**NYU Tandon School of Engineering** **October 4, 2016**
**Computer Science and Engineering**
**CS6083, Fall 2015**

# Problem Set #2 (due October 14)

## Problem 1:

In this problem, you have to write SQL and RA queries for a database modeling messages in a system similar to Twitter. Here are the tables:

User (uid, uname, ucity, ucountry);
Tweet (tid, uid, ttext, ttime);
Follows (uid, followerID);
Retweet (uid, tid, rtime);

Thus, users are identified by a *uid*, and we also store their name and the city and country they live in. Each tweet is identified by a unique *tid*, and contains the *uid* of the user sending it, the message text as a varchar(130), and the time and date the tweet was sent (*ttime*). The *Follows* relation stores which users follow which other users. Finally, a user can retweet a tweet by another user; in that case we store the *uid* of the user doing the retweet, the *tid* of the tweet, and the time and date the retweeting was done. Note that you cannot retweet a retweet, only retweet the original tweet. Of course, a tweet can be retweeted several times.

(a) Suppose the *Follows* table had only the attribute *uid* as primary key. Would that be a good idea? Suppose we remove the *rtime* attribute from the primary key of *Retweet*. How would that impact the schema?

(b) Create the above schema in a database system, choose appropriate attributes types, and define primary keys, foreign keys and other constraints. Data for this schema will be made available on NYU Classes, so please use that data. Load the data into the database using either insert statements or the bulk load facilities. You may use any mainstream relational database system.

(c) Write the following SQL queries and execute them on your database. Show the queries and the results:

(i) Output the names of all followers of the user with name "Jack Smith".
(ii) For each user, output their uid, their names, and the number of tweets they have sent.
(iii) Output the tid of any tweet sent by a user in Chicago that was later retweeted by a user in Miami.
(iv) Output any pair of cities (A, B) such that there are at least 10000 users in A who have a follower in B.
(v) Output the uid and name of the user(s) that has sent the most tweets.
(vi) For each month in 2016, output the city that had the largest number of tweets sent by users from that city.
(vii) We say that a user x *indirectly follows* a user y if x follows another user z that follows y. Output the uid of the user(s) with the largest number of indirect followers.

(d) Write expressions in Relational Algebra for queries (i) to vii.

(e) Write SQL statements to perform the following updates to the database:

(i) For every user who lists their hometown as "The Big Apple", change it to "New York City".
(ii) Delete all tweets sent by a user with name "D. Trump" that were sent between 3am and 5am in the morning.
(iii) For every user x that has retweeted at least 10 tweets by user y, make x a follower of y.

## Problems 2:

In this problem, you have to create views, write queries on the views, and create triggers, for the Twitter scheme above. Execute everything on your database system, and report the results.

(a) Define a view that contains for each user the uid, the uname, the city, the number of followers, and the number of tweets sent. Using this view only, write the following queries:

(i) For each city, output the user with the most followers.

(ii) Change the name of the user with uname "K. Smith" to "Kurt Smith".

(b) Consider the last query in Problem 1(e). Can you write a trigger that automatically makes user x a follower of user y once x retweets the $10^{th}$ tweet by user y?

(c) Write a trigger that disables additional tweets if a user has already sent 50 tweets during the last 60 minutes.

## Problem 3:

In this problem you have to design a database for online bartering or swapping, where users post information about items they want to have, and about items they have but do not need anymore, in the hope of finding someone willing to do a swap. In particular, you should design a database for book swapping; there are a number of real web sites that offer such services, such as bookmooch.com or paperbackswap.com.

In this scenario, users are identified by a unique user name, and they have a first and last name and an address. Books are identified by an ISBN number, and have a title, an author, and a short description. Also, books can be in 4 different conditions, "as new", "good condition", "wear and tear", and "damaged", and for each book we store its value in all four conditions. Note that this information about books and their values will probably be provided by the company running the swapping website, and that users can only swap books for which there are entries in the database. Users of the site maintain two lists, a list of "haves" (books they have and want to give away) and "wants" (books they want to get). For each of the "haves", users have to specify the ISBN and the condition, chosen from the above 4 conditions. For each of the wants, users need to specify the ISBN and a minimum condition they are willing to accept (if they specify "wear and tear" that means they are willing to accept the book in that condition or better). If users agree on a swap, then the complete information about the swap also has to be stored in the database. Note that one swap transaction may involve more than one book from each side; e.g., a user may give 3 books and get 2 books in return. Finally, users that have made a swap can later rate the transaction from one to five stars.

(a) Design a database for the above scenario using the ER model. Draw the ER diagram, show the cardinalities of all relationships, and identify primary keys and any weak entities.

(b) Convert your ER diagram into a relational schema. Identify all tables, attributes, primary keys, and foreign keys.

(c) Write SQL queries for the following questions. If you cannot answer a query using your schema, then you have to modify your solutions in (a) and (b) appropriately.

(i) List all pairs of users (x, y) where a swap *might be possible*, i.e., x wants at least one item that y has and y wants at least one item that x has.

(ii) For each user, output how many swaps they participated in during 2015.

(iii) Output the user names of the most uneven swap, i.e., a swap where one user receives much less in value than the other user. Note that a swap may involve several items on each side.

(iv) Output the names of any users who have only been involved in swaps that were rated 5 stars by both sides.

(d) Create tables in the database system, and insert some sample data (say, 5-10 tuples per table, but choose an interesting data set, so that the queries do not output empty results). Execute the queries in (c) and submit a log.