University of Toronto
CSC343, Fall 2017

Assignment 1: James Leja 1002527571 Chris Stalzer - need Chris's student

Unary operators on relations:

- $\Pi_{x,y,z}(R)$
- $\sigma_{condition}(R)$
- $\rho_{New}(R)$
- $\rho_{New(a,b,c)}(R)$

Binary operators on relations:

- $R \times S$
- $R \bowtie S$
- $R \bowtie_{condition} S$
- $R \cup S$
- $R \cap S$
- $R - S$

Logical operators:

- $\vee$
- $\wedge$
- $\neg$

Assignment:

- $New(a, b, c) := R$

Stacked subscripts:

- $\sigma_{this.something>that.something \wedge \\ this.otherthing \leq that.otherthing}$

Below is the text of the assignment questions; we suggest you include it in your solution. We have also included a nonsense example of how a query might look in LaTeX. We used `\var` in a couple of places to show what that looks like. If you leave it out, most of the time the algebra looks okay, but certain words, *e.g.*, "Offer" look horrific without it.

The characters "\\" create a line break and "[5pt]" puts in five points of extra vertical space. The algebra is easier to read with extra vertical space. We chose "–" to indicate comments, and added less vertical space between comments and the algebra they pertain to than between steps in the algebra. This helps the comments visually stick to the algebra.

Part 1: Queries

1. Find all the users who have never sent a message, but who have been sent at least one message. The message may have been sent to the user or to a group that the user belongs to. Report each user id.

   – Messages that have been sent.

   $hasSent(mID, uID, status) := \sigma_{status='sent'}Status$

   – uID of users who have received a message.

   $hasReceived(uID) := \Pi_{uID}(Status - hasSent)$

   – uID of users who have never sent a message.

   $neverSent(uID) := \Pi_{uID}User - \Pi_{uID}hasSent$

   $Q1(uID) := neverSent \cap hasReceived$

2. Net neutrality is dead, so EVL ISP wants to slow the service of poor users (users who do not use the app enough). To do this, find the users (and return their uid) who sent two or fewer messages in 2017.

   – Get all the messages that were sent in 2017

   $2017Msgs(mid, from, to, content, time) := \sigma_{time.year=2017}Message$

   – Get all the uIDs of the users who sent 3 or more messages in 2017.

   $3orMore2017(uID) :=$

   $$\Pi_{M1.uID}(\sigma_{\substack{M1.from=M2.from \\ \wedge M1.from=M3.from \\ \wedge M1.mid\neq M2.mid \\ \wedge M2.mid\neq M3.mid \\ \wedge M1.mid\neq M3.mid}}[(\rho_{M1}2017Msgs) \times (\rho_{M2}2017Msgs) \times (\rho_{M3}2017Msgs)]$$

   $Q2(uID) := \Pi_{uID}User - 3orMore2017$

3. Find the largest group. Report the group id. If there is a tie, report them all.

   Cannot be expressed.

4. Find privacy fanatics, that is, any user who has all her privacy settings set to none and who has never sent a message to another user who has privacy settings different than her own (meaning different than all none). Note that a private user (settings are all none) who has never sent a message would be considered a privacy fanatic. Return the users uid and name.

   – Get the uIDs of the users with all their privacy settings set to none.

   $PrivUsers(uID) := \Pi_{uID}(\sigma_{lastSeen="none" \wedge profilePhoto="none" \wedge profile="none"}Privacy)$

   – Get the uIDs of the "non-private" users.

$NonPrivUsers(uID) := \Pi_{uID} Privacy - PrivateUsers$

– Get messages sent from private to non-private users.

$PrivtoNonPriv(uID) := \Pi_{PrivateUsers.uID}(\sigma_{\substack{Message.from=PrivUsers.uID \\ \wedge Message.to=NonPrivUsers.uID}} (PrivUsers \times NonPrivUsers \times Message))$

– get the uIDs of the private users who only send to other privates.

$PrivtoPriv(uID) := PrivateUsers = PrivtoNonPriv$

$Q4(uID, name) := \Pi_{uID,name}(User \bowtie PrivtoPriv)$

5. Consider only users whose privacy settings state that everyone may see their lastSeen time (lastSeen = everyone). Among such users, report the uid, name and lastSeen of the user(s) whose lastSeen time is the most recent. Since times are ordered, the most recent time is the largest time value. If there are ties, report all users. These users have the most recent public lastSeen time. – Get the uid's of all users who's allow everyone to see their lastSeen time.

$LastSeenAll(uID) := \Pi_{uID}(\sigma_{lastSeen="Everyone"} Privacy)$

– Get the uID's and times of the users with open privacy settings.

$OpenUser(uID, lastSeen) := \Pi_{user.uID,user.lastSeen}(User \bowtie_{user.uID=LastSeenAll.uID})$

– Get all but the most recent lastSeen times of open users.

$NotRecent(uID) := \Pi_{OU1.uID}\sigma_{OU1.lastSeen<OU2.lastSeen}(\rho_{OU1} OpenUser \times \rho_{OU2} OpenUser)$

– Get uID's of the most recent open users $MostRecent(uID) := LastSeenAll - NotRecent$

– Connect the most recent uID's with the User relation to get their names and lastSeen times.

$Q5(uID, name, lastSeen) := \Pi_{Uid,name,lastSeen}(User \bowtie_{user.uID=MostRecent.uID})$

6. A users contact list can be sorted by the start time. Find users who send their first direct message to a contact in the same order as the contact list order. So if Sue is Pats oldest contact and Jo is the second oldest contact, then Pats first direct message to Sue happens before her first direct message to Jo and so on for all contacts. Include users with empty contact lists. Return users uid.

– Get the all the messages that have been sent directly to a user.

$Dmessage(mID, from, to, time) :=$

$$\Pi_{\substack{Message.mID \\ \wedge Message.from \\ \wedge Message.to \\ \wedge Message.time}} \sigma_{Message.to=User.uID}(Message \times \Pi_{uIDUser})$$

– Get the all but the earliest direct messages.

$AllButEarliest :=$

$$\Pi_{\substack{M1.mID \\ \wedge M1.from \\ \wedge M1.to \\ \wedge M1.time}} (\sigma_{\substack{M1.to=M2.to \\ \wedge M1.from=M2.from \\ \wedge M1.time>M2.time}} (\rho_{M1} DMessage \times \rho_{M2} DMessage))$$

– Get the earliest direct messages.

$Earliest = Dmessage - AllButEarliest$

– Get the uIDs of the users whose earliest direct messages are not in the same order as their contact start time.

$NotInOrder(uID) :=$

$$\Pi_{C1.uID}(\sigma_{\substack{C1.user=C2.user \\ \wedge C1.user=E1.from \\ \wedge C1.user=E2.from \\ \wedge C1.contact=E1.to \\ \wedge C1.contact=E2.to \\ \wedge C1.contact \neq C2.contact \\ \wedge C1.start>C2.start \\ \wedge E1.time<E2.time}} (\rho_{C1} Contact \times \rho_{C2} Contact \times \rho_{E1} Earliest \times \rho_{E2} Earliest))$$

$Q6(uID) := \Pi_{uID} User - NotInOrder$

7. Find the users who have always liked posts in the same order as the order in which they were posted, that is, users for whom the following is true: if they liked n different posts (posts of any users) and

   $$[post\_date\_1] < [post\_date\_2] < ... < [post\_date\_n]$$

   where $post\_date\_i$ is the date on which a post i was posted, then it holds that

   $$[like\_date\_1] < [like\_date\_2] < ... < [like\_date\_n]$$

   where $like\_date\_i$ is the date on which the post i was liked by the user. Report the user's name and email.

8. Report the name and email of the user who has gained the greatest number of new followers in 2017. If there is a tie, report them all.

9. For each user who has ever viewed any story, report their id and the id of the first and of the last story they have seen. If there is a tie for the first story seen, report both; if there is a tie for the last story seen, report both. This means that a user could have up to 4 rows in the resulting relation.

10. A comment is said to have either positive or negative sentiment based on the presence of words such as "like," "love," "dislike," and "hate." A "sentiment shift" in the comments on a post occurs at moment m iff all comments on that post before m have positive sentiment, while all comments on that post after m have negative sentiment — or the other way around, with comments shifting from negative to positive sentiment.

    Find posts that have at least three comments and for which there has been a sentiment shift over time. For each post, report the user who owns it and, for each comment on the post, the commenter's id, the date of their comment and its sentiment.

    You may assume there is a function, called *sentiment* that can be applied to a comment's text and returns the sentiment of the comment as a string with the value "positive" or

"negative". For example, you may refer to $sentiment(text)$ in the condition of a select operator.

Part 2: Additional Integrity Constraints

Express the following integrity constraints with the notation $R = \emptyset$, where R is an expression of relational algebra. You are welcome to define intermediate results with assignment and then use them in an integrity constraint.

1. A comment on a post must occur after the date-time of the post itself. (Remember that you can compare two date-time attributes with simple $<$, $>=$ etc.)
2. Each user can have at most one current story.
3. Every post must include at least one picture or one video and so must every story.