Databases: Exercise 6 (PL/pgSQL, JDBC)

November 27 2014
Due Date: December 15, midnight

Submission Instructions: This exercise consists of programming in PL/pgSQL and Java (using JDBC), and should be submitted electronically, via the submission link on the course homepage. Please run your code one more time just before submission to check that it works! Submit your exercise as a zip file containing all the necessary files.

When you buy a new electronic device you sometimes face the question of which firm's devices will last longer. Therefore, you decide to create an app that will collect information from people who are using electronic devices and enable better statistics than asking your friends what they have.
You are creating this app with a friend who is an experienced app designer, since you took the DB course you decide to take care of the DB part.
You have defined the database backend of the app, containing the following tables: (they are already there you are not supposed to create them in your program):

- Devices(did: integer, fid: integer, dtype: integer) – stores information about devices, including did (the device ID), fid (the firm ID) and dtype (1= Smartphone, 2= TV …).
- Ownerships(ownerid: integer, userid: integer, did: integer, datemonth: integer, dateyear: integer, ownershipDropped: integer) – stores ownership information, where ownerid is the ownership relation identifier, userid is the identifier of the user, did is the device ID and month and year are the date of purchasing the item, ownershipDropped is 0 if the user owns the device and 1 if he owned the device, but no longer does. ownershipDropped is always initialized with the value 0.
- DeviceTracking(tid: integer , ownerid: integer, datemonth: integer, dateyear: integer, sid: integer) – stores tid (the tracking identifier), ownerid (the ownership identifier), month + year of update in device tracking information, and sid (the status identifier) which is a number between 1 and 4, indicating:
    1. The ownership was dropped although the device was working (e.g., the user got a better device, the device was stolen…).
    2. The ownership was dropped because the device stopped working or was no longer good enough.
    3. Needed repair (and the customer is not happy that the repair was needed).
    4. Started misbehaving, i.e., the device works but not as well as it used to work.
- MostPopular(dtype: integer, did: integer, popularity: integer) – stores the most popular devices of each type with popularity ≥ 1. Popularity is defined in Q2.

You can assume that there are foreign key constraints defined on the tables, where this is natural, e.g., every "did" in Ownerships appears in Devices, etc. Also, none of the fields in any of the tables can have

null values (also this is ensured by the table definitions). You can assume that the user will write insert and delete commands to try to insert and delete rows, but will never issue an update command.

You are asked to write the following:

1. Write a PL/pgSQL function called **getMonthPassed (trackingid)**, which accepts a tracking ID and returns the number of months that passed between the tracking and when the ownership began (this will always be a non-negative integer) in a file named **q1.sql**

2. The popularity of a device is the number of devices owned for which the owner is happy with the device (it needed no repair and it never misbehaved), Write a PL/pgSQL function called **getPopularity (deviceid)**, which accepts a device ID and return its popularity (don't use the table MostPopular for this question) in a file named **q2.sql**

3. The popularity of a firm is the sum of popularity across all its devices, Write a PL/pgSQL function called **getFirmPopularity (firmid)**, which accepts firm ID and return its popularity (don't use the table MostPopular for this question) in a file named **q3.sql**

4. In addition to the above functions, you should write triggers that ensure certain consistency conditions on the tables of the database. In particular, your triggers should ensure the following:
    - Rows cannot be deleted from DeviceTracking, Ownerships, Devices: if a user tries to delete, nothing should happen (it should be rejected). Write the code ensuring correctness in a file named **q4A.sql**
    - When there is "Ownerships dropped" in DeviceTracking (i.e., an insert to the table with status 1 or 2), the table Ownerships should be made consistent. Write the code ensuring correctness in a file named **q4B.sql**
    - DeviceTracking rows for a particular ownership identifier must have dates that are not before the date in which the ownership began. Any attempt to add a row that is not consistent with this requirement should be rejected. Write the code ensuring correctness in **q4C.sql**
    - MostPopular should always contain, for each type, the most popular device(s) for that type. Only retain information about popularity in this table for devices that have popularity ≥ 1. Note that in case of a tie, i.e., several devices have the same greatest popularity score for a specific type, all of the most popular devices will appear in MostPopular. You should ensure that this holds regardless of any changes to any of the tables in the database. Write the code that will ensure correctness in a file named **q4D.sql**

5. Submit a file called **dropFunctions.sql** that deletes all functions and triggers created in Q1-4.

Finally, you would like your app to output a list of recommended devices for the user, Your friend the experienced app designer says that if you program the function in JDBC, He will convert it to Android Java and take care of the changes needed to be made in order for it to work via the internet.

Write a Java program, which connects to the database using JDBC. You should write a class **Recommend** (in a file called **Recommend.java**). This class must have at least the following methods:

public void init(String username)

public void close()

public int[] getRecommendation (int dtype, int userid, int k)

The function init of this class should receive a user name (which will be used in the connection string to connect to the database). The function init will be called immediately after an instance of **Recommend** is created. It is init's responsibility to make all the necessary initializations including the DB connection. The function close should close all connections. It will be called at the end of the execution.

Finally, the function **getRecommendation**, will return a list of device IDs of the type dtype (that the user doesn't own) where there exists a user that is k-connected to the user userid and has this device.
A user is connected to another user if they have a common device that they currently own,
User u and user u' are k-connected if there is a sequence $u_0, \dots, u_j$ of users such that $u_0 = u$, $u_j = u'$ $j \leq k$ and for all $i < j$, u_i and u_{i+1} are connected.
The list should be sorted by did.
Example: let's write a device as did\ dtype, assuming we have:

userid	Devices owns
1	121\1, 355\3
2	121\1 , 242\2, 324\3 , 355\3
3	242\2 , 333\3
4	333\3, 311\3

getRecommendation(dtype =3,userid= 1, k=2) should return: (324, 333).
If user 3 had owned 355\3 then the list would have been: (311, 324, 333).

Hint: You don't have to use recursion in SQL, don't worry too much about running times.

Note:

1. Carefully review the lectures about PL/pgSQL and JDBC before starting.
2. You can assume legal input for all tables: correct data types and no null values.
3. We will take off points for style only in extreme cases. No documentation is required unless you do something very special.
4. Pay attention to correct naming of functions, input variables, and files. Pay attention to correct usage of functions, valid zip file, and that everything works properly.
5. We do not have efficiency requirements. This holds for all parts of the exercise. However, if your program does not complete even after a reasonable amount of time, your program may fail some of our automatic tests.
6. Please post any exercise related questions to the forum so that they can benefit the group.

While testing your program make sure to close all connections to the database at the end of your program! Failing to do so will cause all available connections to be expended, which means that nobody will be able to connect to the database.