CIS 318 - Homework 6

Deadline:
Due by 11:59 pm on Thursday, March 22

How to submit:
Submit your files for this homework using ~st10/318submit on nrs-projects, with a homework number of 6

Purpose:
To give you some Java and JDBC practice.

Important notes:
• Your solutions for these problems should avoid using a hard-coded password within your Java source code. Feel free to copy and make use of the getPassword method from the posted in-class Java JDBC examples.
• You are expected to use standard Java naming conventions (as discussed in class) in your Java source code.
• In your Java source code, you are expected to indent the contents of all { }’s by at least 3 spaces, and each { and } should be on its own line, even with the preceding line (as seen in posted class examples).
  – also, all Java classes and methods are expected to start with a comment that at least gives its name, and a purpose statement which explicitly describes either the purpose of the class or what the method expects and what it does and/or returns.
• Remember to also follow the style guidelines and course standards given or discussed previously for the other languages used in this homework.
• Make sure that you have executed the scripts create-bks.sql and pop-bks.sql, and that the bookstore tables are successfully created and populated.

The Problems:

Problem 1
Write a Java command-line application TotalOnHand.java using JDBC that queries the Oracle java account to find out the current total quantity-on-hand of all of the books, and prints that total to the screen within a descriptive message. (Note that it does not expect any command-line arguments, and it may happily ignore any given.)
Problem 2

Write ShowAuthorTitle.java, a Java command-line application using JDBC that queries the Oracle java account, printing to the screen a descriptive heading, then for each book printing the following:

- its author
- then a space and a dash and a space
- then the title (one author - title pair per line)

...in alphabetical order by author and, for books with the same author, in secondary order by title.

They do not have to be lined up nicely (we'll do that with <table>'s when we move on to servlets and JSP). (This also does not expect any command-line arguments and it may also happily ignore any given.)

Your resulting ShowAuthorTitle.java is now ready to submit.

Problem 3

Consider GetPubBooks.java from Homework 5, Problem 2. Right now, it:

- expects 1 or more command-line arguments, names of publishers;
- if it doesn't get at least one command-line argument, it should complain in a descriptive message to the screen and exit;
- otherwise, for each command-line argument, it prints a message to the screen, on its own line, noting that you would eventually list this publisher's books here (including the name of the publisher in the message).

Now modify GetPubBooks.java so that for EACH command-line argument, assumed to be the name of a publisher, it:

- prints a blank line followed by a nice heading noting that these are books published by <that publisher's name>,
- and then lists the titles published by that publisher, one per line, in alphabetical order by title, as queried from the Oracle java account.

Note the following additional requirements:

- What should you do if this is called with the name of an publisher NOT in the database? Then you are permitted to just print the blank line and nice heading noting that these are books by <that publisher's name>, with nothing following it.
- Since there can be multiple command-line arguments, you might be doing multiple very similar queries -- plus you are accepting input from a user, which can introduce the specter of SQL Injection. So, you are expected to use a PreparedStatement rather than a Statement for this application.
Your resulting GetPubBooks.java is now ready to submit.

**Problem 4**

Recall the UpdateLog.java example; you are going to write a variation on this that also affects the same table log_table on the Oracle java account that UpdateLog updates.

As a reminder, here are log_table's columns (and note that its primary key is BOTH columns):

```
SQL> describe log_table
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERNAME</td>
<td>NOT NULL</td>
<td>VARCHAR2(20)</td>
</tr>
<tr>
<td>TIME_LOGGED</td>
<td>NOT NULL</td>
<td>DATE</td>
</tr>
</tbody>
</table>

Write a Java command-line application AddToLog.java using JDBC that inserts a row into log_table for each of its command-line argument (that is, consider each command-line argument to be the username of a new row in log_table, using the current date for time_logged).

For example, if on March 8 you ran:

```
java AddToLog abc1 def2 ghi3
```

... then the following rows would be added to log_table:

```
abc1                     08-Mar-11
def2                     08-Mar-11
ghi3                     08-Mar-11
```

Note the following additional requirements:

- Since there can be multiple command-line arguments, you might be doing multiple very similar inserts -- plus you are again accepting input from a user, so you should still consider the possibility of SQL ingestion. So, you are expected to use a PreparedStatement rather than a Statement for this application, also.

- Note that it should simply complain in a descriptive message to the screen and exit if called with NO command-line arguments (it should NOT try to connect to the database in that case).

Your resulting AddToLog.java is now ready to submit.

**Problem 5**

We are all sharing that java Oracle account - it could get pretty littered while everyone works on Problem 4!

Write a Java command-line application RemoveFromLog.java that seeks to remove all rows from log_table with a username equal to one of its command-line argument. That is,

```
java RemoveFromLog abc1 def2 ghi3
```

...would remove all the rows added by the example call in Problem 4 (as well as any other rows with
the usernames abc1, def2, or ghi3).

Note the following additional requirements:

- The concept here is to do a separate delete for each username -- so, for the same reasons as for AddToLog, you are expected to use a PreparedStatement rather than a Statement for this application.

- It also should simply complain in a descriptive message to the screen and exit if called with NO command-line arguments (it should NOT try to connect to the database in that case).

(Be polite, please - while testing, try to only remove rows you have added... 8-) )

Your resulting RemoveFromLog.java is now ready to submit.

Problem 6

For some light meta-data playing, consider the posted example SpewTableColumns.java, which given a table name as a command-line argument, uses ResultSetMetaData to obtain and output the names of the columns in that table.

Look at the available methods for ResultSetMetaData under the Interfaces section of package java.sql at the Java 1.6 API (remember, there is link to it from the public course web page).

Consider the SQL*Plus describe command - it gives the names of the columns of the argument table name, along with whether they can be null (NOT NULL if they cannot be, blank if they can), and their type. But, since describe is a SQL*Plus command, you cannot call it using JDBC. But you can construct a Java version using JDBC's ResultSetMetaData...

You should find what you need amongst ResultSetMetaData's methods to create a Java command-line application JDescribe.java, which expects to take a table name as its command-line argument, and prints to the screen, for each column in that table:

- the name of each column,
- then a blank-dash-blank,
- NOT NULL and a blank-dash-blank if it cannot be null,
- and a string-depiction of that column's database-specific type name (with one column-name - if-not-null - type-name combo per line).

Note the following additional requirements:

- Have it complain to the screen and exit if anything other than exactly one command line argument is given.
- Have it complain differently to the screen and exit if there exists no table with that name.

It won't be as nicely formatted as the SQL*Plus describe command, but it will do for now. (That is, you don't have to format it any further from the format described above unless you want to...)

Your resulting JDescribe.java is now ready to submit.
**Problem 7**

Consider `SellBook.java` from Homework 5, Problem 3. Right now, it:

- expects exactly two command-line arguments, the ISBN to be sold and the quantity of that book to be sold.
- If it doesn't get exactly two command-line arguments, it should complain in a descriptive message to the screen and exit.
- If its second argument cannot be parsed as an integer, it should complain in a different descriptive message to the screen and exit.
- If that second argument/quantity is less than 0, it should complain in yet-another different descriptive message to the screen and exit.
- But if all is well, print a message to the screen noting that two appropriate command-line arguments were given, including those command-line arguments in your message.

Now modify `SellBook.java` so that, if its two command-line arguments pass the requirements above, it should:

- call the PL/SQL stored function `sell_book` to try to update the Oracle `java` account's database appropriately to sell that many of that book.
  - (Note that a version of `sell_book` and its needed auxiliary subroutines have been created within the `java` Oracle account; do NOT remove or replace any of these subroutines from the `java` Oracle account!)
- use the value that `sell_book` returns to print a descriptive message to the screen, indicating if the sale seems to have succeeded, and if not, why not (be as specific as you can be about what the problem is in this descriptive message printed to the screen).
- NOTE: call `pop_bks.sql` as needed to "restore" the poor `java` account's version of these tables as the class is working on this! And be aware that you are all testing using this one `java` account -- I hope it won't get too bizarre, but I cannot promise it won't.

Your resulting `SellBook.java` is now ready to submit.

**Problem 7 - Aside**

Something to think about (not to turn in with this homework, although it COULD be an exam or class exercise question at some point!):

- `sell_book` checks to make sure the number of books to be sold is >= 0. Why do you think I'm having you verify that this is the case within `SellBook`, and not call `sell_book` if so?
- You might have multiple different applications that happen to sell books as part of their purpose (a web site, an in-store check-out kiosk, even an in-store inventory replenishment system). What are some advantages of having all of the them call something such as PL/SQL stored function `sell_book`, instead of each implementing those business rules for selling a book themselves? What are some disadvantages of having all of them call the PL/SQL stored function `sell_book`?
Your resulting SellBook.java is now ready to submit.

Problem 8

Consider stored function max_on_hand from Homework 1, Problem 4 (and also available from the course Moodle site, under "Some solutions"). This is the version of max_on_hand created within the java Oracle account. Again, do NOT remove or replace this stored function from the java Oracle account!

Write a Java command-line application PrintMaxOnHand.java using JDBC that calls the PL/SQL function max_on_hand and prints to the screen the maximum quantity of books on hand for each publisher name given on its command line. (And yet again, complain in a descriptive message to the screen and exit if this is called with NO command-line arguments.)

For full credit, your solution must call PL/SQL function max_on_hand, preparing a single call that is reused (in the style of a PreparedStatement, even though you actually use CallableStatement for this -- it turns out that CallableStatement can use the same methods as PreparedStatement for this.)

Your resulting PrintMaxOnHand.java is now ready to submit.