

Tutorial: Using EJB Query Language

This tutorial describes how the Financial Brokerage Service (FBS) sample application uses the *EJB Query Language* (EJB QL), a feature introduced in the EJB 2.0 specification. Using a syntax similar to SQL, EJB QL enables you to write queries based on Entity Beans without knowing anything about the underlying relational schema.

This tutorial assumes that you are familiar with the FBS and have installed and configured the required software as described in *About the Financial Brokerage Service*.

Contents

1. Concepts
2. Design
3. Required Software
4. Setup
5. Implementation
6. Resources
7. Feedback





Concepts

The Enterprise JavaBeans query language, EJB QL, was introduced in the EJB 2.0 specification. It is used to define queries for entity beans that operate with container-managed persistence (CMP). EJB QL enables you to specify the semantics of query methods in a portable way.

For an Enterprise application, EJB QL can express queries for two different styles of operations:

- To express the semantics and exact operation of the finder methods defined in the home interface. Finder methods allow the results of an EJB QL query to be used by the clients of the entity bean. Previously, all but the simplest finder methods required the use of vendor-specific functionality to express rich queries.
- To select entity objects or other values derived from an entity bean's abstract schema type through select methods defined on the entity bean class. Select methods allow you to use EJB QL to find objects or values related to the state of an entity bean without exposing the results to the client. These are queries that you make use of yourself inside of your bean's implementation methods.

This tutorial demonstrates the simple use of EJB QL for finder methods. It does not demonstrate the use of the select methods nor the traversal of entity relationships to form joins between objects.

To use EJB QL, you provide a string that contains a `SELECT` clause and a `FROM` clause, and optionally a `WHERE` clause. EJB QL is very similar to a subset of SQL, with a marginally different syntax. The query must be defined in an application's `ejb-jar.xml` file.

The following XML code maps a query to a `findByEmail` method in a bean's Home interface. The FBS sample application uses this query and method to find the user account that corresponds to a given email address. (For more information, see the *Implementation* section.)

```
<entity>
...
  <ejb-class>oracle.otnsamples.ibfbs.usermanagement.ejb.UserAccountBean</ejb-
class>
...
  <abstract-schema-name>UserAccount</abstract-schema-name>
...
  <query>
    <query-method>
      <method-name>findByEmail</method-name>
```

```

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

  <ejb-ql>
    [!CDATA[
      SELECT DISTINCT object(ua)
      FROM UserAccount ua
      WHERE ua.email = ?1]]
    </ejb-ql>
  </query>
  ...

```

In EJB QL the `<abstract-schema-name>` tag defines a value analogous to a table name in standard SQL. In the code above, the abstract schema name is `UserAccount`, and it maps to the bean named `UserAccountBean`.

EJB QL queries are implemented by the EJB container, which translates EJB QL statements to the appropriate database SQL statements when the application is deployed. Thus, the container is responsible for converting elements of the bean description to the appropriate database tables, primary keys, foreign keys, and column names. An EJB QL query is portable to any database supported by the container.

EJB QL uses the abstract persistence schemas of entity beans, including their relationships, for its data model. It defines operators and expressions based on this data model. EJB QL depends on navigation and selection based on the CMP-fields and CMR-fields of the related entity beans. You can navigate from an entity bean to other entity beans by using the names of CMR-fields in EJB QL queries.

Also, with a bean that uses CMP, the Home interface can include `findAll` and `findByPrimaryKey` methods. Each method performs a query under the hood, but the semantics and operation are managed entirely by the container. In fact, the EJB specification explicitly prevents you from trying to subvert its behavior by specifying the queries yourself.





Design

OTN developers wanted to give FBS users a way to retrieve forgotten user names and passwords. The FBS stores user account data in the database, so the developers defined a `findByEmail` method based on an EJB QL query. Given an email address, this method returns a `UserAccount` object, and the developers wrote code to parse the object and email the user name and/or password to the user.

This approach adds the desired functionality to the FBS with a minimum of effort on the developers' part. The biggest task is to define the query in `ejb-jar.xml`. At deployment time, the container (OC4J) generates the code to implement `findByEmail` (and the interfaces that expose it to clients) based on the EJB QL query.

To watch a viewlet that shows how the FBS uses EJB QL, browse to:

- http://otn.oracle.com/sample_code/tutorials/fbs/files/viewlets/ibfbsejbql_viewlet.html





Required Software



This tutorial presents several code examples. If you want to study them in context, download and install the FBS source code. If you also want to build and run the FBS, you will need the software listed in the Required Software section of *About the Financial Brokerage Service*.





Setup



Following is a general procedure for setting up your system to use EJB QL in OC4J to query a bean based on the EMP table. OTN developers have already performed the steps required to use EJB QL in the FBS sample application. To configure your system to build and run the FBS sample application, see the Setup section of *About the Financial Brokerage Service*.

1. Define a data source in your `%J2EE_HOME%/config/data-sources.xml` file to point to the schema you want to query. The FBS application uses a data source configured for OC4J.

```
<data-source
  class="com.evermind.sql.DriverManagerDataSource"
  name="OracleDS"
  location="jdbc/OracleCoreDS"
  xa-location="jdbc/xa/OracleXADS"
  ejb-location="jdbc/OracleDS"
  connection-driver="oracle.jdbc.driver.OracleDriver"
  username="scott"
  password="tiger"
  url="jdbc:oracle:thin:@localhost:1521:oracle"
  inactivity-timeout="30"
/>
```

2. Make sure the `orion-ejb-jar.xml` file points to the data source you have defined. For example, the following element points to the `jdbc/OracleDS` data source.

```
<entity-deployment name="Emp" data-source="jdbc/OracleDS" table="EMP">
```

3. If the required tables (e.g., EMP) already exist, edit the `orion-application.xml` file to turn off the `autocreate-tables` feature. For example:

```
<orion-application autocreate-tables="false" ... >
```





Implementation

This section describes how OTN developers implemented and used an EJB QL query in the FBS sample application to give users a way to retrieve forgotten user names and passwords.

First, the following code from `<ibfbs_home>/etc/ejb-jar.xml` defines the query and other key data.

```
<entity>
  <description>Entity Bean ( Container-managed Persistence )</description>
  <display-name>UserAccount</display-name>
  <ejb-name>UserAccount</ejb-name>
  <local-
home>oracle.otnsamples.ibfbs.usermanagement.ejb.UserAccountHomeLocal</local-
home>
  <local>oracle.otnsamples.ibfbs.usermanagement.ejb.UserAccountLocal</local>
  <ejb-class>oracle.otnsamples.ibfbs.usermanagement.ejb.UserAccountBean</ejb-
class>
  <persistence-type>Container</persistence-type>
  <abstract-schema-name>UserAccount</abstract-schema-name>
  <cmp-version>2.x</cmp-version>
  <prim-key-class>java.lang.Integer</prim-key-class>
  <primkey-field>accountNumber</primkey-field>
  <reentrant>False</reentrant>
  <cmp-field><field-name>accountNumber</field-name></cmp-field>
  <cmp-field><field-name>password</field-name></cmp-field>
  ...
  <cmp-field><field-name>email</field-name></cmp-field>
  ...
  <query>
    <description></description>
```

```

    <query-method>
      <method-name>findByEmail</method-name>
      <method-params>
        <method-param>java.lang.String</method-param>
      </method-params>
    </query-method>
    <ejb-ql>select distinct object(ua) from UserAccount ua where ua.email =
?1</ejb-ql>
  </query>
</entity>

```

The XML tags themselves are pretty informative, but let's dissect this query element piece by piece.

- The query element is associated with a specific entity bean definition. An entity bean can have zero or more query definitions.
- The method name matches one of the methods defined on the home interface. If there is more than one method of the same name, the method parameters will be used to identify which specific method is being described.
- The method params provide information about the parameters that accompany the method call. The number of parameters and their types are required information and are used to associate logical values to be used in the query and to help disambiguate overloaded method names. It's important that you get the order and type of the parameters correct since these will be mapped onto the parameters passed into the query.
- The EJB-QL element defines the actual query that will be used. The most striking difference from standard SQL is the requirements to specify the OBJECT keyword when returning a single entity type. The entities to be used within the query are specified through the use of the <abstract-schema-name> that is supplied as part of the definition of the entity bean. To support parameterized queries, you use logical placeholders of the form ?n, where n represents the location of the value in the method params list, starting from an index of 1. EJB QL supports standard SQL operations of between and like, but it does not yet support aggregate and sorting functions (these are due to be supported in EJB 2.1).

So what we have here is a fully expressed query for the `findByEmail` method using EJB QL. It will take one parameter, a string representing an email address, and will then query the persistent store for all `UserAccount` entities which have a matching email address. If there are two entities with the same results, then it will return only one of the matching entities in the collection.

The following code from

```

<ibfbs_home>/src/oracle/otnsamples/ibfbs/usermanagement/ejb/UserManagementSessionFacadeBean.java

```

calls the `findByEmail` method to get an object representing a user account, then parses the object for account number, password, first name, and last name.

```

public Hashtable getUserAccount(String email) throws RemoteException {
    Hashtable userHash = new Hashtable();
    try {
        // Find the User Account Local Entity Bean by passing the Email
        Collection uaColl = userAccountHomeLocal.findByEmail(email);
        Iterator uaIter = uaColl.iterator();
        if (uaIter.hasNext()) {
            UserAccountLocal userAccountLocal = (UserAccountLocal) uaIter.next();
            // Add the queried values to the Hashtable
            userHash.put("ACCOUNTNUMBER", userAccountLocal.getAccountNumber());
            userHash.put("PASSWORD", userAccountLocal.getPassword());
            userHash.put("FIRSTNAME", userAccountLocal.getFirstName());
            userHash.put("LASTNAME", userAccountLocal.getLastName());
        }
    } catch (FinderException ex) { // Trap Errors While Finding EJB Object
        System.out.println("Finder Exception in getUserAccount : "
            + ex.toString());
        throw new RemoteException("Finder Exception While Getting User Account = "
+
            ex.toString());
    }
    return userHash;
}

```





Resources

This tutorial is part of a series, *J2EE and EJB 2.0 Techniques*, based on the Financial Brokerage Service (FBS) sample application. Following are links to resources that can help you understand and apply the concepts and techniques presented in this tutorial.

Resource	URL
EJB Query Language	http://otn.oracle.com/tech/java/oc4j/doc_library/902/ejb/ql.htm
How-To: Use EJB QL for Finder Methods	http://otn.oracle.com/tech/java/oc4j/htdocs/how-to-ejb-ql.html
Viewlet: EJB QL in the FBS	http://otn.oracle.com/sample_code/tutorials/fbs/files/viewlets/ibfbsejbql_viewlet.html
EJB 2.0	http://java.sun.com/products/ejb/2.0.html
Oracle9iAS Containers for J2EE (OC4J)	http://otn.oracle.com/tech/java/oc4j/





Feedback

If you have questions or comments about this tutorial, you can:

- Post a message in the OTN Sample Code discussion forum. OTN developers and other experts monitor the forum.
- Send email to the author. <mailto:Robert.Hall@oracle.com>

If you have suggestions or ideas for future tutorials, please send email to:

- <mailto:Raghavan.Sarathy@oracle.com>

