

## Chapter 8: CMP Entity Beans

1) Container-Managed Persistence.....	8-2
2) Primary Key Class .....	8-3
3) Container-Managed Persistence Example.....	8-6
4) Mapping Container-Managed Fields .....	8-7
5) Deployment Settings for Product Bean.....	8-10
6) Settings for Custom Finders .....	8-14
7) EJB Query Language .....	8-15
8) Running the Product Application.....	8-17
9) Mapping to Multiple Database Tables.....	8-18

*The Art of Knowledge.*

Evaluation  
Copy

## Container-Managed Persistence

- With Container-managed persistence, the Entity Bean class does not contain any database access code. Instead, the container (WebLogic) generates the SQL statements.
- The container must know which instance variables are stored in the database. These variables are called **container-managed fields**.
- A container-managed field must be `public` and of one of the following types:
  - ▶ Java serializable class
  - ▶ Java primitive
  - ▶ Reference of a home interface
  - ▶ Reference of a remote interface
- You define one (or more) of the container-managed fields as the primary key field(s) of the Entity Bean.

## Primary Key Class

- The primary key class is specified in the `ejb-jar.xml` file.
- In most cases, the primary key class is `java.lang.String` or some other class provided in Sun's class library. You need to define your own primary key class if a primary key is composed of multiple fields – i.e., a compound key.
- A primary key class has the following requirements:
  - ▶ Must be a `public` class
  - ▶ All fields must be `public`
  - ▶ Field names must match the corresponding container-managed fields in the Entity Bean class (applies to container-managed persistence only)
  - ▶ Must have a `public` default constructor
  - ▶ Must implement the `hashCode()` and `equals()` methods
  - ▶ Must be serializable
- Note that the `<primkey-field>` element in the `ejb-jar.xml` file is not used if the key is a compound key.
- With Bean-managed persistence:
  - ▶ `ejbCreate()` returns an instance of the primary key class
  - ▶ `ejbFindByPrimaryKey()` verifies the existence of a database row for the given primary key

## Primary Key Class

- With Container-managed persistence:
  - ▶ `ejbCreate()` returns null
  - ▶ Do not have to write the code for `ejbFindByPrimaryKey()`
- A client can get the primary key of an Entity Bean by calling the `getPrimaryKey()` method of class `EJBObject`.
  - ▶ The Bean gets its own primary key by calling the `getPrimaryKey()` method of class `EntityContext`.
- Here is an example of a primary key class:

### `LineItemPK.java`

```
1. public class LineItemPK
2.     implements java.io.Serializable {
3.
4.     public String orderId;
5.     public String productId;
6.
7.     public int hashCode() {
8.         return (orderId + productId).hashCode();
9.     }
10.
11.    public boolean equals (Object obj) {
12.
13.        LineItemPK key2 = (LineItemPK) obj;
14.
15.        if (orderId.equals (key2.orderId) &&
16.            productId.equals (key2.productId)) {
17.
18.            return true;
19.        } else {
20.            return false;
21.        }
22.    }
23. }
```

## Implementing Entity Bean Methods with Container-Managed Persistence

- The home and remote interfaces are the same whether an Entity Bean uses Bean-managed or Container-managed persistence. However, the code in the Entity Bean class is different.
  
- `ejbCreate()`
  - ▶ Returns null, because the container ignores its return value
  - ▶ After the `ejbCreate()` method executes, the container inserts the container-managed fields into the database.
  
- `ejbLoad()` and `ejbStore()`
  - ▶ These methods are typically empty, but could contain code to uncompress (or compress) data read from (or written to) the database.
  
- Finder methods
  - ▶ The container, not the programmer, implements these methods.

Evaluation  
Copy

## Container-Managed Persistence Example

- The source files for this example are in the `examples\product` directory.
- We will use the data source that you previously configured in the Administration Console.
  - ▶ The JNDI name is: `jdbc/exampleTXDataSource`.
  - ▶ The JNDI name of the data source is referenced in the `weblogic-cmp-rdbms-jar.xml` file.
- Before running the example in this chapter, you must first create the `product` table. Use the `TableUtility` program you wrote in the JDBC chapter. (If the `product` table already exists, drop and re-create it.)

*The Art of Knowledge.*

Evaluation  
Copy

## Mapping Container-Managed Fields

- Mapping Container-managed fields involves three steps:
  - ▶ Specify the field as Container-managed in `ejb-jar.xml`
  - ▶ Specify the persistence type in `weblogic-ejb-jar.xml`
  - ▶ Specify the persistence mappings in `weblogic-cmp-rdbms-jar.xml`
- Fields are defined as Container-managed using the `<cmp-field>` element within an `<entity>` element in `ejb-jar.xml`:

```
<ejb-jar>
  <enterprise-bean>
    <entity>
      ...
      <cmp-field>
        <field-name>fieldName</field-name>
      </cmp-field>

    </entity>
  </enterprise-bean>
</ejb-jar>
```

- Persistence type is specified by the `<persistence>` element within an `<entity-descriptor>` element in `weblogic-ejb-jar.xml`.

## Mapping Container-Managed Fields

- Currently WebLogic supports one persistence type: `Weblogic_CMP_RDBMS`.
- Persistence type example:

```
<weblogic-ejb-jar>
  <weblogic-enterprise-bean>
    <ejb-name>BeanName</ejb-name>
    <entity-descriptor>

      <persistence>
        <persistence-use>
          <type-identifier>
            WebLogic_CMP_RDBMS
          </type-identifier>
          <type-version>6.0</type-version>
          <type-storage>
            META-INF/weblogic-cmp-rdbms-jar.xml
          </type-storage>
        </persistence-use>
      </persistence>

    </entity-descriptor>

  </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

## Mapping Container-Managed Fields

- The `weblogic-cmp-rdbms-jar.xml` file defines a data source and table name, and the mapping of EJB fields to database fields.

```
<weblogic-rdbms-bean>

  <ejb-name>BeanName</ejb-name>
  <data-source-name>DSName</data-source-name>
  <table-map>
    <table-name>TableName</table-name>
    <field-map>
      <cmp-field>fieldName</bean-field>
      <dbms-column>columnName</dbms-column>
    </field-map>
    ...
  </table-map>
</weblogic-rdbms-bean>
```

- ▶ Each `<cmp-field>` element name should match a `<cmp-field>` element in the `ejb-jar.xml` file.

## Deployment Settings for Product Bean

- In `ejb-jar.xml`:

```
1. <entity>
2.   <ejb-name>Product</ejb-name>
3.   <home>examples.product.ProductHome</home>
4.   <remote>examples.product.Product
5.     </remote>
6.   <ejb-class>examples.product.ProductBean
7.     </ejb-class>
8.   <persistence-type>Container
9.     </persistence-type>
10.  <prim-key-class>java.lang.String
11.    </prim-key-class>
12.  <reentrant>False</reentrant>
13.  <abstract-schema-name>ProductBean
14.    <abstract-schema-name>
15.    <cmp-version>2.x</cmp-version>
16.    <cmp-field>
17.      <field-name>productId</field-name>
18.    </cmp-field>
19.    <cmp-field>
20.      <field-name>description</field-name>
21.    </cmp-field>
22.    <cmp-field>
23.      <field-name>price</field-name>
24.    </cmp-field>
25.    <cmp-field>
26.      <field-name>unit</field-name>
27.    </cmp-field>
28.    <cmp-field>
29.      <field-name>qtyOnHand</field-name>
30.    </cmp-field>
31.    <primkey-field>productId</primkey-field>
32.    <query>
33.      Custom Finders - details shown later </query>
34.  </entity>
```

## Deployment Settings for Product Bean

- In `ejb-jar.xml`: (cont'd)

```
35. <assembly-descriptor>
36.     <container-transaction>
37.         <method>
38.             <ejb-name>Product</ejb-name>
39.             <method-name>*</method-name>
40.         </method>
41.         <trans-attribute>Required
42.         </trans-attribute>
43.     </container-transaction>
44. </assembly-descriptor>
```

- Note that the `ejb-jar.xml` file for this CMP entity bean is similar to that for the BMP entity bean in the previous chapter, except that:

- ▶ The persistence type is `Container` rather than `Bean`.
- ▶ A `<cmp-version>` element specifies the version of the EJB specification used by the bean.
- ▶ The `<resource-ref>` element is replaced with a series of `<cmp-field>` elements (one for each Container-managed field) and exactly one `<primkey-field>` element.
- ▶ `<abstract-schema-name>` specifies an identifier for the persistence schema of the entity bean. The schema is derived from the list of `cmp` fields and the corresponding access methods in the bean class. Abstract schema names can be used in EJB Query Language.

## Deployment Settings for Product Bean

- In `weblogic-ejb-jar.xml`:

```
1. <weblogic-ejb-jar>
2.   <weblogic-enterprise-bean>
3.     <ejb-name>Product</ejb-name>
4.     <entity-descriptor>
5.       <entity-cache>
6.         <max-beans-in-cache>1000
7.       </max-beans-in-cache>
8.     </entity-cache>
9.     <persistence>
10.      <persistence-use>
11.        <type-identifier>
12.          WebLogic_CMP_RDBMS
13.        </type-identifier>
14.        <type-version>6.0</type-version>
15.        <type-storage>
16.          META-INF/weblogic-cmp-rdbms-jar.xml
17.        </type-storage>
18.      </persistence-use>
19.    </persistence>
20.  </entity-descriptor>
21.  <jndi-name>
22.    ejb/examples/product/ProductHome
23.  </jndi-name>
24. </weblogic-enterprise-bean>
25. </weblogic-ejb-jar>
```

- Note that the `<reference-descriptor>` element used in the BMP bean example (previous chapter) has been replaced with a `<persistence>` element.

## Deployment Settings for Product Bean

- Our Container-managed persistence example requires a third deployment file, `weblogic-cmp-rdbms-jar.xml`. In the `<table-map>` element, each `<field-map>` element maps a field of the EJB class to a column in the database.
- In `weblogic-cmp-rdbms-jar.xml`:

```
1. <weblogic-rdbms-bean>
2.   <ejb-name>Product</ejb-name>
3.   <data-source-name>
4.     jdbc/exampleTXDataSource</data-source-name>
5.   <table-map>
6.     <table-name>product</table-name>
7.     <field-map>
8.       <cmp-field>productId</bean-field>
9.       <dbms-column>id</dbms-column>
10.    </field-map>
11.    <field-map>
12.      <cmp-field>description</bean-field>
13.      <dbms-column>descrip</dbms-column>
14.    </field-map>
15.    <field-map>
16.      <cmp-field>price</bean-field>
17.      <dbms-column>price</dbms-column>
18.    </field-map>
19.    <field-map>
20.      <cmp-field>unit</bean-field>
21.      <dbms-column>unit</dbms-column>
22.    </field-map>
23.    <field-map>
24.      <cmp-field>qtyOnHand</bean-field>
25.      <dbms-column>qty</dbms-column>
26.    </field-map>
27.  </table-map>
28. </weblogic-rdbms-bean>
```

## Settings for Custom Finders

- In `ejb-jar.xml`: (in the `<entity>` element)

```

1. <query>
2.   <query-method>
3.     <method-name>
4.       findByDescription</method-name>
5.     <method-params>
6.       <method-param>
7.         java.lang.String</method-param>
8.     </method-params>
9.   </query-method>
10. <ejb-ql>
11.   <![CDATA[SELECT OBJECT(p) FROM
12.     ProductBean AS p WHERE
13.     p.description = ?1]]>
14. </ejb-ql>
15. </query>
16. <query>
17.   <query-method>
18.     <method-name>
19.       findInRange</method-name>
20.     <method-params>
21.       <method-param>double</method-param>
22.       <method-param>double</method-param>
23.     </method-params>
24.   </query-method>
25. <ejb-ql>
26.   <![CDATA[SELECT OBJECT(p) FROM
27.     ProductBean AS p WHERE
28.     p.price >= ?1 AND p.price <= ?2]]>
29. </ejb-ql>
30. </query>

```

## EJB Query Language

- Enterprise JavaBeans Query Language (EJB QL) is a SQL-like language to describe CMP finder methods. EJB QL is intended to be a portable replacement for vendor-specific query languages such as WebLogic Query Language (WLQL).
- EJB QL is based on ANSI SQL92, but is enhanced to allow navigation over entity bean relationships. EJB QL queries are written using the names of the `cmp` fields of the entity beans. A special element in the deployment descriptor (`<abstract-schema-name>`) provides an identifier string that can be used to refer to a particular entity bean.
- EJB QL queries can also define select methods, which are similar to finder methods, but whose results are not available to the bean's client (for internal use within the bean class).
- An EJB QL query is a string that contains a `SELECT` clause and a `FROM` clause, and that may contain an optional `WHERE` clause.
- An EJB QL query may have parameters that correspond to the parameters of the finder or select method for which it is defined. Parameters are referred to as `?1`, `?2`, and so on.

## EJB Query Language - Examples

- Find all products:

```
SELECT OBJECT(p) FROM ProductBean AS p
```

- Find all products sold by the dozen:

```
SELECT OBJECT(p) FROM ProductBean AS p
WHERE p.unit = 'DZ'
```

- Find all products with prices within the range specified by two input parameters:

```
SELECT OBJECT(p) FROM ProductBean AS p
WHERE p.price >= ?1 AND p.price <= ?2
```

- Find all products with descriptions that begin with TR:

```
SELECT OBJECT(p) FROM ProductBean AS p
WHERE p.description LIKE 'TR'
```

- Find all products with quantity on hand not equal to a specified value:

```
SELECT OBJECT(p) FROM ProductBean AS p
WHERE NOT p.qtyOnHand = ?1
```

## Running the Product Application

- If necessary, set up the development environment by running the `setEnv.cmd` script in the course files home directory.
- Change to the `examples\product` directory.
- Run the build file by typing the command: `ant`
- Start WebLogic. (Be sure that the `product` table has been created before attempting to deploy the Product Bean.)
- Run the client with the command line:

```
java examples.product.ProductClient
```
- Note that if you run the Product client a second time, an exception is thrown because of a duplicate key. There are two ways to avoid the exception:
  - ▶ Create an empty `product` table using the `TableUtility` program. (Drop the existing table first.)
  - ▶ Run the Product client with “no” as a command line argument. This disables the portion of the client program that creates new rows in the `product` table.

## Mapping to Multiple Database Tables

- When a CMP bean is mapped to multiple tables, each table contains a row that is mapped to a particular bean instance. Thus, all tables will contain the same number of rows at any point in time.
- In addition, each table will contain all of the bean's primary key values. Consequently, each table must have the same number of primary key columns, and corresponding primary key columns in different tables must have the same type, though they may have different names.
- Each `<table-map>` element must specify a mapping from the primary key column(s) for a particular table to the primary key field(s) of the bean. Non-primary key fields may only be mapped to a single table.
- Example: A Department bean is mapped to 2 tables: DeptTable1, DeptTable2.
  - ▶ Department bean has:
    - primary key cmp-fields: deptId1, deptId2
    - non-primary key cmp-fields: location, budget, holidayPolicy
    -
  - ▶ DeptTable1 has columns: t1\_deptId1, t1\_deptId2, t1\_location, t1\_budget
  - 
  - ▶ DeptTable2 has columns: t2\_deptId1, t2\_deptId2, t2\_holidayPolicy

## Mapping to Multiple Database Tables

- In `weblogic-cmp-rdbms-jar.xml`:

```
31.
32. <table-map>
33.   <table-name>DeptTable1</table-name>
34.
35.   <field-map>
36.     <cmp-field>deptId1</cmp-field>
37.     <dbms-column>t1_deptId1</dbms-column>
38.   </field-map>
39.   <field-map>
40.     <cmp-field>deptId2</cmp-field>
41.     <dbms-column>t1_deptId2</dbms-column>
42.   </field-map>
43.   <field-map>
44.     <cmp-field>location</cmp-field>
45.     <dbms-column>t1_location</dbms-column>
46.   </field-map>
47.   <field-map>
48.     <cmp-field>budget</cmp-field>
49.     <dbms-column>t1_budget</dbms-column>
50.   </field-map>
51. </table-map>
52.
53. <table-map>
54.   <table-name>DeptTable2</table-name>
55.
56.   <field-map>
57.     <cmp-field>deptId1</cmp-field>
58.     <dbms-column>t2_deptId1</dbms-column>
59.   </field-map>
60.   <field-map>
61.     <cmp-field>deptId2</cmp-field>
62.     <dbms-column>t2_deptId2</dbms-column>
63.   </field-map>
64.   <field-map>
65.     <cmp-field>holidayPolicy</cmp-field>
66.     <dbms-column>t2_holidayPolicy</dbms-column>
67.   </field-map>
68. </table-map>
```

## Exercises

1. Implement a `findLowInventory()` method for the `Product` Bean.

- ▶ The method should take an integer parameter and return a collection of the primary keys for all products with quantity on hand less than the specified integer.
- ▶ Solution: `solutions\product1`

2. Create a `LineItem` Bean:

- ▶ Attributes should include order ID, product ID, and quantity ordered. Business methods in the remote interface should include:

```
public void    setQty (int qty);  
public int    getQty ();  
public String getOrderID ();  
public String getProductID ();
```

- ▶ Implement the `LineItem` Bean using Container-managed persistence. Primary key should consist of order ID + product ID.
- ▶ The home interface should include a `create()` method that takes three parameters: order ID, product ID, and quantity.
- ▶ Include a finder method that allows you to find all `LineItem`'s associated with a particular order ID.
- ▶ Write a client program and test.
- ▶ Solution: `solutions\lineitem`