1. Preliminaries

Under Resources→Lab4 on Piazza, there are some files that are discussed in this document. Two of the files are create_lab4.sql script and lab4_data_loading.sql. The create_lab4.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 NewReads table. lab4_data_loading.sql (which will be posted soon) 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

Three important files under Resources→Lab4 are *RunChirpApplication.java*, *ChirpApplication.java* and *postgresql-42.1.4.jar*, which contains a JDBC driver. (There's also a RunStoresApplication.java file from an old version of this assignment; it won't be used in this assignment, but it may help you understand it, as we explain in Section 6.) Place those three files into your working directory; that directory should be on your Unix PATH, so that you can execute files in it. Also, follow these instructions for "Setting up JDBC Driver, including CLASSPATH"

Modify *RunChirpApplication.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 now use a confidentiality sensitive 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 *RunChirpApplication.java* file that you submit to us, as that information will be unencrypted.

You can compile the *RunChirpApplication.java* program with the following command:
> *javac RunChirpApplication.java*

To run the compiled file, issue the following command:
> *java RunChirpApplication*

Note that if you do not modify the username and password to match those of your PostgreSQL account in the program, the execution will return an authentication error.

If the program uses methods from the *ChirpApplication* class and both programs are located in the same folder, any changes that you make to *ChirpApplication.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 *cmps182-db.lt.ucsc.edu*, which is the class DB server. To avoid such complications, we advise that you use the provided *postgresql-42.1.4.jar* file, which contains a compatible JDBC library.

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.

4. Description of methods in the ChirpApplication class

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

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

- *getUsersByAddress:* This method takes as string argument called *theAddress.* It returns the list of userID values for ChirpUsers who have that address.

- *makeUsersInactive:* This method has no parameters. The ChirpUsers table has a Boolean attribute called active. We'll say that a user is active if the value of active is TRUE, and we'll say that a user is inactive if that attribute is FALSE. There may be some active users who haven't made any posts after December 31, 2016. Update the ChirpUsers rows for these users so that they become inactive. (Don't update rows of users that were already inactive.) *makeUsersInactive* should return the number of users who were made inactive by your update.

- *purgeBadUsers*: This method has one integer parameter, censorLimit. It invokes a stored function *purgeBadFunction* that you will need to implement and store in the database according to the description that appears in Section 5. *purgeBadFunction* should take have the same parameter, censorLimit. It will take actions to purge all information about certain users from the database; Section 5 explains which users will be purged, and what purging means. *purgeBadFunction* returns an integer value, and the *purgeBadUsers* method should return the same integer value.

  The method *purgeBadUsers* must only invoke the stored function *purgeBadFunction*, which does all of the assignment work; do not implement *purgeBadUsers* using a bunch of SQL statements through JDBC.

Each method is annotated with comments in the ChirpApplication.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
Setting up JDBC Driver, including CLASSPATH
Information about queries and updates
Guide for defining stored procedures/functions

5. Stored Function

As Section 4 mentioned, you should write a stored function called *purgeBadFunction*, that has one integer parameter, censorLimit. The ChirpPosts table has a Boolean attribute censored indicating whether or not a particular post has been censored. Some users may have made no more than censorLimit post that have been censored; these are good users. But some users may have made more than censorLimit posts that were censored; these are bad users. ChirpBase has decided to purge all information about bad users from its database, and *purgeBadFunction* will change multiple tables to do accomplish the purging.

Here's what you'll do in *purgeBadFunction* to purge a bad user:

- Delete the ChirpUsers tuple for the bad user.
- For any ChirpUsers tuple that has the bad user as spouseID, change spouseID to NULL.
- Delete all the ChirpPosts tuples that were made by the bad user.
- Delete all the ChirpFollowers tuples involving that bad user, whether as userID or followerID.

The *purgeBadFunction* stored function should return the number of bad users you purged, i.e., the number of ChirpUsers tuples that you deleted.

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

```
\i purgeBadFunction.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 *purgeBadUsers* method through JDBC, as described in the previous section. You should include the *purgeBadFunction.pgsql* source file in the zip file of your submission, along with your versions of the Java source files *ChirpApplication.java* and *RunChirpApplication.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 in a Piazza post. For Lab4, you should write a stored function that has only an IN parameter; that's legal in both PSM and PostgreSQL.

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**. It 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.

RunChirpApplication.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 two tests of the *getUsersByAddress* method, one with theAddress value '4 Privet Drive, Surrey', and the other with the theAddress value 'Tottenham Court Road, London'. Your code should print the userID values returned by *getUsersByAddress* for each value. Remember that your method should run correctly for any value of theAddress, not just for those values.

  Inside RunChirpApplication.java, you should print out the outputs you got from *getUsersByAddress* with the theAddress argument set to each of these values as follows:

  Output of getUsersByAddress when the parameter theAddress is '4 Privet Drive, Surrey':
  output here

  Output of getUsersByAddress when the parameter theAddress is 'Tottenham Court Road, London':
  output here

- Write one test for the *makeUsersInactive* method. Inside RunChirpApplication.java, you should print out the result returned by *makeInactiveUsers* (that is the number of users who were made inactive) as follows:

  Output of makeUsersInactive
  output here

- Write one test for the *purgeBadUsers* method. The test should run with censorLimit value 3. Your code should print the result (number of users that were purged) that was returned by *purgeBadUsers*.

7. Submitting

1. Remember to add comments to your Java code so that the intent is clear.
2. Place the java programs ChirpApplication.java and RunChirpApplication.java, and the stored procedure declaration code *purgeBadFunction.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 ChirpApplication.java RunChirpApplication.java purgeBadFunction.pgsql

4. Lab4 is due on Canvas by 11:59pm on Sunday, June 3, 2018. Late submissions will not be accepted, and there will be no make-up Lab assignments.