Full Stack Development/Software Workshop 2

Assignment 1

Marks available: 50% of the module mark

IMPORTANT

Use the **exact** names **and** capitalisation of all tables, classes, methods, etc. that are specified in this document. Do not add to or remove parameters from methods, omit access specifiers, include a `static` declaration, etc. for which the exact signature is already given below.

Do not add any further import statements apart from `HashSet` if you decide to use that.

You must add `SQLException` handling code inside your methods. This means you need a `try-catch` block, not simply a `throws` declaration on the method.

The point of this assignment is that you develop SQL queries that deliver the result you need. Don't simply get the 'raw data' from the database and then do your processing in Java. See the approach taken in the sample assignment.

Introduction

For this assignment you will creating some initial components of a full stack social media application. These components are the database and a Java program that connects to the database and executes queries on the database.

Jabber

Jabber is a fictitious social media platform. It is similar to other such platforms in that users can follow each other (or not) and post messages. Messages from user A will be seen by user B if user B follows user A. Users can 'like' messages posted by other users whether they follow them or not.

Setting up the database

For this assignment you will use the default 'postgres' database that is created when you install PostgreSQL. This database has no password. If you are using the Windows operating system, however, or if you set a password yourself, you will need to use the password you set in the Java file, as discussed below.

Firstly, import the table definitions and data using the PostgreSQL import facility that you have used previously in the module (and shown below). The two files you need can be downloaded from Canvas and are called *jabberdef.sql* and *jabberdata.sql*. Import jabberdef.sql first and then import jabberdata.sql.

The command to import these files is as follows. Remember, you need to state the path on your own computer where you put the files. You might also need to put the path in quote marks if your path has

space characters in it. Note that you must use a unix-style path even on Windows. For unix-style paths the separator is /.

```
\i /your_path/jabberdef.sql
\i /your_path/jabberdata.sql
```

You need to replace `your_path` with the path to the files on your computer.

Once you have imported the tables, you can list them by performing the following command in PostgreSQL:

```
\dt
```

Note that this will list all of the tables you may have created in this database[1]. Run SQL commands to view the contents of each of the newly-imported tables by doing 'SELECT * FROM *tablename*;' (replacing *tablename* with the name of the table you want to look at).

There now follows a description of each table in this database and its purpose. The relational schema are shown for your information only in the appendix.

There are four tables.

jabberuser represents a user of the jabber social media platform. Each user has a unique id number called *userid*. They also have a *username* which is the name that appears on the jabber platform, and an email address stored in the attribute *emailaddress*..

jab represents a post by a jabberuser on the jabber social media platform. A post is called a *jab*. A jab is a short message of up to 512 characters. Each jab has a unique id number stored in *jabid*, a *userid* which is the *userid* of the user who posted the jab, and the *jabtext* which is the actual text of the jab.

follows represents a follows relationship between jabberusers on the jabber social media platform. If user A follows user B this means that user A will see user B's posts on their jabber. Each row in **follows** has two *userid* numbers: *useridA* and *useridB*. Each row has the following meaning: *userida* follows *useridb*.

likes represents a likes relationship between jabberusers and jabs by other users on the jabber social media platform. A row is added to this table when one follower 'likes' a post by another user. The *userid* is the *userid* of the user who liked the post whilst the *jabid* is the *jabid* of the post that they liked.

Exercise: The Database Server

Only move to this stage when you have set up the database, as above. For this exercise you will create a Java Server that performs queries on the database. This server will simply connect to the database and then perform queries. The results of each query must be returned as shown in the list of required methods below.

Create a new Java project and add the file `JabberServer.java` that can be downloaded from Canvas. This file contains the code you will need to connect to the database, and method stubs for the required methods.

In your new project you will also need to make available the external jar file *postgresql-42.2.18.jar*. This files contains the drivers that you need to connect to a PostgreSQL database using Java. If you don't know how to do that, there are instructions for Eclipse and IntelliJ on Canvas. In essence, you need to

[1] You might have created some other tables in this database when doing your own work. This does not matter.

add *postgresql-42.2.18.jar* as an external jar file in your project. Do not try and 'import' this file. You also don't need to open it to view its contents.

In the `JabberServer` class you will see the following method stubs. You must add the code to create the functionality required for each method. For each method you must write the SQL that will give the required result and then retrieve the result of that query from the database. As it says above, you must do your processing in the SQL query, not in the Java code. Look at the sample assignment solution to see examples of this.

```
public ArrayList<String> getFollowerUserIDs(int userid)
```

This method returns a list of the *userid*s (as `String`s) of the jabberusers that follow the user with the userid `userid`. [**Marks: 5**]

```
public ArrayList<String> getFollowingUserIDs(int userid)
```

This method returns a list of the *userid*s of the jabberusers that the user with the userid `userid` is following. [**Marks: 5**]

```
public ArrayList<ArrayList<String>> getLikesOfUser(int userid)
```

This method returns a list of `ArrayList`s with the *username* and *jabtext* of all jabs liked by the user *userid*. The *username* is the *username* of the user who **posted** the jab. Thus, each of the `ArrayList`s returned will have two elements: {username, jabtext}.[**Marks: 6**]

```
public ArrayList<ArrayList<String>> getTimelineOfUser(int userid)
```

This method returns the 'timeline' of a user. A user's timeline is all of the jabs posted by users they follow. Each row of the result should be the username of the user who posted the jab and the jab text. Thus, each of the `ArrayList`s returned will have two elements: {username, jabtext}. [**Marks: 6**]

```
public ArrayList<ArrayList<String>> getMutualFollowUserIDs()
```

This method returns a list of `ArrayList`s, each containing a pair of 'mutual follows'. A mutual follow is when user A follows user B **AND** user B follows user A. The *userid*s are to be returned. There should be no duplicate follows relationships in the returned list. By definition, if there exists a mutual follows relationship between user A and user B then user A follows user B and user B follows user A. The returned list should contain only one instance of this relationship between user A and user B, i.e. it should return only {[userA, userB]} **not** {[userA, userB],[userB, userA]}. [**Marks: 7**]

```
public void addUser(String username, String emailadd)
```

This method adds a new user to the jabberuser table with username `username` and email address `emailadd`. You will need to generate a new *userid* number that does not already exist in the table. [**Marks: 4**]

```
public void addJab(String username, String jabtext)
```

This method adds a new jab to the jab table. The user is represented by `username` and the jab by `jabtext`. You must find the *userid* of the user with the username `username` to do this. You will need to generate a

new *jabid* number that does not already exist in the table. [**Marks: 4**]

`public void addFollower(int userida, int useridb)`

This method adds a new follows relationship: `userida` follows `useridb`. [**Marks: 4**]

`public void addLike(int userid, int jabid)`

This method adds a new like: user `userid` likes jab `jabid`. [**Marks: 4**]

`public ArrayList<String> getUsersWithMostFollowers()`

This method returns the *userid*s of the user(s) with the most followers. This might be more than one user if more than one user has an equally high number of followers. [**Marks: 5**]

Submission requirements

You must submit the following zipped as a **single** zip file. The zip file should have your name and student ID as its name, e.g. iankenny1234567.zip.

1. Submit the file `JabberServer.java` completed with your code. Do not change the file name, class name, method signatures or any of the other code that was already in the file. Make sure you have also not included any more `import` statements, other than `HashSet` if you decide to use that.

Add your name and student ID as a comment at the top of all submitted files.

If you choose to add other classes to your project then you must also include these in your submission. They must have the same package declaration as `JabberServer`.

Marking

If your submitted file does not compile you will receive a mark of zero.

If you submit a `.class` file instead of a `.java` file you will receive a mark of zero.

The file you submit by the deadline will be the one that is marked. It will not be possible to submit a file after that point, unless you have an extension.

You can see the number of marks for each method above. If your method produces the expected result then you can expect all of the marks for the method[2].

If one of your methods does not produce the correct result but does contain largely or entirely correct SQL, then you may be able to get **up to** 60% of the marks for that method. This will be at the discretion of the module team but in a transparent manner that will be announced. Work marked in this manner will take longer to process and return the results.

[2]But we reserve the right to give lower marks or zero marks in cases where the student gained the correct result through other means than writing an SQL query and submitting that the database, or has simply returned the correct values without getting them from the database and other such means, for example by doing their processing in Java instead of SQL. We also, of course, reserve the right to withhold marks in cases of plagiarism

Appendix: the relational schemas

jabberuser(
userid int,
username varchar(255),
emailaddress varchar(255))

Primary key: userid
Foreign key: ∅
Not null: username, emailaddress
Unique: ∅
Default: ∅
Check: ∅

jab(
userid int,
jabid int,
jabtext varchar(512))

Primary key: jabid
Foreign key: jabberuser(userid)
Not null: userid, jabtext
Unique: ∅
Default: ∅
Check: ∅

follows(
useridA int,
useridB int)

Primary key: {useridA, useridB}
Foreign key: jabberuser(userid)
Not null: ∅
Unique: ∅
Default: ∅
Check: ∅

likes(
userid int,
jabid int)

Primary key: {userid, jabid}
Foreign key: jabberuser(userid), jab(jabid)
Not null: ∅
Unique: ∅
Default: ∅
Check: ∅