1. Preliminaries

Under Resources→Lab4 on Piazza, there are some files that are discussed in this document. Two of the files are lab4_ create.sql script and lab4_data_loading.sql. The lab4_ create.sql script creates all tables within the schema Lab4. The schema is (almost) the same as the one we used for Lab3; there is no NewMemberships table, but there is a view (mentioned below). lab4_data_loading.sql will load data into those tables, just as a similar file did for Lab3. Alter your search path so that you can work with the tables without qualifying them with the schema name:

```
ALTER ROLE <username> SET SEARCH_PATH TO Lab4;
```

You must log out and log back in for this to take effect. To verify your search path, use:

```
SHOW SEARCH_PATH;
```

Note: It is important that you do not change the names of the tables. Otherwise, your application may not pass our tests, and you will not get any points for this assignment.

2. Instructions to compile and run JDBC code

Two important files under Resources→Lab4 are *RunNileBooksApplication.java* and *NileBooksApplication.java.* You should also download the file *postgresql-42.2.5.jar*, which contains a JDBC driver, from https://jdbc.postgresql.org/download/postgresql-42.2.5.jar

Place those 3 files into your working directory. That directory should be on your Unix PATH, so that you can execute files in it. Also, follow the instructions for "Setting up JDBC Driver, including CLASSPATH" that are at https://jdbc.postgresql.org/documentation/head/classpath.html

Modify *RunNileBooksApplication.java* with your own database credentials. Compile the Java code, and ensure it runs correctly. It will not do anything useful with the database yet, except for logging in and disconnecting, but it should execute without errors.

If you have changed your password for your database account with the "ALTER ROLE username WITH PASSWORD <new_password>;" command in the past, and you are using a confidential password (e.g. the same password as your Blue or Gold UCSC password, or your personal e-mail password), make sure that you do not include this password in the *RunNileBooksApplication.jav*a file that you submit to us, as that information will be unencrypted.

You can compile the *RunNileBooksApplication.java* program with the following command (where the ">" character represents the Unix prompt):

```
> javac RunNileBooksApplication.java
```

To run the compiled file, issue the following command:

```
> java RunNileBooksApplication
```

Note that if you do not modify the username and password to match those of your PostgreSQL account in your program, the execution will return an authentication error. (We will run your program as ourselves, not as you, so we don't need your password.)

If the program uses methods from the *NileBooksApplication* class and both programs are located in the same folder, any changes that you make to *NileBooksApplication.java* can also be compiled with a javac command similar to the one above.

You may get ClassNotFound exceptions if you attempt to run your programs locally and there is no JDBC driver on the classpath, or unsuitable driver errors if you already have a different version of JDBC locally that is incompatible with *cmps180-db.lt.ucsc.edu,* which is the class DB server. To avoid such complications, we advise that you use the provided *postgresql-42.2.5.jar* file, which contains a compatible JDBC library.

Note that Resources→Lab4 also contain a *RunStoresApplication.java* file from an old CMPS 180 assignment; it won't be used in this assignment, but it may help you understand it, as we explain in Section 6.

3. Goal

The fourth lab project puts the database you have created to practical use. You will implement part of an application front-end to the database. As good programming practice, all of your methods should catch erroneous parameters, such as a value for *numberReviewedBooks* that's not positive in *getAuthorsWithManyReviewedBooks*, and a value for *theCount* in *increasePublishersPrices* that's not positive, and print out appropriate error messages.

4. Description of methods in the NileBooksApplication class

NileBooksApplication.java contains a skeleton for the *NileBooksApplication* class, which has methods that interact with the database using JDBC.

The methods in the *NileBooksApplication* class are the following:

- *getAuthorsWithManyReviewedBooks:* This method has an integer argument called *numberReviewedBooks,* and returns the authorID for each author in Authors that has at least *numberReviewedBooks* different books that have at least one review. A value for *numberReviewedBooks* that's not greater than 0 is an error.

- *fixTotalOrdered:* For Lab3's createview.sql section, we said that a book in Books has a "bad book total" if that book's totalOrdered is not equal to the sum of the quantity values for the Orders of that book, and we defined a view, BadBookTotals, that had information about bad books. The *fixTotalOrdered* method has one integer argument, aPublisherIDtoFix, which is a publisherID. *fixTotalOrdered* should change the totalOrdered values for each "bad book" that was published by aPublisherIDtoFix, updating totalOrdered so that it is correct, i.e., so that it's equal to the sum of the quantity values for the Orders of that book. *fixTotalOrdered* should return the number of bad books whose totalOrdered values were "fixed".

  fixTotalOrdered is relatively easy to write if you use the BadBookTotals view, which has 3 attributes, bookID, totalOrdered and badQuantitySum. (Refer to Lab3 for more detail.) To help you, we included the BadBookTotals view in this assignment's `lab4_create.sql` script.

- *increasePublishersPrices*: This method has two integer parameters, thePublisherID and theCount, and invokes a stored function *increasePricesFunction* that you will need to implement and store in the database according to the description in Section 5. *increasePricesFunction* should have the same two parameters, thePublisherID and theCount. price is an attribute of the Books table. *increasePricesFunction* will increase the price for some tuples published by the publisher identified by thePublisherID; Section 5 explains which book prices should be increased, and how much they should be increased. The *increasePublishersPrices* method should return the same integer result as the *increasePricesFunction* stored function.

  The *increasePublishersPrices* method must only invoke the stored function *increasePricesFunction*, which does all of the assignment work; do not implement the *increasePublishersPrices* method using a bunch of SQL statements through JDBC. However, *increasePublishersPrices* should check to see whether theCount is greater than 0, and report an error if it's not.

Each method is annotated with comments in the NileBooksApplication.java file with a description indicating what it is supposed to do (repeating most of the descriptions above). Your task is to implement methods that match the descriptions. The default constructor is already implemented.

For JDBC use with PostgreSQL, the following links should be helpful. Note in particular, that you'll get an error unless the location of the JDBC driver is in your CLASSPATH.

Brief guide to using JDBC with PostgreSQL:
https://jdbc.postgresql.org/documentation/head/intro.html
Setting up JDBC Driver, including CLASSPATH:
https://jdbc.postgresql.org/documentation/head/classpath.htm
Information about queries and updates:
https://jdbc.postgresql.org/documentation/head/query.html
Guide for defining stored procedures/functions:
https://www.postgresql.org/docs/10/plpgsql.html

5. Stored Function

As Section 4 mentioned, you should write a stored function called *increasePricesFunction* that has two integer parameters, thePublisherID and theCount. *increasePricesFunction* will increase the prices of some of the books published by the publisher whose publisherID is thePublisherID. But you'll only increase prices for at most theCount books.

category is an attribute of Books. Non-Fiction books have category 'N'; Fiction books have category 'F'. *increasePricesFunction* should increase the price of Non-Fiction books by 1.50, and you'll increase the price of Fiction books by 2.50. (Leave the price unchanged for any book that has any other value for category.) However, you won't increase the price of all the Fiction and Non-Fiction books published by thePublisherID; at most theCount books should have price increases.

Which books in Books should have price increases? Answer: The most expensive books published by thePublisherID should have price increases. For example, if theCount is 3:

a) If thePublisherID published 3 or more books, then the prices of the most expensive 3 books that thePublisherID published should be increased.
b) If thePublisherID published only 2 books, then the prices of the most expensive 2 books that thePublisherID published should be increased.
c) If thePublisherID published no books, then there should be no price increases.

The *increasePricesFunction* stored function should return the number of books published by thePublisherID whose prices were increased. As cases b) and c) above demonstrate, that number might be less than theCount.

If theCount is 3, and there are multiple books that have the third-most expensive price, then you may increase the price of any one of them. But you should never increase the price of more than theCount books.

Write the code to create the stored function, and save it to a text file named *increasePricesFunction.pgsql.* To create the stored function *increasePricesFunction*, issue the psql command:

```
\i increasePricesFunction.pgsql
```

at the server prompt. If the creation goes through successfully, then the server should respond with the message "CREATE FUNCTION". You will need to call the stored function within the *increasePublishersPrices* method through JDBC, as described in the previous section. You should include the *increasePricesFunction.pgsql* source file in the zip file of your submission, along with your versions of the Java source files *NileBooksApplication.java* and *RunNileBooksApplication.java* that were described in Section 4.

As we noted above, a guide for defining stored functions with PostgreSQL can be found here on the PostgreSQL site. PostgreSQL stored functions have some syntactic differences from the PSM stored procedures/functions that were described in class, and PostgreSQL. For Lab4, you should write a stored function that has only IN parameters; that's legal in both PSM and PostgreSQL.

We'll give you some more hints on Piazza about writing PostgreSQL stored functions.

6. Testing

The file *RunStoresApplication.java* (this is not a typo) contains sample code on how to set up the database connection and call application methods **for a different database and for different methods**. *RunStoresApplication.java* is provided only for illustrative purposes, to give you an idea of how to invoke the methods that you want to test in this assignment. It is not part of your Lab4 assignment, so it should not be submitted as part of your solution.

RunNileBooksApplication.java is the program that you will need to modify in ways that are similar to the content of the *RunStoreApplication.java* program, but for this assignment, not for a Stores-related assignment. You should write tests to ensure that your methods work as expected. In particular, you should:

- Write one test of the *getAuthorsWithManyReviewedBooks* method with the *numberReviewedBooks* argument set to 3. Your code should print the result returned as output. Remember that your method should run correctly for any value of *numberReviewedBooks*, not just when it's 3.

  You should also print a line describing your output in the Java code of *RunNileBooksApplication*. The overall format should be as follows:

  ```
  /*
  * Output of getAuthorsWithManyReviewedBooks
  * when the parameter numberReviewedBooks is 3.
  output here
  */
  ```

- Write one test for the *fixTotalOrdered* method that fixes the totalOrdered values for the "bad books" published by the publisher whose publisherID is 94519. Print out the result of *fixTotalOrdered* (that is the number of "bad books" whose totalOrdered was updated) as follows:

  ```
  /*
  *Output of fixTotalOrdered when thePubisherIDtoFix is 94519
  output here
  */
  ```

- Write two tests for the *increasePublishersPrices* method. The first test should have theCount value 2 and thePublisherID value 98035. The second test should have theCount value 4 and thePublisherID value 98035 (yes, the same value for thePublisherID as in the first test). Your code should print the result (total number of books published by thePublisherID whose price was increased) returned by each test, with a description of what each test that you ran was, just as for the previous methods. Please be sure to run the tests in the specified order, running with theCount 2, and then with theCount 4. The order affects your results.

Important: You must run all of these method tests in order starting with the database provided by our create and load scripts. Some of these methods change the database, so using the database we've provided and executing methods in order might matter.

7. Submitting

1. Remember to add comments to your Java code so that the intent is clear.
2. Place the java programs NileBooksApplication.java and RunNileBooksApplication.java, and the stored procedure declaration code *increasePricesFunction.pgsql* in your working directory at unix.ucsc.edu.
3. Zip the files to a single file with name Lab4_XXXXXXX.zip where XXXXXXX is your 7-digit student ID, for example, if a student's ID is 1234567, then the file that this student submits for Lab4 should be named Lab4_1234567.zip. To create the zip file, you can use the Unix command:

   ```
   zip Lab4_1234567 NileBooksApplication.java RunNileBooksApplication.java increasePricesFunction.pgsql
   ```

4. Lab4 is due on Canvas by 11:59pm on Thursday, December 6, 2018 (not Sunday, December 2—you've been given additional time for Lab4). Late submissions will not be accepted, and there will be no make-up Lab assignments.