 1.

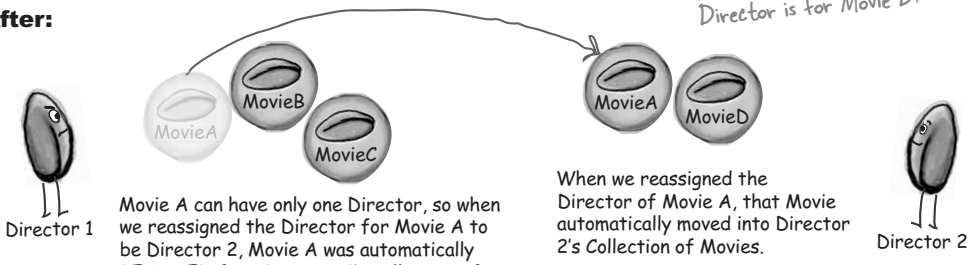


There is only one Movie in Director 2's Collection. It's the only Movie that will return Director 2 if you call getDirector() on the Movie.

```
movieA.setDirector(movieD.getDirector());
```

this says, "set Movie A's Director to be whatever the Director is for Movie D."

After:



Movie A can have only one Director, so when we reassigned the Director for Movie A to be Director 2, Movie A was automatically REMOVED from Director 1's Collection of Movies. Remember, the Container manages both sides of the relationship!

When we reassigned the Director of Movie A, that Movie automatically moved into Director 2's Collection of Movies.

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

assignments in relationships

If the multiplicity of the relationship field is ONE, it's a MOVE
If the multiplicity is MANY, it's a COPY

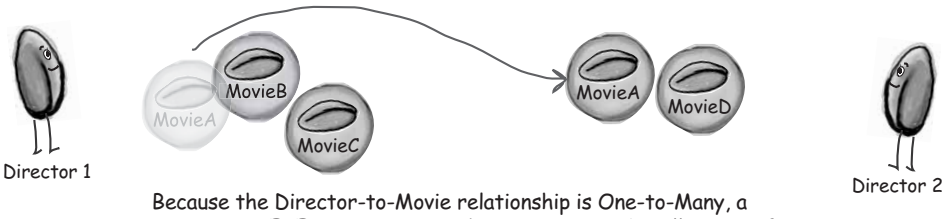
```
<ejb-relationship-role-name>MovieBean</ejb-relationship-role-name>
<multiplicity>Many</multiplicity>

<ejb-relationship-role-name>DirectorBean</ejb-relationship-role-name>
<multiplicity>One</multiplicity>
```

When we assigned Director 2 to Movie A, the Movie moved from Director 1's Collection to Director 2's Collection. The Director CMR field has a multiplicity of ***One*** in its relationship with Movie, so a Movie can't exist in the Collection of two different Directors, because a Movie can have only ***One*** Director.

But just because we move the Movie, doesn't mean the Director also moves. The multiplicity of Movie is Many, so when you assign another Movie's Director to a *different* Movie, that Director reference is *copied*, and both Movies will now have a reference.

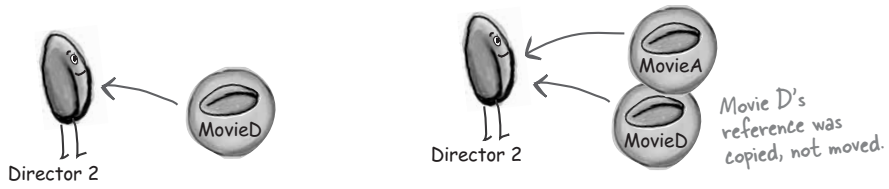
The MOVIE reference was MOVED from one Director to another

```
movieA.setDirector(movieD.getDirector());
```



Because the Director-to-Movie relationship is One-to-Many, a Movie can NEVER exist in more than one Director's Collection. If you change a Movie's Director, the Movie is not copied! It is ***moved*** to the new Director's Collection.

The DIRECTOR reference was COPIED from one Movie to another



Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Multiplicity and Assignments

Given this relationship:

```
<ejb-relationship-role-name>MovieBean</ejb-relationship-role-name>
<multiplicity>One</multiplicity>

<ejb-relationship-role-name>TrailerBean</ejb-relationship-role-name>
<multiplicity>One</multiplicity>
```

And this current scenario:

Draw what the picture will look like after you run:

```
movie1.setTrailer(movie2.getTrailer());
```

Draw your picture here...

Answer these questions:

What is the result of: `movie1.getTrailer()`

What is the result of: `movie2.getTrailer()`

What is the result of: `trailerA.getMovie()`

What is the result of: `trailerB.getMovie()`

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

exercise solution

Multiplicity and Assignments

Given this bi-directional relationship:

```
<ejb-relationship-role-name>MovieBean</ejb-relationship-role-name>
<multiplicity>One</multiplicity>

<ejb-relationship-role-name>TrailerBean</ejb-relationship-role-name>
<multiplicity>One</multiplicity>
```

And this current scenario:

Draw what the picture will look like after you run:

```
movie1.setTrailer(movie2.getTrailer());
```

Answer these questions:

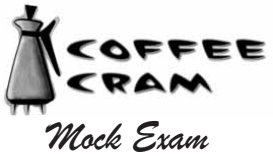
`movie1.getTrailer()` — returns trailerB

`movie2.getTrailer()` — returns null! Movie has a multiplicity of One, so a Trailer can have only ONE Movie. A Trailer can't be with both Movie 1 and Movie 2

`trailerA.getMovie()` — also returns null... Trailer has a multiplicity of One, so there can be only one Trailer per Movie. When TrailerB went to Movie 1, TrailerA was kicked out, disconnected from the relationship.

`trailerB.getMovie()` — returns Movie1, not that it has been assigned to Movie1, the Container updates the field returned from the Trailer's getMovie().

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01



1 What's true for an entity bean provider using container-managed persistence to create persistent relationships? (Choose all that apply.)

- ❑ A. A local interface is required for such a bean to have a bidirectional relationship with another entity bean.
- ❑ B. Such a bean can have relationships with only session beans.
- ❑ C. Relationships can be only one-to-one or one-to-many.
- ❑ D. A getter method return type can be java.util.List.

2 What's true for an entity bean provider using container-managed persistence to create persistent relationships? (Choose all that apply.)

- ❑ A. A remote interface is required for such a bean to have a bidirectional relationship with another entity bean.
- ❑ B. Such a bean can have relationships with only message-driven beans.
- ❑ C. A get method return type can use java.util.Map.
- ❑ D. Relationships can be one-to-one, one-to-many, or many-to-many.

3 When a remote client invokes a method on an entity bean using container-managed persistence, and that bean has already been removed, what exception will be thrown?

- ❑ A. `javax.ejb.AccessLocalException`
- ❑ B. `javax.ejb.ObjectNotFoundException`
- ❑ C. `java.rmi.NoSuchObjectException`
- ❑ D. `java.rmi.StubNotFoundException`
- ❑ E. `javax.ejb.NoSuchEntityException`

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

4 Given the container-managed unidirectional relationship:

Foo (0-1) —> Bar (0-1)

And the object relations:

f1 —> b1
f2 —> b2

What will be true after the following code runs? (Choose all that apply.)

```
f2.setBar(f1.getBar());
```

- ❑ A. `f1.getBar() == null`
- ❑ B. `b2.getFoo() == null`
- ❑ C. `b1.getBar() == null`
- ❑ E. none of the above

5 If an entity bean A has been removed from a relationship with bean B, in which case(s) will bean A's accessor method for bean B return a non-null value? (Choose all that apply.)

- ❑ A. one-to-one
- ❑ B. many-to-one
- ❑ C. many-to-many
- ❑ D. all of the above
- ❑ E. none of the above

6 Which deployment descriptor element's value(s) must be a type of collection? (Choose all that apply.)

- ❑ A. `<ejb-name>`
- ❑ B. `<cmr-field>`
- ❑ C. `<cmr-field-type>`
- ❑ D. `<ejb-relation>`

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

7 Which deployment descriptor element(s) must have exactly two declarations of another deployment descriptor element? (Choose all that apply.)

- ❑ A. `<ejb-name>`
- ❑ B. `<cmr-field>`
- ❑ C. `<cmr-field-type>`
- ❑ D. `<ejb-relation>`
- ❑ E. `<ejb-relationship-role>`

8 Which set(s) of elements are mandatory within an ejb-relationship-role element? (Choose all that apply.)

- ❑ A. `<cmr-field>`, `<multiplicity>`
- ❑ B. `<cmr-field>`, `<relationship-role-source>`
- ❑ C. `<multiplicity>`, `<relationship-role-name>`
- ❑ D. `<multiplicity>`, `<relationship-role-source>`
- ❑ E. `<relationship-role-name>`, `<relationship-role-source>`

9 Which are valid values for a cmr-field-type element? (Choose all that apply.)

- ❑ A. String
- ❑ B. Integer
- ❑ C. `java.util.Set`
- ❑ D. `java.util.List`
- ❑ E. `java.util.Collection`

10 Which set(s) of elements are mandatory within an ejb-relation element?

- ❑ A. `<ejb-relation-name>`, `<ejb-relationship-role>`
- ❑ B. `<ejb-relationship-role>`, `<ejb-relationship-role>`
- ❑ C. `<description>`, `<ejb-relation-name>`, `<ejb-relationship-role>`
- ❑ D. `<ejb-relation-name>`, `<ejb-relationship-role>`, `<ejb-relationship-role>`

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

11 Given CMP beans CustomerBean, OrderBean, and LineItemsBean with the following relationships:

CustomerBean (1) <—> OrderBean (n)
OrderBean (1) <—> LineItemsBean (n)

and the following EJB QL query:

```
SELECT DISTINCT OBJECT (c)
FROM Customer c, IN (c.Order) o, IN (o.lineItems) li
WHERE li.product_type = 'refrigerator'
```

Which of the following properly describes the result of the query? (Choose all that apply.)

- ❏ A. The query is invalid.
- ❏ B. All orders that include a line item that refers to a refrigerator.
- ❏ C. All line items that refer to a refrigerator.
- ❏ D. All customers that have order(s) that refer to a refrigerator.

12 Given CMP beans CustomerBean, OrderBean, and LineItemsBean with the following relationships:

CustomerBean (1) <—> OrderBean (n)

OrderBean (1) <—> LineItemsBean (n)

Which will return all orders that have line items? (Choose all that apply.)

- ❏ A. SELECT DISTINCT o
  FROM Order o, IN (o.lineItems) li
- ❏ B. SELECT DISTINCT OBJECT(o)
  FROM Order o, IN (o.lineItems) li
- ❏ C. SELECT OBJECT(o)
  FROM Order o
  WHERE o.lineItems = 0
- ❏ D. SELECT OBJECT(o)
  FROM Order o

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

13 What's true about EJB QL path expressions? (Choose all that apply.)

- ❑ A. In a path expression the `(.)` is considered the navigation operator.
- ❑ B. Path expressions can terminate with either `cmr-field` or `cmp-field`.
- ❑ C. A path expression that ends in a `cmr-field` cannot be further composed.
- ❑ D. A path expression can end with a single value or a collection value.

14 What's true about EJB QL queries? (Choose all that apply.)

- ❑ A. Of the three clause types, SELECT, FROM, and WHERE, only the SELECT clause is required.
- ❑ B. The SELECT clause designates query domain.
- ❑ C. The WHERE clause determines the types of objects to be selected.
- ❑ D. An EJB QL query may have parameters.

15 What's true about EJB QL WHERE clauses? (Choose all that apply.)

- ❑ A. Identification variables used in a WHERE clause can be defined only in the FROM clause.
- ❑ B. Identification variables can represent a single value or a collection.
- ❑ C. The number of input parameters must equal the number of parameters for the finder or selector method.
- ❑ D. Input clauses can only be used in WHERE clauses.

16 Given the EJB QL expression:

```
p.discount BETWEEN 10 AND 15
```

Which expression is equivalent?

- ❑ A. `p.discount > 10 AND p.discount < 15`
- ❑ B. `p.discount >= 10 AND p.discount < 15`
- ❑ C. `p.discount > 10 and p.discount <= 15`
- ❑ D. `p.discount >= 10 and p.discount <= 15`

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

17 What's true about EJB QL, IN expressions? (Choose all that apply.)

- ❑ A. The **`single_valued_path_expression`** must have a String value.
- ❑ B. The NOT logical operator can be used in an IN expression.
- ❑ C. The string literal list can be empty.
- ❑ D. The following expression is legal: **`o.country`** IN ("UK", "US")

18 Given the EJB QL expression:

product.code LIKE 'F_s'

Which value(s) would result in a true comparison? (Choose all that apply.)

- ❑ A. Fs
- ❑ B. F1s
- ❑ C. FXs
- ❑ D. F37s
- ❑ E. Fits

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

Mock Exam Answers

1 What's true for an entity bean provider using container-managed persistence to create persistent relationships? (Choose all that apply.)

- [x] A. A local interface is required for such a bean to have a bidirectional relationship with another entity bean.
- [ ] B. Such a bean can have relationships with only session beans.
- [ ] C. Relationships can be only one-to-one or one-to-many.
- [ ] D. A getter method return type can be java.util.List. – must be Collection or Set

2 What's true for an entity bean provider using container-managed persistence to create persistent relationships? (Choose all that apply.)

- [ ] A. A remote interface is required for such a bean to have a bidirectional relationship with another entity bean. – must be local
- [ ] B. Such a bean can have relationships with only message-driven beans.
- [ ] C. A get method return type can use java.util.Map. – only Collection or Set
- [x] D. Relationships can be one-to-one, one-to-many, or many-to-many.

3 When a remote client invokes a method on an entity bean using container-managed persistence, and that bean has already been removed, what exception will be thrown? (spec: 132)

- [ ] A. `javax.ejb.AccessLocalException`
- [ ] B. `javax.ejb.ObjectNotFoundException`
- [x] C. `java.rmi.NoSuchObjectException`
- [ ] D. `java.rmi.StubNotFoundException`
- [ ] E. `javax.ejb.NoSuchEntityException`

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

mock exam answers

4 Given the container-managed unidirectional relationship:

(spec: 138)

Foo (0-1) —> Bar (0-1)

And the object relations:

f1 —> b1
f2 —> b2

What will be true after the following code runs? (Choose all that apply.)

```
f2.setBar(f1.getBar());
```

The setBar() method "breaks" both of the existing object relationships and creates a single new one.

- ☑ A. `f1.getBar() == null`
- ☑ B. `b2.getFoo() == null`
- ❑ C. `b1.getFoo() == null`
- ❑ D. none of the above

5 If an entity bean A has been removed from a relationship with bean B, in which case(s) will bean A's accessor method for bean B return a non-null value? (Choose all that apply.)

(spec: 132)

- ❑ A. one-to-one
- ❑ B. many-to-one
- ☑ C. many-to-many
- ❑ D. all of the above
- ❑ E. none of the above

6 Which deployment descriptor element's value(s) must be a type of collection? (Choose all that apply.)

(spec: 463–464)

- ❑ A. `<ejb-name>`
- ❑ B. `<cmr-field>`
- ☑ C. `<cmr-field-type>` – Collection or Set.
- ❑ D. `<ejb-relation>`

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

7 Which deployment descriptor element(s) must have exactly two declarations of another deployment descriptor element? (Choose all that apply.) (spec: 467–468)

- ❑ A. **<ejb-name>**
- ❑ B. **<cmr-field>**
- ❑ C. **<cmr-field-type>**
- ☑ D. **<ejb-relation>** its gotta have two <ejb-relationship-role> sub elements
- ❑ E. **<ejb-relationship-role>**

8 Which set(s) of elements are mandatory within an ejb-relationship-role element? (Choose all that apply.) (spec: 468)

- ❑ A. **<cmr-field>, <multiplicity>**
- ❑ B. **<cmr-field>, <relationship-role-source>**
- ❑ C. **<multiplicity>, <relationship-role-name>**
- ☑ D. **<multiplicity>, <relationship-role-source>** who's in the relationship, and how are they related
- ❑ E. **<relationship-role-name>, <relationship-role-source>**

9 Which are valid values for a cmr-field-type element? (Choose all that apply.) (spec: 464)

- ❑ A. String
- ❑ B. Integer
- ☑ C. **java.util.Set** Either a local interface type, or a Collection or Set of the local interface type
- ❑ D. **java.util.List**
- ☑ E. **java.util.Collection**

10 Which set(s) of elements are mandatory within an ejb-relation element? (spec: 467–468)

- ❑ A. **<ejb-relation-name>, <ejb-relationship-role>**
- ☑ B. **<ejb-relationship-role>, <ejb-relationship-role>** gotta have two partners
- ❑ C. **<description>, <ejb-relation-name>, <ejb-relationship-role>**
- ❑ D. **<ejb-relation-name>, <ejb-relationship-role>, <ejb-relationship-role>**

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

mock exam answers

11 Given CMP beans CustomerBean, OrderBean, and LineItemsBean with the following relationships: (spec: 233)

CustomerBean (1) <—> OrderBean (n)
OrderBean (1) <—> LineItemsBean (n)

and the following EJB QL query:

```
SELECT DISTINCT OBJECT (c)
FROM Customer c, IN (c.Order) o, IN (o.lineItems) li
WHERE li.product_type = 'refrigerator'
```

Which of the following properly describes the result of the query? (Choose all that apply.)

The SELECT clause specifies that the query must return Customer objects.

- ❑ A. The query is invalid.
- ❑ B. All orders that include a line item that refers to a refrigerator.
- ❑ C. All line items that refer to a refrigerator.
- ☑ D. All customers that have order(s) that refer to a refrigerator.

12 Given CMP beans CustomerBean, OrderBean, and LineItemsBean with the following relationships: (spec: 236)

CustomerBean (1) <—> OrderBean (n)

OrderBean (1) <—> LineItemsBean (n)

Which will return all orders that have line items? (Choose all that apply.)

- ❑ A. SELECT DISTINCT o
  FROM Order o, IN (o.lineItems) li
- ☑ B. SELECT DISTINCT OBJECT(o)
  FROM Order o, IN (o.lineItems) li
- ❑ C. SELECT OBJECT(o)
  FROM Order o
  WHERE o.lineItems = 0
- ☑ D. SELECT OBJECT(o)
  FROM Order o

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

13 What's true about EJB QL path expressions? (Choose all that apply.) (spec: 225–226)

- ☑ A. In a path expression the **`(.)`** is considered the navigation operator.
- ☑ B. Path expressions can terminate with either **`cmr-field`** or **`cmp-field.`**
- ☐ C. A path expression that ends in a **`cmr-field`** cannot be further composed.
- ☑ D. A path expression can end with a single value or a collection value.

14 What's true about EJB QL queries? (Choose all that apply.) (spec: 218–219)

- ☐ A. Of the three clause types, SELECT, FROM, and WHERE, only the SELECT clause is required.
- ☐ B. The SELECT clause designates query domain.
- ☐ C. The WHERE clause determines the types of objects to be selected.
- ☑ D. An EJB QL query may have parameters.

15 What's true about EJB QL WHERE clauses? (Choose all that apply.) (spec: 227–228)

- ☑ A. Identification variables used in a WHERE clause can be defined only in the FROM clause.
- ☐ B. Identification variables can represent a single value or a collection.
- ☐ C. The number of input parameters must equal the number of parameters for the finder or selector method.
- ☑ D. Input clauses can only be used in WHERE clauses.

16 Given the EJB QL expression: (spec: 229)

```
p.discount BETWEEN 10 AND 15
```

Which expression is equivalent?

- ☐ A. **`p.discount > 10 AND p.discount < 15`**
- ☐ B. **`p.discount >= 10 AND p.discount < 15`**
- ☐ C. **`p.discount > 10 and p.discount <= 15`**
- ☑ D. **`p.discount >= 10 and p.discount <= 15`**

Head First EJB™ By Bert Bates, Kathy Sierra ISBN: 0596005717 Publisher: O'Reilly
Print Publication Date: 2003/10/01

mock exam answers

17 What's true about EJB QL, IN expressions? (Choose all that apply.) (spec: 229–230)

- [x] A. The **`single_valued_path_expression`** must have a String value.
- [x] B. The NOT logical operator can be used in an IN expression.
- [ ] C. The string literal list can be empty.
- [ ] D. The following expression is legal: **`o.country`** IN ("UK", "US") watch out for double quotes!

18 Given the EJB QL expression: (spec: 230)

product.code LIKE 'F_s'

Which value(s) would result in a true comparison? (Choose all that apply.)

- [ ] A. Fs
- [x] B. F1s
- [x] C. FXs
- [ ] D. F37s
- [ ] E. Fits

The % character is a wildcard for a sequence of characters. The _ is a wildcard for a single character