

An introduction to EJB-CMP/CMR, Part 3

Presented by developerWorks, your source for great tutorials

ibm.com/developerWorks

Table of Contents

If you're viewing this document online, you can click any of the topics below to link directly to that section.

1. About this tutorial	2
2. EJB-QL and your example	5
3. Using simple EJB-QL commands	8
4. Conclusion	14
5. Feedback	16

Section 1. About this tutorial

Should I take this tutorial?

This is the third part of a tutorial series on Container-managed persistence and relationships in Enterprise JavaBeans 2.0. This part begins on EJB Query Language (EJB-QL) and continues into part 4 that wraps up the subject.

This tutorial assumes that you have completed the first and second part of this series. If you are not familiar with EJB or you need to refresh your memory, I recommend that you read "Enterprise JavaBeans Fundamentals," an excellent IBM tutorial written by two excellent authors, Richard Monson-Haefel and Tim Rohaly. Another good reference work is *Enterprise JavaBeans Developer's Guide to the 2.0 Specification for Trivera Technologies*, which was recently updated. You can find links to both on the [Resources](#) on page 15 panel at the end of this tutorial.

In addition, you should have some background knowledge of SQL. You will also need familiarity with XML, as you will be editing the deployment descriptor. And some experience with the Ant build tool will be helpful.

What this tutorial covers

In the first tutorial in this series, you read about CMP/CMR concepts. Then you created a simple EJB 2.0 CMP entity bean with a finder method implemented in EJB-QL. When you are done with this tutorial series, you will be able to create complex EJB-QL queries.

In the second tutorial, you created the following relationships between CMP entity beans:

- One-to-one
- One-to-many
- Many-to-many

In the third and fourth tutorial in this series, you'll use these relationships as raw material for EJB-QL queries to create select and finder methods. You will build on the example developed in the first two tutorials to cover complex query construction with EJB-QL. You will also create a simple client that accesses the functionality you create to run the queries and test that they actually work.

This third tutorial covers EJB-QL basics, including:

- Writing finder methods in the home interface
- Writing EJB-QL in deployment descriptors
- Finder methods
- The `IN` operator in the `FROM` clause
- The `IN` operator in the `WHERE` clause
- The `MEMBER OF` operator

The fourth and last tutorial in this series, coming soon, covers advanced EJB-QL, including:


- Comparison operators, including `LIKE`
- Select methods
- Logical operators
- The `BETWEEN` clause
- `IS NULL`

About the Author



[Rick Hightower](#), Director of Development at [eBlox](#), has over a decade of experience as a software developer. He leads the adoption of new processes like Extreme Programming, and technology adoption like adoption of CMP and CMR.

Rick's publications include [Java Tools for eXtreme Programming](#), which covers deploying and testing J2EE projects (published by [John Wiley](#)), contributions to Java Distributed Objects (published by Sams), and several articles in [Java Developer's Journal](#).

<p><i>Java Tools for XP</i></p>		<p>Covers creating, testing and deploying J2EE applications using:</p> <ul style="list-style-type: none"> • JUnit, • Cactus, • JMeter and • Ant, etc.
---------------------------------	--	---

Also expect to see his book on Jython from Addison Wesley in the near future.

Rick has also taught classes on developing Enterprise JavaBeans, JDBC, CORBA, Applets as CORBA clients, etc.

Rick is also updating the next version of the Enterprise JavaBeans Developer's Guide to the 2.0 Specification for TriveraTech. The last version of this guide, which covered EJB 1.1 was distributed to over 100,000 developers. This free guide discusses key EJB architectural concepts so developers can have a deeper understanding of EJB. The newest version of this guide will be released soon.

Section 2. EJB-QL and your example

Introduction to EJB-QL

Back in the bad old days of EJBs, you had to write your own finder methods with JDBC. Finder methods are not new; they have been around since EJB 1.0. If you were lucky, your application vendor provided a means to map in queries easily, but that made your beans less portable. Thankfully, in EJB 2.0 you can define your finder methods with a standard query language, called *EJB-QL*, in your deployment descriptors.

EJB-QL looks a lot like SQL. Thus, if you are familiar with SQL, EJB-QL will be a breeze to learn.

EJB-QL is in some respects a subset of SQL, in that it does not support all of the SQL operations. In other respects, it is a superset of SQL, in that it adds support for traversing relationships between entities and working with collection-based CMR fields.

In writing these tutorials, I have tried to leave the parts of EJB-QL that are similar to SQL alone, assuming that you have a solid background in SQL. However, you should be able to follow along even if you are not a SQL guru. Your sample application will serve as a good testing ground on which you can learn EJB-QL concepts.

Application design: Entity design

If you recall, the examples in your series use a fictitious authentication subsystem for an online content management system that is designed to:

- Log users in to the system
- Authenticate users in certain roles
- Allow users to be organized into groups to allow group operations
- Store user information, such as address and contact data
- Manage (that is, add, edit, delete) roles, users, and groups

An overview of the system reveals that there are four distinct entities:

- User
- Group
- Role
- UserInfo

The above entities have the following three relationships:

- Users are associated with Roles (many-to-many)
- A User has UserInfo (one-to-one)
- A Group contains Users (one-to-many)

The above relationships yield four CMR fields, as follows:

- `User.roles`
- `User.userInfo`
- `Group.users`
- `User.group`

The CMR fields outnumber the relationships because the relationship between `User` and `Group` is bidirectional.

In order to test these entity beans, I created a session bean called `UserManagement` and a client called `Client`. The entity beans only have local interfaces, so the session bean acts as a facade to access the authentication system.

The relationships and entities were implemented in the first two tutorials in this series. For this tutorial, I reworked the example code to make it easier to port to different application servers and to make it map to real SQL tables. Thus, included in the example code for this tutorial is a build script that has the complete SQL DDL for all of the tables. I used EJBDoclets to create the beans and do the mappings to the SQL tables. This includes adding a primary key class and changing the finder methods to work with that class. The example as presented here is almost identical in structure to the example from the earlier tutorials. Note that in the previous tutorials you relied on the reference implementation to create the tables for you, which, although convenient, doesn't really resemble a real-world situation.

Note that I had to change the abstract schema name of the `Group` bean from `Group` to `UserGroup` because `Group` was a keyword in the EJB-QL of one of the application servers that I used to test the examples. (Yes, that's right, all of the examples were tested on real-world application servers that support EJB-QL.)

Parts 3 and 4 of this tutorial series build on this example, adding EJB-QL to build queries that define finder and select methods to perform a series of tasks. In this tutorial, you'll build methods that:

- Find all the groups in the system
- Find a group by a group name
- Find all users that are currently in a group
- Find all users that are currently *not* in a group

- Find users who are staff members

The left-hand side of the subsequent panels in this tutorial contains your sample code, while the right-hand side includes text that explains that code. Some of the code may be repeated on the right hand side as well, but I wanted you to have the code in its original context for comparison purposes.

Section 3. Using simple EJB-QL commands

Simple queries, many entities

In its simplest form, EJB-QL looks a lot like SQL. The EJB-QL is defined in the query element in the deployment descriptor. (The query element is a subelement of the entity element.) For example, to return all of the `Groups` in the system, you would use the following EJB-QL:

```
SELECT OBJECT(g) FROM UserGroup g
```

Instead of tables or views in the `FROM` clause, you use the schema name for the entity bean defined by the `abstract-schema-name` element.

To define a `findAll()` method, you would do the following:

1. Define a `findAll()` method in the home interface
2. Define the query element that contains the EJB-QL for the finder

A `findAll()` finder method that returned more than one entity would look like this:

```
public Collection findAll() throws javax.ejb.FinderException;
```

Notice that the finder methods that return more than one entity must return `java.util.Collection` or `java.util.Set`. (Finder methods that return `java.util.Set` are guaranteed to return entities only once in the `Set`.) Also notice that all finder methods must be declared to throw `javax.ejb.FinderException`.

The query element consists of the `query-method`, `result-type-mapping`, and `ejb-ql` elements, as follows:

```
<query>
  <query-method>
    <method-name>findAll</method-name>
    <method-params>
    </method-params>
  </query-method>
  <result-type-mapping>Local</result-type-mapping>
  <ejb-ql><![CDATA[
    SELECT OBJECT(g) FROM UserGroup g
  ]]></ejb-ql>
</query>
```

The `query-method` element contains the name of the method in the `method-name` element and a list of method parameters, which is a list of Java types for the method

parameters; the idea is to capture the method signature that would uniquely identify the method in the home interface.

The `findAll()` method does not have any parameters. The `query-method` element for `findAll()` is defined as follows:

```
<query>
  <query-method>
    <method-name>findAll</method-name>
    <method-params>
    </method-params>
  ...
</query>
```

The `result-type-mapping` element denotes whether the finder method returns local or remote interfaces of the entity bean. In this case, you are returning the local interface. The `result-type-mapping` is defined as follows:

```
<query>
...
  <result-type-mapping>Local</result-type-mapping>
...
</query>
```

The `ejb-ql` element is where the EJB-QL for the query is defined. The `ejb-ql` is defined as follows:

```
<query>
...
  <ejb-ql><![CDATA[
    SELECT OBJECT(g) FROM UserGroup g
  ]]></ejb-ql>
</query>
```

Notice that the `FROM` clause uses the `UserGroup`, which is the schema name for the `Group` entity bean. Also notice the use of the `OBJECT` keyword, which must be used to return entity beans. The above query returns all of the `Groups` in the system.

Simple query, single entity

You can pass arguments from finder method as parameters in the `WHERE` clause.

Let's say you wanted to create a query to find the group whose name was equal to a certain name that you're interested in. You'd use the following EJB-QL:

```
SELECT OBJECT(g) FROM UserGroup g WHERE g.name = ?1
```

The ?1 refers to the first argument of the method. If there were two arguments, then ?2 would refer to the second argument passed to the finder method, and so on.

The finder method in the home for the above examples is as follows:

```
public GroupLocal findByGroupName(String name)
    throws javax.ejb.FinderException;
```

Notice that this finder method also demonstrates a finder that expects a single entity as a result, as it returns `GroupLocal` and not `Collection`. In fact, if the query should return more than one result, a `FinderException` would be thrown.

Since this method takes a single argument, the argument must be declared in the deployment descriptor with the `method-param` element, as follows:

```
<query>
  <query-method>
    <method-name>findByGroupName</method-name>
    <method-params>
      <method-param>java.lang.String</method-param>
    </method-params>
  </query-method>
  ...
</query>
```

Note that the `name` used in the `WHERE` clause (as in `g.name`) is a CMP field.

```
SELECT OBJECT(g) FROM UserGroup g WHERE g.name = ?1
```

Thus, CMP fields can be used in EJB-QL queries.

IN query, all users in a group

Let's say you want to find all the users who are in a group. If you're thinking about doing this the SQL way, you're likely considering a correlated subquery, or maybe using the `join` keyword to join the `User` and `Group` tables. You should quit thinking in SQL; think in EJB-QL instead.

Remember all those relationships you set up in the last tutorial? They each have a corresponding CMR field. If you recall, the `Group` bean had a CMR field called `users`. Since a group has many users, this relationship field is a collection.

EJB-QL added special support for specifying CMR collections in the `FROM` clause: the

`IN` keyword. You can specify a CMR collection in the `FROM` clause. Thus, to find all users who are in groups, you would use the `IN` keyword in the `FROM` clause with the collection CMR field `users` of the `Group` bean, as follows:

```
SELECT DISTINCT OBJECT(user)
FROM UserGroup g, IN (g.users) user
```

The above code allows you to select all users who are in a group. Notice that this example uses the `IN` keyword in the `FROM` clause. On the next panel, you will use the `IN` keyword in the `WHERE` clause.

Also notice that the `DISTINCT` keyword is used; this keyword is used in the same way as it is in SQL -- in other words, no duplicate entity will be returned.

CMR paths and `IN` in the `WHERE` clause

Let's say you want to track users who were also staff members. More specifically, you want to track technical staff members as opposed to nontechnical staff members. You could write queries as follows:

```
SELECT DISTINCT OBJECT(user) FROM User user
WHERE user.userInfo.employee = TRUE
```

```
SELECT DISTINCT OBJECT(user)
FROM User user
WHERE user.userInfo.employee = TRUE AND
      user.group.name IN ('engineering', 'IT')
```

```
SELECT DISTINCT OBJECT(user)
FROM User user
WHERE user.userInfo.employee = TRUE AND
      user.group.name NOT IN ('engineering', 'IT')
```

Notice the first EJB-QL statement:

```
SELECT DISTINCT OBJECT(user) FROM User user
WHERE user.userInfo.employee = TRUE
```

The `WHERE` clause has a full EJB-QL path, that is, `user.userInfo.employee`. Recall from the last tutorial that the `User` bean has a CMR field of type `UserInfo` bean. The `UserInfo` bean has a boolean CMP field called `employee`. Thus, you are able to use the complete path using CMR fields. The CMR field path is used to access the CMP field for the comparison operator. EJB-QL uses the dot (`.`) to navigate from CMR field to CMR field to CMP field, which is similar to the Java syntax for navigating

members.

The second EJB-QL statement demonstrates the logical `AND` operator and the use of the `IN` operator in the `WHERE` clause:

```
SELECT DISTINCT OBJECT(user)
FROM User user
WHERE user.userInfo.employee = TRUE AND
      user.group.name IN ('engineering', 'IT')
```

The functionality of the `IN` operator in the `WHERE` clause is just like SQL. Basically, you determine if the `user.group.name` is equal to one of the values in the set (`'engineering', 'IT'`). Also note that `user.group.name` is another example of navigating over `CMR` fields to use `CMP` fields of related beans.

Lastly, let's use the `NOT` logical operator to get the nontechnical staff members:

```
SELECT DISTINCT OBJECT(user)
FROM User user
WHERE user.userInfo.employee = TRUE AND
      user.group.name NOT IN ('engineering', 'IT')
```

With the information on this panel, you should be able to use `CMR` paths in EJB-QL and use the `IN` operator in the `WHERE` clause.

Finding all members of a group using `MEMBER OF`

The `MEMBER OF` operator allows you to determine if an entity bean is a member of a specific collection-based relationship.

For example, the following query only returns the users who are a member of a group:

```
SELECT DISTINCT OBJECT(user)
FROM UserGroup g, User user
WHERE user MEMBER OF g.users
```

Thus, the `MEMBER OF` operator checks to see if the user is a member of a group of users:

```
user MEMBER OF g.users
```

You can use the logical operator `NOT` to check to see if a user is *not* in a group:

```
SELECT DISTINCT OBJECT(user)
FROM UserGroup g, User user
WHERE user NOT MEMBER OF g.users
```

The `MEMBER OF` operator without `NOT` gives the same response as using the `IN` operator in the `FROM` clause. Notice that the `MEMBER OF` operator is always in the `WHERE` clause.

Section 4. Conclusion

What you've learned, and looking ahead

In this tutorial I covered EJB-QL basics, including:

- Writing finder methods in the home interface
- Writing EJB-QL in deployment descriptors
- Finder methods
- The `IN` operator in the `FROM` clause
- The `IN` operator in the `WHERE` clause
- The `MEMBER OF` operator

With this knowledge, you were able to build finder methods that:

- Find all the groups in the system
- Find a group by a group name
- Find all users who are currently in a group
- Find all users who are currently *not* in a group

The fourth and last tutorial in this series covers advanced EJB-QL as follows:

- Comparison operators, including `LIKE`
- Select methods
- Logical operators
- The `BETWEEN` clause
- `IS NULL`

After the last tutorial, you will be able to build queries that:

- Find users who are staff members
- Find users who have user info
- Find users who do *not* have user info
- Find users who have salaries over a certain amount
- Find users who have salaries in a certain range
- Find users whose name is like a string pattern
- Select e-mails of users in a group

Also, the fourth tutorial will show you how to use select methods to drastically reduce implementation of helper methods (by up to a factor of 20), so stay tuned!

Resources

Web-based resources:

- Read the [first](#) and [second](#) tutorials in this series.
- Caucho.com offers a [good tutorial on EJB CMP/CMR and EJB-QL](#) .
- Read [the J2EE tutorial from Sun](#).
- I've updated [the Developer's Guide to Understanding EJB 2.0](#) to keep it current; check it out.
- "[Enterprise JavaBeans fundamentals](#)" is an excellent developerWorks tutorial to get you started with EJBs.
- I'm maintaining a site with [information on using this tutorial's examples on various application servers](#).

Books:

- [Mastering Enterprise JavaBeans \(2nd Edition\)](#), Ed Roman, Scott W. Ambler, Tyler Jewell, Floyd Marinescu (John Wiley & Sons, 2001). This is the EJB encyclopedia -- an invaluable reference.
- [EJB Design Patterns: Advanced Patterns, Processes, and Idioms](#), Floyd Marinescu (John Wiley & Sons, 2002). If you are working with EJBs and you call yourself an expert, you'd better know the material in this book!
- [Enterprise JavaBeans \(3rd Edition\)](#) , Richard Monson-Haefel (O'Reilly & Associates, 2001). Get this one too!
- [Java Tools for Extreme Programming](#), Richard Hightower and Nicholas Lesiecki (John Wiley & Sons, 2001). Covers building and deploying J2EE applications with EJBs.

Section 5. Feedback

Feedback

Please let me know whether this tutorial was helpful to you and how I could make it better. I'd also like to hear about other tutorial topics you'd like to see covered. Thanks!

Colophon

This tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial generator. The open source Toot-O-Matic tool is an XSLT stylesheet and several XSLT extension functions that convert an XML file into a number of HTML pages, a zip file, JPEG heading graphics, and two PDF files. Our ability to generate multiple text and binary formats from a single source file illustrates the power and flexibility of XML. (It also saves our production team a great deal of time and effort.)

You can get the source code for the Toot-O-Matic at www6.software.ibm.com/dl/devworks/dw-tootomatic-p. The tutorial [Building tutorials with the Toot-O-Matic](#) demonstrates how to use the Toot-O-Matic to create your own tutorials. developerWorks also hosts a forum devoted to the Toot-O-Matic; it's available at www-105.ibm.com/developerworks/xml_df.nsf/AllViewTemplate?OpenForm&RestrictToCategory=11. We'd love to know what you think about the tool.