

Introduction to EJB-CMP/CMR, Part 4

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 the example	5
3. Advanced EJB-QL finder commands	9
4. Using select methods with EJB-QL	12
5. Conclusion	16
6. Feedback	18

Section 1. About this tutorial

Should I take this tutorial?

This is the fourth part of a tutorial series on container-managed persistence and relationships in Enterprise JavaBeans 2.0. This part wraps up EJB Query Language (EJB-QL).

This tutorial assumes that you have completed the first, second, and third 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 on page 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, you will be able to create complex EJB-QL queries.

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

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

In the third and fourth tutorials in this series, you are using these relationships as raw material for EJB-QL queries to create select and finder methods. Here you will continue to 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.

The third tutorial 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

This tutorial covers advanced EJB-QL, including:


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

About the author



[Rick Hightower](#), CTO of [Trivera Technologies](#), has over a decade of experience as a software developer. Rick consults, trains, and mentors in addition to writing articles, books, and courses about enterprise Java development.

Rick's publications include the best-seller [Java Tools for eXtreme Programming](#) (John Wiley & Sons, 2001), which covers deploying and testing J2EE projects. He has also contributed to [Java Distributed Objects](#) (Sams, 1998), and has written several articles for [Java Developer's Journal](#).

<p>Java Tools for XP</p>		<p>Covers creating, testing, and deploying J2 applications using:</p> <ul style="list-style-type: none"> • JUnit • Cactus • JMeter • Ant
--	--	--

Expect to see his book on Jython from Addison Wesley in July 2002.

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

Rick recently updated the newest version of the *Enterprise JavaBeans Developer's Guide to the 2.0 Specification for Trivera Technologies*. The last version of this guide, which covered EJB 1.1, was distributed to over 60,000 developers. This free guide discusses key EJB architectural concepts offering developers a deeper understanding of EJB. Download your copy today at [Trivera Technologies](#).

Section 2. EJB-QL and the example

Simple EJB-QL: A review

This panel contains a brief summary of the concepts covered in the third tutorial in this series. If you're already comfortable with these simple EJB-QL concepts, skip ahead to the next panel.

With 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.

The EJB-QL for a finder method is defined in the query element in the deployment descriptor. For example:

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

Remember, 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;
```

Remember that you can pass arguments from finder method as parameters in the `WHERE` clause. For example:

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

The `?1` refers to the first argument of the method.

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

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

Remember that CMP fields can be used in EJB-QL queries, as demonstrated above by `g.name`.

To join the CMR field `users` as `user`:

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

To determine if a CMP field is in a set:

```
SELECT DISTINCT OBJECT(user)
FROM User user
WHERE user.group.name IN ('engineering','IT')
```

The `MEMBER OF` operator checks to see if an entity is a member of a CMR collection; thus, to check if the user is a member of a group of users:

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

This sums up what I covered in the previous installment of this series.

Application design: Entity design

If you recall, the examples in this 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, you 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 included 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.)

This tutorial builds on this example, adding EJB-QL to build queries that define to finder and select methods that do the following:

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

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. Advanced EJB-QL finder commands

Finding a user with roles using the `IS EMPTY` operator

The `IS EMPTY` operator is used to check to see if a collection CMR field is empty. For example, if you wanted to check to see which users had not been assigned roles, you could write the following query:

```
SELECT DISTINCT OBJECT(user)
FROM User user
WHERE user.roles IS EMPTY
```

Thus, you check to see if the user's roles are empty, that is, `WHERE user.roles IS EMPTY`. To find out which users *do* have roles, you could use a query that looks like this:

```
...
SELECT DISTINCT OBJECT(user)
FROM User user
WHERE NOT (user.roles IS EMPTY)
...
```

Note that you cannot use CMR collections that were defined in the `FROM` clause.

Finding a user with `UserInfo` using the `IS NULL` operator

The `IS NULL` operator is used to see if a CMP field or a noncollection-based CMR field is null. (Remember, collection-based CMR fields are never null: they can only be empty, not null.)

So, if you want to see if a user does *not* have user information, you could write a query as follows:

```
SELECT OBJECT(user)
FROM User user
WHERE user.userInfo IS NULL
```

Thus, `WHERE user.userInfo IS NULL` checks to see if the `userInfo` is present or absent -- that is, if it exists.

If you only want a user who *does* have user information, you could write a query that looks like this:

```
SELECT OBJECT(user)
FROM User user
WHERE user.userInfo IS NOT NULL
```

Finding a user with a certain salary using comparison operators and the `BETWEEN` operator

EJB-QL has the `BETWEEN` operator and logical operators that act much like their SQL equivalents. Though I won't go into too much detail explaining SQL basics, for the sake of completeness I will discuss these operators in this panel.

If you wanted to write a query to get all employees whose salaries were higher than a certain figure, you might do it like this:

```
SELECT OBJECT(user)
FROM User user
WHERE user.userInfo.salary > ?1
```

Notice that the above code compares the CMP field `salary` via the logical operator `>` (greater than) against the operand that will equate to the value of the first method parameter (`?1`). Also, notice that `userInfo` is a CMR field of `user`, so you can use the dot notation to navigate down to a CMP field that you want to use. This is quite an extension to SQL, and makes queries easier to write. In standard SQL, you would have to join the two tables before you could use `salary` in the comparison. This dot navigation in EJB-QL is nice syntactic sugar.

If you wanted to write a query to get all the employees in a certain salary range, you could do it like this:

```
SELECT OBJECT(user)
FROM User user
WHERE user.userInfo.salary
BETWEEN ?1 AND ?2
```

This query expects a finder method with two parameters for the two ends of the range declared with the `BETWEEN` operator. Thus, the method declaration of the finder method in the home interface would look like this:

```
public interface UserLocalHome extends javax.ejb.EJBLocalHome {
    ...
    public Collection findUsersBySalaryRange
        (Integer start, Integer stop)
        throws javax.ejb.FinderException;
    ...
}
```

Finding users by last names using the `LIKE` operator

I use the `LIKE` operator a lot. You might say that I really like it. (Yuck, that was bad!)

The EJB-QL `LIKE` operator works just like its SQL equivalent. You can use it to see if a CMP field is like a string pattern. Just as in SQL, the percent (`%`) and underscore (`_`) symbols are used as wild cards. The underscore is used as a wild card representing a single character, and the percent symbol is used to represent any number of characters.

A query that uses a `LIKE` operator would look something like this:

```
SELECT DISTINCT OBJECT(user)
FROM User user
WHERE user.userInfo.lastName LIKE ?1
```

If you check out the code in the code listing on the left-hand side of this panel, you will note that a method exposes this functionality from the `UserManagement` session bean so that you can access it from the client with the `getUserByLastNameLike()` method. Thus, to use `getUserByLastNameLike()`, you can specify the pattern string as a parameter.

For example, to find users with last names like "High", you would pass the following parameter value:

```
users = userMgmt.getUserByLastNameLike("High%");
```

Section 4. Using select methods with EJB-QL

Using `ejbSelect` to get user IDs from a group

So far you have only defined finder methods with EJB-QL. But you can define *select* methods as well.

While finder methods have been around since EJB 1.0, select methods are new to EJB 2.0. Finder methods are for external clients that want to find an entity bean. They are tied to the home interface and return homogeneous entity types based on their homes (singletons and collections of said entity). For instance, `EmployeeHome` finder methods always return `Employee` entities or collections of `Employee` entities.

Conversely, select methods are much more flexible. Select methods return heterogeneous types, and are not associated with a home. Clients of the entity beans cannot use select methods directly; select methods can only be used internally, within the entity bean. Instead of returning entity beans as finder methods do, select methods return CMP fields or collections of CMP fields.

Let's say you want to write a select method that returns a collection of user IDs from all the users in a group. Such a select method would thus be part of the `Group` entity bean. You would do the following:

1. Define an abstract select method in the entity bean
2. Use the abstract method somewhere
3. Define the query element that contains the EJB-QL for the finder

In step 1, you define the select method in the group entity bean as follows:

```
public abstract class GroupBean implements EntityBean{
    ...
    public abstract Collection ejbSelectUserIDs(String groupName)
                                   throws FinderException;
    ...
}
```

Notice that all select methods must start with `ejbSelect` plus the name of the select method. This should be nothing new to you. As a veteran EJB developer, you know that all special methods in the entity bean implementation class start with `ejb`, that is, `ejbCreate`, `ejbFind`, `ejbActivate`, and so on.

In step 2, you use the `ejbSelect` method you defined. Remember that this method cannot be used externally -- that is, it is not available in the local or remote interface of your bean. Thus, in order to use this `ejbSelect` method, you will need to create a

method that makes it accessible, as follows:

```
public abstract class GroupBean implements EntityBean{
    ...
    public String [] getUserIDs()throws javax.ejb.FinderException{
        Collection c = this.ejbSelectUserIDs(this.getName());
        return (String []) c.toArray(new String[c.size()]);
    }
    ...
}
```

Simple enough. Note that the `getUserIDs` method was also added to the local interface for the `Group` entity bean.

In step 3, you define the query element in the deployment descriptor, as follows:

```
...
    <query>
        <query-method>
            <method-name>ejbSelectUserIDs</method-name>
            <method-params>
                <method-param>java.lang.String</method-param>
            </method-params>
        </query-method>
        <ejb-ql><![CDATA[
            SELECT user.email
            FROM UserGroup AS g, IN (g.users) AS user
            WHERE g.name = ?1
        ]]></ejb-ql>
    </query>
...

```

There's nothing new here as far as the subelements of the query element go; the subelements above are identical to the subelements in the descriptor for finder methods. Vive la sameness! However, lets examine the EJB-QL:

```
SELECT user.email
FROM UserGroup AS g, IN (g.users) AS user
WHERE g.name = ?1
```

Notice that the `SELECT` clause selects a CMP field, instead of an entity bean as it would in all of the finder methods.

Why use select methods?

At this point, you may be staring at your screen (or, if you have a fast printer, a piece of paper) and wondering: "Big deal! So What?" I know that's what you are thinking. I mean, after all, you can already get access to all of the users' IDs (that is, their e-mail addresses) via the CMR relationship from `group` to `users`:

```
public String[] getUserIDsInGroup(String groupName){
    try{
        GroupLocal group = groupHome.findByGroupName(groupName);
        Collection users = group.getUsers();
        Iterator iterator = users.iterator();
        ArrayList userList = new ArrayList (50);

        while(iterator.hasNext()){
            UserLocal user = (UserLocal)iterator.next();
            String email = user.getEmail();
            userList.add(email);
        }
        return (String [])
            userList.toArray(new String[userList.size()]);
    }catch(FinderException e){
        throw new EJBException
            ("Unable to get user in role ", e);
    }
}
```

But now, with the power of `select` methods, you can rewrite the code as follows:

```
public class UserManagementBean implements SessionBean {
    ...
    public String[] getUserIDsInGroup(String groupName)
        throws FinderException{
        return groupHome.findByGroupName(groupName).getUserIDs();
    }
    ...
}
```

It *is* a big deal! Less code means less maintenance. You've replaced ten lines of code with one line of code. You have to love that!

Homework assignment

In this tutorial, I've covered almost all of EJB-QL, barring only the built-in functions `CONCAT`, `LENGTH`, `LOCATE`, `SUBSTRING`, `ABS`, and `SQRT`. You will notice that many of your favorite SQL functions -- including `COUNT`, `SUM`, and `AVERAGE`, among others --

are missing. The `ORDER BY` clause is also missing, but it will be added in EJB 2.1 and it is supported by most application servers anyway.

As an exercise to the reader, I offer the following method, which uses the CMR collection field; rewrite it with a select method that uses the `CONCAT` function. I expect that you can reduce the number of lines of code by a factor of 20:

```
public String[] getUsersInGroup(String groupName){
    try{
        GroupLocal group = groupHome.findByGroupName(groupName);
        Collection users = group.getUsers();
        Iterator iterator = users.iterator();
        ArrayList userList = new ArrayList(50);

        while(iterator.hasNext()){
            UserLocal user = (UserLocal)iterator.next();

            String firstName = user.getUserInfo().getFirstName();
            String lastName = user.getUserInfo().getLastName();
            String email = user.getEmail();

            StringBuffer sUser = new StringBuffer(80);
            sUser.append(firstName + ", ");
            sUser.append(lastName + ", ");
            sUser.append(email);

            userList.add(sUser.toString());
        }
        return (String [])
            userList.toArray(new String[userList.size()]);
    }catch(FinderException e){
        throw new EJBException
            ("Unable to get user in role ", e);
    }
}
```

[Email me](#) and let me know how this exercise went. It should be fun.

Section 5. Conclusion

What you've learned in this tutorial

This tutorial covered advanced EJB-QL, as follows:

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

With the above you were able to write finder and select methods to do the following:

- 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, this tutorial demonstrated how to use select method to drastically reduce implementation of helper methods (by as much as 20 times).

The end

If you have finished this four-part tutorial, you are well on your way to being a CMP/CMR and EJB-QL expert. In the first tutorial, you mastered CMP concepts and were introduced to EJB-QL to build a simple query. In the second, you built all three type of multiplicities for relationships and you explored bidirectional and unidirectional relationships. In the last two tutorial, you have learned how to use EJB-QL to build finder and select methods on top of the CMR relationships you defined in the second tutorial.

Thus, you covered CMP, CMR, and EJB-QL, and can feel confident in tackling your next EJB project. Congratulations! Thank you for sticking with it.

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 6. 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.