University of Toronto
CSC343, Fall 2017

Assignment 1: James Leja 1002527571
Christopher Stalzer 1002155286

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.

   – uIDs of users who have sent a message.

   $hasSent(uid) := \Pi_{user.uid}\ (\sigma_{message.from=user.uid}(Message \times User))$

   – uID of users who have received a message.

   $hasReceived(uid) := \Pi_{uid} Status$

   – uID of users who have never sent a message.

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

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

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.

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

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

$PrivtoNonPriv(uID) :=$

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

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

$PrivtoPriv(uID) := PrivUsers - PrivtoNonPriv$

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

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. Return all pairs of users with the same name. Return the two uids and the common name. Return each pair only once. (For example, if user 1 and user 2 are both named 'Pat', then return either [1, 2, 'Pat'] or [2, 1, 'Pat'] but not both).

– Get the uid's and name of each pair of users with the same name.

$$Q7(uId, uId, Name) := \Pi_{U1.uID, U2.uID, U1.name} \sigma_{\substack{U1.uID<U2.uID \\ \wedge U1.name=U2.name}} (\rho_{U1} User \times \rho_{U2} User)$$

8. For each user and contact, report the time that the first direct message was sent from the user to the contact and the time the last direct message was sent. Return the uid of the user (in an attribute named user) and the contact (in an attribute named contact) and the first time (earliest) (in an attribute named first) and last (most recent) time (in an attribute named last). If a user has not sent any direct messages to a contact then include the user and contact with the value 0 for both the first and last times.

– Get all direct messages

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

$\Pi_{mID, from, to, time} \sigma_{Message.to} = User.uidMessage \times User$

– Return uid's of Users that have sent no messages, and append two 0 attributes

$NoMessages(uid, uid, 0, 0) := \Pi_{uid, contact}(Contact) - \Pi_{to}, from(dMessage)$

–Gather all but the earliest messages

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

$$\Pi_{M1.mid, M1.from, M1.time} \sigma_{\substack{M1.time>M2.time \\ \wedge M1.from=M2.from \\ \wedge M1.to=M2.to}} \rho_{M1} dMessage \times \rho_{M2} dMessage$$

–Get the earliest messages of each user

$Earliest(mID, from, to, content, time) := dMessage - AllButEarliest$

–Gather all but the latest direct messages messages

$AllButLatest(mid, from, to, time) :=$

$\Pi_{M1.mid,M1.from,M1.time}\sigma_{\substack{M1.time<M2.time \\ \wedge M1.from=M2.from \\ \wedge M1.to=M2.to}} \rho_{M1}dMessage \times \rho_{M2}dMessage$

–Get the latest messages for each user

$Latest(mid, from, to, time) := dMessage - AllButLatest$

–Combine earliest and latest and return the from uid, to uid, and their times

$EarliestLatest(uid, uId, time, time) :=$

$\Pi_{E.from,E.to,E.time,L.time}\ \sigma_{\substack{E.from=L.from \\ \wedge E.to=L.to}}\ \rho_E Earliest \times \rho_L Latest$

–Return User's first and last message and their times and Users with no messages

$Q8(User, Contact, First, Last) := EarliestLastest \cap NoMessages$

9. A 'spammer' is a user who posts unwanted direct messages that are not read. A spammer must have sent at least direct message (so this message will appear in the Status relation). Because users may not be aware that someone is a spammer, they may read some of their initial messages. However, once they decide a certain user is a spammer, the receivers stop reading all messages from the spammer. This means that for a user who is sent a direct message from a spammer there are no delivered messages with a time that is earlier than any read message from the spammer. Return the spammer's user id and all their privacy settings (Privacy.lastSeen, Privacy.photo, Privacy.profile). Do not consider groups for this question. Only consider direct messages sent from a user to another single user (not to a group).

– Get all direct messages.

$dMessage(mID, from, to, content, time) := \Pi_{mID,from,to,content,time}$
$\sigma_{Message.to=User.uID}Message \times User$

$dMessage\&Status() := dMessage \bowtie Status$

$NotSpam(uID) := \Pi_{D1.from}\sigma_{\substack{D1.mID!=D2.mID \\ \wedge D1.from=D2.from \\ \wedge D1.to=D2.to \\ \wedge D1.time>D2.time \\ \wedge D1.status="Read" \\ \wedge D2.status="Delivered"}} \rho_{D1}dMessage\&Status \times$

$\rho_{D2}dMessage\&Status$

$Spammer(uID) := User - NotSpam$

$Q9(uID, lastSeen, Photo, Profile) := Spammer \bowtie Privacy$

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. The receiver (Message[to]) of a message must be either a user (User[uid]) or a group (Group[gid]).

   – Comment?

   $Message[to] - (User[uid] \cap Group[gid]) = \emptyset$

2. A user can only send messages to users in her contacts (Contact[contact]) and the time of the message must be after the start of the contact. This includes direct messages sent to a user and messages sent to a group. All members of the group must be in the user's contacts.

   – All the direct messages.

   $DirectMsg(mid, from, to, content, time) :=$

   $$\Pi_{\substack{Message.mid,\\ Message.from,\\ Message.to,\\ Message.content,\\ Message.time}} \sigma_{Message.to=User.uid}(Message \times \Pi_{uid} User)$$

   – All the group messages.

   $GroupMsg := Message - DirectMsg$

   – A relation of all the combinations of group messages and group where the constraints are followed (i.e. all members of the group are part of the sender's contacts, and the message was sent after the contact start time).

   $GrpCorrect(mid, from, to, content, time, gid, uid) :=$

   $$\Pi_{\substack{GroupMsg.mid,\\ GroupMsg.from,\\ GroupMsg.to,\\ GroupMsg.content,\\ GroupMsg.time,Group.gid,\\ Group.uid}} \sigma_{\substack{GroupMsg.from=Contact.user\\ \wedge GroupMsg.to=Group.gid\\ \wedge Group.uid=Contact.contact\\ \wedge GroupMsg.time>Contact.start}} (GroupMsg \times Group \times Contact)$$

   – Get all the instances that violate the group constraint.

   $GrpViolators(mid, from, to, content, time, gid, uid) :=$

   $(\sigma_{GroupMsg.to=Group.gid}(GroupMsg \times Group)) - GrpCorrect$

   – Get all the mids where a user sends a direct message to a contact, and the

message is sent after Contact.start.

$$DirCorrect(mid) := \Pi_{mid}(\sigma_{\substack{DirectMsg.from=Contact.user \\ \wedge DirectMsg.to=Contact.contact \\ \wedge DirectMsg.time>Contact.start}} (DirectMsg \times Contact))$$

– Get all the message mids that violate the direct message constraint.

$$DirViolators(mid) := \Pi_{mid} DirectMsg - DirCorrect$$

– Expressing the entire integrity constraint.

$$DirViolators = \emptyset \ \wedge \ GrpViolators = \emptyset$$

3. The total size of all attachments in a message must be less than 128MB.

   – Cannot be expressed

4. The Status relation may not contain a message and user if the message was not sent to that user (either directly or the user was part of a group that received the message).

   – Natural join of Status and Message.

$$MsgWithStatus(mid, uid, status, from, to, content, time) := Status \bowtie Message$$

   – Get the group messages from MsgWithStatus.

$$GrpMsgWithStatus := \sigma_{MsgWithStatus.uid!=MsgWithStatus.to} \ MsgWithStatus$$

   – Get the direct messages from MsgWithStatus.

$$DirMsgWithStatus := MsgWithStatus - GroupMsgWithStatus$$

   – All the instances where a messsage was sent to a group, and a user in that group has a tuple in status

$$GroupStatus(mid, uid, status) :=$$
$$\Pi_{mid,uid,status}(\sigma_{\substack{GrpMsgWithStatus.to=Group.gid \\ \wedge GrpMsgWithStatus.uid=Group.uid}} (GrpMsgWithStatus \times Group))$$

   – All instances that should make up Status.

$$ShouldBeStatus := GroupStatus \cup \Pi_{mid,uid,status} DirMsgWithStatus$$

   – The integrity constraint.

$$Status - ShouldBeStatus = \emptyset$$