University of Toronto
CSC**343S, Winter 2011**

Assignment 2

Due: Thursday 10 March, at 10:00 pm sharp!

IMPORTANT: The submission instructions for this assignment may change slightly once we complete implementation of our autotesting framework. Read the discussion board for any updates.

The two parts of this assignment can be done in either order.

Part 1: Writing SQL queries

For this first part of the assignment, you are given a database schema; your task is to populate the database with data, and write queries to be run against the database.

Domain

You will be writing queries on a database about ultrasound medical tests in an obstetrics clinic. When writing your queries, the only things you can count on are the things that the schema enforces. Do not make any additional assumptions about the data, even if they make sense in the domain. Your queries should work for any database that is valid with respect to the schema.

The queries

The output of each query must be stored in a separate table. The tables must be named query1 through query5, to correspond to the five questions below. For your convenience, I will provide a schema that defines these tables. To put your query result into the appropriate table, use INSERT INTO as follows: `INSERT INTO <queryX> (<your query>)`; We will be autotesting your code, so it is crucial that you follow these instructions. If you don't, your code will be considered incorrect.

You are encouraged to use virtual views to define intermediate results, and it's a good idea to add commentary explaining what you're doing. The comment symbol is two hyphens.

1. For every doctor, report the doctor's billing number and full name; the clinic ID and address where they've sent the most patients for ultrasound exams; and the number of patients they've sent to that clinic for ultrasound exams.
2. Researchers often study how pregnancies differ according to the mother's age. For each year when any births are recorded in the database, report the number of mothers who gave birth at age less than 30, age 30-35 and age over 35, as well as the average birth weight of babies born to mothers of age age less than 30, age 30-35 and age over 35.
3. Suppose we are studying technician workloads. Every technician has a maximum number of exams they've ever done in one day. Find the technician whose maximum is the lowest. Report that technician's ID, full name, the maximum number of exams they've ever done in one day, the minimum number of exams they've ever done in one day, and the average number of exams they have done in a day (averaging only across days when they've actually done an exam).
4. The database may not have information about the outcome of every pregnancy. Find every pregnancy for which we don't have have a birth recorded within 10 months of the pregnancy's start date. Report the patient ID and full name, and the start date of the pregnancy.

5. A given type of measurement, such as head circumference, is often taken several times during a single exam because it may not be very accurate. The schema allows for this by including a repetition number attribute in the Result relation. Ideally, the difference between repetitions is very small. For each clinic, for every technician who performed any exams in that clinic, and for every type of measurement the technician took in those exams, find the maximum difference between repetitions of that measurement within one exam. Report the lab, technician, measurement and the maximum difference. Include only those tuples where the maximum difference is greater than the average difference for that type of measurement across all labs and technicians.

Part 2: Building a database to answer some questions

For this part of the assignment, you are provided with real data, scraped from Twitter, and some questions to answer. In order to answer the questions, you will first turn the raw data into a database, by (a) defining a schema that can represent the data and (b) converting the raw data into INSERT INTO statements in order to get it into the database. This is a realistic scenario; we often have data that we want to explore and it is rarely already conveniently in a database. The data is also often much less "clean" that what you'll be using here, but I won't be giving you a messy dataset just for the experience of cleaning it up.

Your schema

Define a schema that is capable of holding all the data in the dataset. It needn't be capable of doing more than that. For instance, you may set a limit on the length of a bio. Some future dataset may exceed that limit. Don't worry about that.

When designing the schema, refer back to the basic ideas we discussed in our lectures on the relational model. We will later learn some more rigorous concepts about schema design that will prepare you for designing more complex schemas.

The data

The dataset contains information about real user profiles that we scraped recently from Twitter. It consists of a series of user profiles, one after the other. Each user profile has the following elements, in this order:

- A line containing a non-blank, non-empty username. You may assume that usernames are unique, that is, a single username will not occur more than once in the file.
- A line for the user's actual name. If they did not provide a name, this line will be blank.
- A line for the user's location, or a blank line if they did not provide one.
- A line for the URL of a website, or a blank line if they did not provide one.
- One or more lines for the user's bio, followed by a line with nothing but the keyword ENDBIO on it. This marks the end of the bio, and is not considered part of it. (You may assume that no bio has the string ENDBIO within it.) If the user did not provide a bio, the ENDBIO line will come immediately after the website line, with no blank line in between.

- One or more lines each containing the username of someone that this user follows, terminated by a line with the keyword END. (You may assume that no one has the username END.) A user cannot be on his or her own follows list.

Notice that the keywords act as separators in this file, and that all their letters are capitalized.

Populating the database

Write a program, in any language you choose, that reads the Twitter data and produces from it a series of INSERT INTO statements that will put the data into your database. **Hint:** Watch out for the ' symbol in the data. Remember that you can include it inside an attribute value in SQL, but only if it is preceded by another ' symbol.

Queries

1. **Non-reciprocals**: Report all pairs of userIDs A and B such that A follows B but B does not follow A.

2. **People in common:** Suppose you are looking for links between users gvwilson and dianelynnhorton. Report the names and user IDs of all people that both of them follow or that are followed by both both of them. Do not include duplicates.

3. **Popularity:** Let the "popularity" of user a be the difference between the number of followers that a has (users who follow a) and the number of users that a follows. Note that popularity can be negative. Report the userID, name, location and popularity of all users, in non-ascending order by popularity.

4. **Three degrees of separation:** For a given user a, the users in the set $\{x|a \text{ follows } x\}$ are considered to be one "degree of separation" from a. If you add in all users who are one degree of separation from these, you have the users who are *two* degrees of separation from a. And so on. This definition does not exclude cycles. If, for instance, p follows q, q follows s, and s follows p, then p is three degrees of separation from him or herself.

   For each user, report their user ID and the number of different users you can get to in three degrees of separation from them. Put the results in non-ascending order by this number.

Capturing your query results

Because you are defining your own schema, we can't autotest your code. Instead, you will hand in the results of your queries.

As I did for Part 1, define a table to hold your results for each query. Name them query1, query2, and so on. Insert your query results into those tables and then use a `SELECT * FROM` statement to print the contents of each table. Put all of this SQL into file `twitter-queries.sql`. When you import this into postgres, cut and paste everything from the import command down into a file `twitter-results.txt` that you will hand in.

We may run your code to confirm that it matches the output you hand in.

Submission instructions

As before, you must declare your team (whether it is a team of one or two students) and hand in your work electronically using the MarkUs online system. Well before the submit deadline, you should declare your team in case you encounter any problems and need my help.

Once you have submitted, check that you have submitted the correct version; new or missing files will not be accepted after the due date. Remember that you may submit a new version of any file later (before the deadline, of course); look in the "Replace" column.

Be sure to review the course policy for grace days and late assignments.

What to submit

For Part 1, submit

- ultra.sql
  Your ultrasound queries.

For Part 2, submit

- twitter-schema.ddl
  This must include all statements that define your schema.
- twitter-data.ddl
  This must include all the INSERT INTO statements that your twitter parser generated to build the database.
- twitter-parser.*extension*, where *extension* is the appropriate extension for the programming language you used.
  The program you used to parse the twitter data file and generate twitter-data.ddl.
- twitter-queries.sql
  Definitions for tables to store query results, and INSERT INTO statements that use your queries to fill those tables.
- twitter-results.txt
  Output from your SQL queries.

Marking

For Part 1, we will use autotesting to help assess correctness. Therefore it is crucial that you follow the instructions given above for storing your query results in the specified tables. If you don't, your code will be considered incorrect.

You will not be graded for "efficiency". Since the DBMS will rewrite your queries for efficiency, you should focus on clarity. Some of your marks will be for aesthetic values such as concise and easy to understand SQL, consistent spacing and capitalization, and easy to read results in twitter-results.txt.

You may work at home, but you must make sure that your code runs on postgres on the cdf machines. Code that fails on cdf, even if it works in another environment, will be marked as not working.