Lab 6: Additional JPA Topics

In this lab, you will exercise various JPA features.

Objectives:

- To work with secondary database tables in a JPA application

Part 1: Secondary Tables

In this part, you will code an Employee entity that maps properties from columns in two tables.

Steps:

_1. Start by creating the database tables and populating them with sample data:

a. If the HSQLDB server is still running, kill it, then restart it with this command:

```
java -cp hsqldb.jar org.hsqldb.Server -database lab06db
```

Note: We are using the original HSQLDB version in this lab. Be sure to type *hsqldb.jar* in the above command, rather than *hsqldbv2.jar*.

b. Start the HSQLDB admin console as in earlier labs, ensuring that you use *hsqldb.jar* instead of *hsqldbv2.jar*. Then enter and execute:

```
CREATE TABLE EMPLOYEE (
   EMP_ID INTEGER NOT NULL IDENTITY,
   NAME VARCHAR(100),
   HIRE_DATE DATE,
   PRIMARY KEY(EMP_ID));

CREATE TABLE EMP_POS (
   POS_ID INTEGER NOT NULL IDENTITY,
   EMP_ID INTEGER,
   TITLE VARCHAR(100),
   PRIMARY KEY(POS_ID),
   FOREIGN KEY(EMP_ID) REFERENCES EMPLOYEE(EMP_ID));
```

From the admin console's menu, choose *View - Refresh Tree* - you should see two tables in the tree view.

Note how the EMP_POS table has a foreign key into the EMPLOYEE table.

c. Enter and execute the following to add some sample data:

Copyright © Descriptor Systems 2015. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

```
INSERT INTO EMPLOYEE (NAME,HIRE_DATE)
  VALUES ('Sam Smith', '2010-4-5');
INSERT INTO EMPLOYEE (NAME,HIRE_DATE)
  VALUES ('Sue Jones', '2009-3-8');

INSERT INTO EMP_POS (EMP_ID,TITLE)
  VALUES(1, 'Director of Sales');
INSERT INTO EMP_POS (EMP_ID,TITLE)
  VALUES(0, 'Chief Executive Officer');
```

d. To ensure that there's good data in the tables, press Clear, then in the entry box , enter **SELECT * FROM EMPLOYEE** and press the Execute button. You should see a couple of rows of Employee information.

e. You can leave the HSQLDB admin console window up so that you can explore the database whenever you like, but for now, go back to Eclipse.

_2. In the same fashion as in earlier labs, create a new Java project named **Lab06JPA** and add the JPA facet.

_3. Copy the *log4j.properties* file from an earlier project to the new project's *src* folder.

_4. Edit the persistence.xml file and update it:

a. Change the *name* to **lab06**.

b. Configure the database connection parameters and properties in the same fashion as in earlier labs. Use *validate* for the *hibernate.hbm2ddl.auto* property.

_5. Referring to the example in the course book as a guide, use the Eclipse create-JPA entity wizard to create an entity named **Employee** in the **part1** package with properties that map from the EMPLOYEE table's EMP_ID, NAME and HIRE_DATE columns and the EMP_POS table's TITLE column. You will need to use the @SecondaryTable and @Column annotations.

Note that the EMP_ID is auto-generated with IDENTITY and the HIRE_DATE is a simple date, so be sure to use the appropriate JPA annotations to configure them.

_6. In the *part1* package, create a class named **QueryEmployees** with a main() method that queries all employees and displays them to the console.

Part 2: Compound Primary Keys

In this part, you will create a database table named ACCOUNT that uses a compound key: ACCOUNT_NUM and USERID. You will populate the table, then write and map an entity. Then, you'll write a simple program to query accounts.

Steps:

_1. Start by creating the database table and populating it with sample data:

a. Ensure that the HSQLDB server is running from the last part of lab.

b. In the HSQLDB admin console, execute:

Copyright © Descriptor Systems 2015. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

```
CREATE TABLE ACCOUNT (
   ACCT_NUM INTEGER NOT NULL,
   USERID INTEGER NOT NULL,
   ACCT_NAME VARCHAR(100),
   CREATE_DATE DATE,
   PRIMARY KEY(ACCT_NUM, USERID));
```

From the admin console's menu, choose *View - Refresh Tree* - you should see the new table in the tree view.

c. Enter and execute the following to add some sample data:

```
INSERT INTO ACCOUNT VALUES (12, 3, 'Harry Wolfe', '2014-5-6');
INSERT INTO ACCOUNT VALUES (66, 7, 'Sue Simons', '2014-6-12');
INSERT INTO ACCOUNT VALUES (45, 3, 'John Fielder', '2014-3-18');
INSERT INTO ACCOUNT VALUES (467, 7, 'Dakota White', '2014-2-22');
```

d. To ensure that there's good data in the tables, press Clear, then in the entry box , enter **SELECT * FROM ACCOUNT** and press the Execute button. You should see the of rows of account information.

e. You can leave the HSQLDB admin console window up so that you can explore the database whenever you like, but for now, go back to Eclipse.

_2. Back in Eclipse, use the JPA create-entity wizard to create a new class named **Account** in the *part2* package that will act as your entity with properties that match the ACCOUNT table. Be sure to map both primary key columns to key properties.

_3. When you finish the wizard, Eclipse generates both the Account class for the entity and AccountPK for the ID class. Please open and examine AccountPK.java and note that it defines fields for the key columns, get/set methods and hashcode() and equals() methods.

_4. Open and examine Account.java and note how it specifies AccountPK as the IdClass and how both key properties have the @Id annotation. Which of the two strategies for compound primary keys did Eclipse generate for?

Compound key strategy: _____

Note that the Account class has a field of type java.util.Date. As you have done earlier, annotate the field to map to the database's DATE column type.

_5. Create a new class in the *part2* package named **QueryAccount** with a main() method. Complete the main() method:

a. Create an EntityManagerFactory and EntityManager in the usual fashion.

b. Create a query to find the Account with accountNumber=467 and userID=7 using the compound key. This query should return a single result.

c. Display the Account info to the console.

d. Close the entity manager and factory.

_6. Run QueryAccount.java as a Java Application.

Part 3: Optimistic Concurrency and Versioning

In this part, you will create a rudimentary application that lets you purchase items from inventory. You will run the application twice simultaneously to demonstrate a concurrency violation.

Copyright © Descriptor Systems 2015. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

Steps:

_1. Start by creating the database table and populating it with sample data:

a. Ensure that the HSQLDB server is running from the last part of lab.

b. In the HSQLDB admin console, execute:

```
CREATE TABLE INVENTORY (
   ITEM_NUM INTEGER NOT NULL IDENTITY,
   DESCRIPTION VARCHAR(100),
   QUANTITY_IN_STOCK INTEGER,
   VERSIONING INTEGER,
   PRIMARY KEY(ITEM_NUM));
```

From the admin console's menu, choose *View - Refresh Tree* - you should see the new table in the tree view.

Note the VERSIONING column - you will configure JPA to use this to detect optimistic concurrency violations.

c. Enter and execute the following to add some sample data:

```
INSERT INTO INVENTORY(DESCRIPTION,QUANTITY_IN_STOCK,VERSIONING)
  VALUES ('Bracket B58', 1, 0);
INSERT INTO INVENTORY(DESCRIPTION,QUANTITY_IN_STOCK,VERSIONING)
  VALUES ('Flange assembly B38', 1, 0);
INSERT INTO INVENTORY(DESCRIPTION,QUANTITY_IN_STOCK,VERSIONING)
  VALUES ('Fastener X65', 6, 0);
INSERT INTO INVENTORY(DESCRIPTION,QUANTITY_IN_STOCK,VERSIONING)
  VALUES ('Siphon hose', 4, 0);
```

d. To ensure that there's good data in the tables, press Clear, then in the entry box , enter **SELECT * FROM INVENTORY** and press the Execute button. You should see the of rows of inventory information. Please note the generated ITEM_ID values for a future part of the lab.

e. You can leave the HSQLDB admin console window up so that you can explore the database whenever you like, but for now, go back to Eclipse.

_2. Use the JPA create-entity wizard to create an entity named **Inventory** in the *part3* package. Be sure to map all of the columns to properties.

Edit the entity class and ensure that the primary key uses the IDENTITY strategy and the property mapped to the VERSIONING column has the @Version annotation.

_3. In the *part3* package, create a class named **InventoryManager**. This class doesn't need a main() method. Complete the class:

a. Write a method named **getInventory** that accepts an EntityManagerFactory reference. The method should query all Inventory entities and return the List.

Be sure to close the entity manager, but not the factory.

b. Write a method named **getItem** that accepts an inventory-item number and an EntityManagerFactory reference. The method should query that specific Inventory entity and return it.

Be sure to close the entity manager, but not the factory.

Copyright © Descriptor Systems 2015. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

c. Write a method named **purchaseItem** that accepts references to an Inventory entity and the EntityManagerFactory. The method should start a transaction, merge the Inventory entity into the persistence context, then commit the transaction.

   Be sure to close the entity manager, but not the factory.

_4. In the *part3* package, create a class named **InventoryApp** with a main() method. Start completing the main method:

a. Create a java.util.Scanner to accept user input and an EntityManagerFactory:

```
Scanner scanner = new Scanner(System.in);
EntityManagerFactory emf = Persistence
        .createEntityManagerFactory("lab06");
```

b. Create an instance of your InventoryManager class.

c. Write an infinite loop:

```
while(true)
{

}
```

d. Within the loop, first print a simple command menu:

```
System.out.println(
    "1) List inventory\n" +
    "2) Purchase item\n" +
    "3) Quit");
```

e. Write the skeleton of the keystroke processing logic:

Copyright © Descriptor Systems 2015. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

```
done: while(true)
{
    System.out.println(
        "1) List inventory\n" +
        "2) Purchase item\n" +
        "3) Quit");

    char key = scanner.nextLine().charAt(0);
    switch(key)
    {
        case '1':
            System.out.println("Current inventory");
            break;

        case '2':
            System.out.println("Purchasing item");
            break;

        case '3':
            break done;
    }
}
```

f. After the loop exits, close the scanner and the entity manager factory.

_5. In the case for the '1' menu key, call the InventoryManager class getInventory() method, then loop to display the list to the console.

Run the InventoryApp to test that this part of the code successfully displays the inventory list.

Be sure to quit the program and/or press the red-square icon on the Eclipse Console view's toolbar.

_6. Complete the case for the '2' menu key:

a. Prompt the user to enter an item number, then use the scanner to retrieve a string. Call Integer.parseInt() to convert the string to an integer.

b. Call the InventoryManager's *getItem* method to retrieve the Item.

c. If the quantity in stock is greater than zero, decrement it, then call the InventoryManager's *purchaseItem()* method. Display a message indicating the purchase was successful.

If there are no items in stock, notify the user.

_7. Run your program and test this new function. Try purchasing the "Fastener X65" item (it should have several in stock).

Be sure to quit the program and/or press the red-square icon on the Eclipse Console view's toolbar.

Note: To ensure that there's no programs still running, in the Eclipse Console toolbar, press the double-X icon followed by the red-square icon repeatedly until there are no consoles.

_8. Now test the basic function of the program:

a. Run the program again, and press '1' to list inventory. Note that there's only one 'flange assembly in stock'. Press '2' to start purchasing that item, but don't enter the item number yet. We will refer to this run as "User 1".

Copyright © Descriptor Systems 2015. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

b. Now run the program again. We will refer to this run as "User 2".

Press '1' to list inventory, noting there's still a single flange assembly in stock, since User 1 is in the middle of a purchase, but hasn't completed it.

Press '2' to start a purchase, then attempt to purchase a flange assembly. The purchase should be successful.

In the Eclipse Console view's toolbar, press the double-X icon to remove User 2's console -- Eclipse should redisplay User 1's console.

c. Back in Eclipse, as User 1, first note that as far as this user is concerned, there is still a flange in stock. Enter item number '1' to attempt to purchase a flange. Your logic to detect zero inventory should kick in, telling User 2 they are out of luck.

d. Quit User 2's run of the program.

Note: To ensure that there's no programs still running, in the Eclipse Console toolbar, press the double-X icon followed by the red-square icon repeatedly until there are no consoles.

_9. Now let's introduce a delay into the program to make it easier to test and detect a concurrency violation:

In the InventoryManager *purchaseItem()* method, immediately after the call to start a transaction, sleep for 30 seconds:

```
try
{
    TimeUnit.SECONDS.sleep(30);
}
catch (InterruptedException e)
{
    e.printStackTrace();
}
```

_10. Test a concurrency violation:

a. Run InventoryApp again referring to it as "User 1". List the inventory then start purchasing a siphon hose.

b. Run InventoryApp again, referring to it as "User 2". List inventory and also start purchasing a siphon hose.

Wait for User's 2 purchase to complete. As you are waiting, User 1's purchase will finish first, and Eclipse will display its console. User 1's purchase should be successful, since it started first. But keep waiting - User 2's purchase should fail with an OptimisticConcurrency violation and Eclipse should display that console. Examine the exception trace and ensure you understand what happened.

Hover your mouse over the Console view's toolbar buttons until you find the one for "Display selected console". Click that icon, and note you can switch between the two runs of the program. Verify the expected results, then press the Console toolbar's double-X icon to clear the consoles. Quit any running instance by pressing the red-square icon.

_11. Now let's make the program a little more user-friendly:

a. In the InventoryApp, write a *try* block around the call to purchaseItem() and the "successful purchase" output, then catch the OptimisticLockException. In the catch block, display a message telling the user that an error occurred and that they should retry the purchase.

b. Test the program again like you did in the last step, but this time, have both users attempt to purchase a "Bracket B58", which only has one item in stock. Ensure that your exception handling works as expected.

Copyright © Descriptor Systems 2015. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

Copyright © Descriptor Systems 2015. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.