Carnegie Mellon University
15-415 - Database Applications
Fall 2009, Faloutsos
Assignment 2: Formal Query Languages

Due: 9/15, 1:30pm, in class - hard-copy please

Reminders

- Weight: **5%** of the homework grade.
- Out of **100** points.
- Lead TA: B. Aditya Prakash
- Rough time-estimates: 2∼4 hours. (30-60mins for Question 2, 1-1.5hrs for each of Questions 1 and 3)
- You are encouraged to **type** your answers. Illegible handwriting may get no points, at the discretion of the grader. Greek letters, join symbols etc, may be hand-drawn, as long as they are neat and legible.

Remember that:

- There could be more than one correct answer. We shall accept them all.
- Whenever you are making an assumption, please state it clearly.

Question 1: Micro-blogging and Relational Algebra [55 points]

If you are not familiar with *Twitter*, please check `twitter.com`. We will work with a fictitious, simplified, Twitter-like setting, with the following specifications:

- Users post 'tweets', that is short pieces of text
- They may tag their tweets with zero or more tags of their own choice. For example a user tweeting about the G20 summit may decide to use the tag '*G20*' (prefixed by a 'sharp' sign: `#G20`, if we follow the convention imposed by the `twitter` site).
- A user 'u' may follow zero or more other users, which means that their 'tweets' are visible to user 'u' when he/she logs in.

For the above setting, we will use the following schema:

- `Person (pname, city, street)` - Assume the `pname` is unique
- `Follows (pname1, pname2)` - Person `pname1` *follows* person `pname2`

- `Tweets (tid, ttitle, ttext)` - *Tweet* with `tid` has title `ttitle` and text `ttext`

- `PersonTweets (pname, tid, ts)` - Person `pname` posted *tweet* `tid` at timestamp `ts`
- `TweetTag (tid, tagname)` - *Tweet* `tid` had `tagname` in its list of tags.

We now want to extract some information from the database using the power of relational algebra. For each of the following questions

(a) if they can be answered with relational algebra, give the corresponding expression **or**
(b) if impossible, state so clearly, and justify briefly (1-2 lines)

[**5 points each**]:

1.1 Find all the people (`pname`) who posted a tweet with tag 'Obama'.
1.2 Find all the different, distinct tags ever used.
1.3 Find all the tags 'Bob Smith' uses in his tweets. (i.e. *Bob's tweeting interests*)
1.4 Find all the tags 'Bob Smith' reads in the tweets of the people he follows. (i.e. *Bob's reading interests*)
1.5 Find all the people (`pname`) whose reading and tweeting interests don't intersect, i.e. who don't read the tags that they write about.
1.6 Find all the people (`pname`) from 'Pittsburgh' who used the tag 'G20'.
1.7 Find all the people (`pname` and `city`) who follow people who follow 'Ashton Kutcher' (i.e. *Second-level followers*)
1.8 Find all the people (`pname`) who could have potentially read a tweet about the IPhone 3GS, before its launch, i.e. all the people who follow people who posted a tweet with title 'New IPhone' or tag 'IPhone 3GS' before '00:00:00 06-19-2009'.
1.9 Find all the people (`pname` and `city`) who follow at least everyone that 'Bob Smith' follows.
1.10 Find the name (`pname`) of the full cascade of 'Bob Smith', that is, all the people that follow him directly or indirectly, through one or more intermediaries. That is, we want the people that follow 'Bob Smith' (first-level followers), union their followers (second-level followers), union their own followers, etc.
1.11 Find all pairs of people (`pname`) who have at least one follower in common.

Note: You can use any fundamental or derived operators shown in class or in the textbook. Also, feel free to create and use views. For example,

$$PITT\_PEOPLE \equiv \sigma_{city='Pittsburgh'}(Person)$$

defines a view PITT_PEOPLE containing everyone in Person from Pittsburgh.

Question 2: Relational Calculus [24 points]

2.1 Give relational tuple calculus (RTC) expressions for Questions 1.2, 1.4 and 1.7. [**4 points each**]
2.2 Give relational domain calculus (RDC) expressions for Questions 1.5, 1.8 and 1.9. [**4 points each**]

Question 3: Perfect Matches [21 points]

We are in-charge of an online social network where we are given a simple table with schema:

Hobbies (`pid, hobby`) - Person with `pid` has `hobby` as a hobby

The goal of this question is to write a **relational algebra query** that gives us *"perfect matches"* i.e. pairs of people with exactly the same hobbies. (As you have probably observed, social network sites try hard to recommend possible new friends to their users, to keep them returning) The "perfect match" query is tricky and involves double negation - so let's try to solve it by the following steps.

Warm up

3.1 Write a query to find all person-person-hobby triplets (`pid1, pid2, hobby`) such that person `pid1` has `hobby` as a hobby, but person `pid2` doesn't. Call this the `PPH` view. [**5 points**]

Hint: First try to find all the non-existing `(pid, hobby)` *pairs.*

Full query

3.2 Given the earlier view `PPH`, write the full relational algebra expression to find all pairs of people with the exact same set of hobbies. Remove mirror pairs and self-pairs (see note below). [**6 points**]

Verification with SQL

3.3 Give the equivalent SQL query of Q 3.2. (include the SQL query in hard copy) Again, make sure you remove mirror pairs and self-pairs. [**5 points**]

3.4 Run the SQL query of Q 3.3 on the SQLite3 sample database at `http://www.cs.cmu.edu/~christos/courses/dbms-F09/hws/hw2/15415-hw2.db` and paste the output. [**2 points**]

Sanity check, for the database file: running the command

```
your-machine% sqlite3 15415-hw2.db 'select count(*) from Hobbies;'
```

should give

```
30
```

3.5 Make a `query-hw2.txt` file such that running it on SQLite3 gives the output of Q 3.3 i.e. doing the following

```
your-machine% sqlite3 15415-hw2.db < query-hw2.txt
```

returns the desired pairs. Please e-mail the file to `badityap AT cs.cmu.edu` with subject *HW2 Query*. **[3 points]**

Important Note: A subtle point is that we want to **remove mirror pairs** *[like (a, b) and (b, a)] as well as* **self pairs** *[(a, a)]. The simplest way to remove both such pairs is to include a selection at the end:* `WHERE a < b`