CS 480: Database Systems
Homework 4: JDBC
Due Date: March 30th by 11:59 PM
No Late Submission Accepted

In this homework, you will write a Java program that interfaces with a database using JDBC (Java Database Connectivity). Your program should operate on the following database with two relations:

person(pid:integer, name:string, age:integer)
likes(pid:integer, mid:integer)

The *person* relation gives information about a person. The *likes* relation gives information about which person likes what other person. That is, if (100,200) is a tuple in *likes* then this shows that the person with *pid* 100 likes the person with *pid* 200. (Note this does not imply that 200 likes 100). Observe that, *pid* is not a key of the *likes* relation. The attributes *pid* and *mid* in *person* are both foreign keys referencing *pid* in *person*.

First, your program should create the tables *person* and *likes*.
After setting up the tables, your program has to read an input file called *transfile*. Each line in *transfile* is a transaction. There are six types of transactions. The first character of each line gives the type of the transaction. We call this as *transaction code*.

Transaction 1: Delete an Existing person
Example: 1 100

If the transaction code is 1, then you have to delete an existing person. The other field in this line is the pid of the person (i.e., the *pid*) to be deleted from *person* relation. In this case, **you also need to delete the corresponding tuples from the *likes* relation**.

Suppose you delete a person with pid=100 from the person relation, then you have to delete all tuples with pid=100 as well as all tuples with mid=100 from the likes relation.
You should output "error" if no tuple for the specified person exists in the *person* relation. If the person tuple is successfully deleted, then output "done".

Transaction 2: Insert a new person
Example: 2 50 Mary 30 100 200

In this case, the other fields, in the line are *pid, name, age* followed *by* zero or more *mids.* Each of them is separated by one or more spaces. Here *pid, age and mids* are integers while *name* is a string without spaces.
For this transaction, you have to insert the tuple (*pid, name, age*) into the *person* relation and insert zero or more tuples in the likes relation. Note that there can be zero or more mids. **The number of tuples inserted in the *likes* relation is as many as the number of distinct mids specified in the transaction**. If a given mid in the input is not present in the *person* table then an error message should be given and no tuple should be inserted in the *likes* relation for that mid.

You have to check that there are no duplicates in the person relation. If all the required tuples are inserted to the *person* and *likes* successfully, then output "done". If there is any problem then output "error".

Transaction 3: Output the average age of all persons
Example: 3

If the transaction code is 3 then you have to output the average age of all persons in the *person* relation. In this case, round the average age to the nearest integer before printing.

Transaction 4: Output names of all persons liked by a person
Example: 4 50

If the transaction code is 4, then you have to output names of all persons that are liked by the person with the given pid, and the persons liked by those persons, and so on. For example, if the pid is 100, then you have to output names of all persons that are liked by 100, and also all the persons that are liked by those persons and so on. That is, **you have to output all the names of persons that are directly or indirectly liked by 100**.

Transaction 5: Output the average age of persons liked, directly or indirectly, by the person with given pid.

Example: 5 50

If the transaction code is 5, then you have to output the average age of all persons that are directly or indirectly liked by the person with the given pid. If the average price is not an integer, round it to the closest integer.

Transaction 6

Example: 6

If the transaction code is 6, then you have to check if there is any person who **directly** likes two or more persons, and output the name of that person. If there is no such person, then output a message "there is no person that likes two or more persons".

When you read a line whose transaction code is 3, 4, 5 or 6 then you output the results of the transactions. If any problem occurs, output "error". After you have read all the transactions from *transfile*, your program should drop all the tables that were created.

Important Notes:

- **Don't forget to drop all tables you created at the end of your program.**
- **The name of the database your program connects, should be "homework4db" on the local host (username: root, password: root)**

Assumptions:
- The input file is always valid, and you don't need to check if it's correct.
- There are at most 100 persons in the database
- A person's name is a string of maximum 20 characters without any space.
- A person's age is a non-negative integer whose value is >= 0 and <= 100.

A sample input file is as follows:
*****transfile*****

```
2 100 Mary 30
2 101 Ford 35 100
2 102 Dell 40 101 100
2 103 Cathy 25 101
4 102
3
5 103
1 100
```

Using JDBC:
- You can download JDBC from https://dev.mysql.com/downloads/connector/j/. Select "**Platform Independent**" from the drop-down menu, download the zip format, extract it and copy the JAR file into your project folder.

"If you're using an IDE like Eclipse or Netbeans, then you can add it to the classpath by adding the JAR file as Library to the Build Path in project's properties."

For further instructions on using JDBC, please refer to this link (If you have MySQL installed, skip the first step).

Submission:
Test your program using your own *transfile*. Save your Java project, a readme file and *transfile* into a directory named by your netid. Compress the directory into a zip file, name the file as **YourNetID.zip** and upload it to **Gradescope**.