University of Toronto
CSC343, Winter 2018

Assignment 1

Due: Friday Feb 2, 2018, 5pm

Learning Goals

By the end of this assignment you should be able to:

1. read a new relational schema, and determine whether or not a particular instance is valid with respect to that schema,
2. apply the individual techniques for writing relational algebra queries and integrity constraints that we learned in class,
3. combine the individual techniques to solve complex problems, and
4. identify problems that cannot be solved using relational algebra.

These skills we leave you well prepared to learn SQL.

Later in the course, you will learn about how to develop your own schema based on knowledge of the domain. Even though developing a schema is really the first step in building a database, it is a more advanced task than querying an existing database; this is why we will be learning about it and practicing it later.

Domain

For this assignment, you will operate on a database for WhatsUpp, an internet SMS application. WhatsUpp has been acquired by EVL ISP. You need to know a few facts about the domain.

- WhatsUpp maintains a set of **contacts** for each user which includes all other users to whom they have sent a message.
- A **message** has a content and may have one or more attachments (photos, videos or documents).
- A message may be sent to a single user or to a group. Each group has a group id (gid). A message to a single user is a *direct* message.

Schema

Relations

- User(uid, name, profile, email, phone, photo, lastSeen)

  A tuple in this relation represents a WhatsUpp user, `uid` is a unique identifier. `name`, `profile`, `email`, and `phone` are information about this user. `photo` is the url of the profile photo of this user. `lastSeen` is the last time this user used WhatsUpp. Assume the time is a value that includes the date and time.
- Contact(user, contact, start)

  A tuple in this relation represents the fact that the user with identifier `user` may send messages to the user `contact` after the time `start`.

- Group(gid, uid)

  `uid` is a member of group `gid`. Group ids (`gid`) and user ids (`uid`) can both appear in `Message[to]` (next relation), and the set of group ids does not overlap with the set of user ids (see constraint below). So a group id cannot be a user id and vice verse.

- Message(mid, from, to, content, time)

  A message is sent by a user (`from`) to another user or to a group (`to`). Content represents the text in the message. `time` is the time (a value that represents the date and time) of when the message was sent.

- Attachment(mid, contentId, size)

  A tuple in this relation represents the fact that the photo, video, or document stored at url `contentId` is included in message `mid.size` is the size in megabytes (MB) of the attachment.

- Privacy(uid, lastSeen, photo, profile)

  The privacy settings of user `uid`. The value of all three non-key attributes is: none, contacts, everyone. If `lastSeen` has a value of `none`, then no one can see the user's lastSeen value. If `lastSeen` has a value of `contacts`, then only users in the contact list (in `Contact[contact]`) can see the user's lastSeen value. Finally, a value of `everyone` means all WhatsUpp users can see the when the user was lastSeen on WhatsUpp. The semantics of the attributes photo and profile are the same. For example, `[101, none, contacts, everyone]` means that no one can see user 101's lastSeen time, only her contacts can see her photo, and everyone may see her profile.

- Status(mid, uid, status)

  A tuple in this relation represents the fact that message `mid` has been sent (from the creator of the message), delivered to user `uid`, or read by user `uid`. So the attribute `status` has one of the three values: `sent, delivered`, or `read`. When a message `mid` is sent to a user `uid` (either directly or as a member of a group), a tuple `Status(mid, uid, 'sent')` is created. The status attribute is updated once the receiver `uid` has received (`status = delivered`) or read (`status = read`) the message.

Integrity constraints

- Contact[user] $\subseteq$ User[uid]
- Contact[contact] $\subseteq$ User[uid]
- Group[uid] $\subseteq$ User[uid]
- Group[gid] $\cap$ User[uid] $= \emptyset$
- Message[from] $\subseteq$ User[uid]
- Attachment[mid] $\subseteq$ Message[mid]
- Attachment[size] ≤ 16
- Privacy[uid]] $\subseteq$ User[uid]
- Status[mid] $\subseteq$ Message[mid]
- Status[uid] $\subseteq$ User[uid]
- Status[status] $\subseteq$ { "sent", "delivered", "read" }
- Privacy[lastSeen] $\subseteq$ { "none", "contacts", "everyone" }

- Privacy[profilePhoto] $\subseteq$ { "none", "contacts", "everyone" }
- Privacy[profile] $\subseteq$ { "none", "contacts", "everyone" }

Warmup: Getting to know the schema

To get familiar with the schema, ask yourself questions like these (but don't hand in your answers):

- Can the same user appear multiple times in a group?
- Can a user send a message to a group that includes herself?
- Can the database store the status of a message to a user who was not sent the message?
- How does the schema allow *any* number of photos or videos to be included in a message, but restrict a user to having only one profile photo?
- Can the total size of a message with all its attachments be more than 16 MB?

Part 1: Queries

Write the queries below in relational algebra. There are a number of variations on relational algebra, and different notations for the operations. You must use the same notation as we have used in class and on the slides. You may only use assignment, and the operators we have used in class: $\Pi, \sigma, \bowtie, \bowtie_{condition}, \times, \cap, \cup, -, \rho$. Assume that all relations are sets (not bags), as we have done in class, and do not use any of the extended relational algebra operations from Chapter 5 of the textbook (for example, do not use the division operator or aggregation).

Some additional points to keep in mind:

- Do not make any assumptions about the data that are not enforced by the original constraints given above. Your queries should work for any database that satisfies those constraints.
- Assume that every tuple has a value for every attribute. For those of you who know some SQL, in other words, there are no null values.
- The condition on a select operation can use comparison operators (such as $\leq$ and $\neq$) and boolean operators ($\vee$, $\wedge$ and $\neg$). Simple arithmetic is also okay, `e.g.`, attribute1 $\leq$ attribute2 + 5000.
- You may assume that all attribute values are ordered so they can be compared with $=, <, >, \leq, \geq,$ *etc.* You may refer to the year component of a time attribute d using the notation d.year. For Q8, you may assume 0 is a valid time value.
- You are encouraged to use assignment to define intermediate results.
- It's a good idea to add commentary explaining what you're doing. This way, even if your final answer is not completely correct, you may receive part marks.
- The order of the columns in the result doesn't matter.
- When asked for a maximum or minimum, if there are ties, report all of them.

If a query cannot be expressed in relational algebra (with the operators we are considering in this assignment), then, simply write "cannot be expressed". Note: The queries are not in order according to difficulty.

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.

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.

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

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 user's *uid* and `name`.

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.

6. A user's 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 Pat's oldest contact and Jo is the second oldest contact, then Pat's 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 user's `uid`.

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).

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.

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).

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.

If a constraint cannot be expressed in relational algebra (with the operators we are considering in this assignment), simply write "cannot be expressed".

1. The receiver (`Message[to]`) of a message must be either a user (`User[uid]`) or a group (`Group[gid]`).
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.
3. The total size of all attachments in a message must be less than 128MB.
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).

When writing your queries for Part 1, don't assume that these additional integrity constraints hold.

Style and formatting requirements

In order to make your algebra more readable, and to minimize errors, we are including these style and formatting requirements:

- In your assignment statements, you must include names for all attributes in the intermediate relation you are defining. For example, write

  $$HighestGrade(sID, oID, grade) := \quad \ldots$$

- Use meaningful names for intermediate relations and attributes, just as you would in a program.
- If you want to include comments, put them before the algebra that they pertain to, not after. Make them stand out from the algebra, for example by using a different font. For example, this looks reasonable:

  – Students who had very high grades in any offering of a csc course.
  $High(sID) := \Pi_{sID}\sigma_{dept='csc' \wedge grade>95}(Took \bowtie Offering)$

A modest portion of your mark will be for good style and formatting.

Submission instructions

Your assignment must be typed; handwritten assignments will not be marked. You may use any word-processing software you like. Many academics use LaTeX. It produces beautifully typeset text and handles mathematical notation well. If you would like to learn LaTeX, there are helpful resources online. Whatever you choose to use, you need to produce a final document in pdf format.

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. Instructions for doing so are posted on the Assignments page of the course website. Well before the due date, you should declare your team and try submitting with MarkUs. You can submit an empty file as a placeholder, and then submit a new version of the file later (before the deadline, of course); look in the "Replace" column.

For this assignment, hand in just one file: A1.pdf. If you are working in a pair, only one of you should hand it in.

Check that you have submitted the correct version of your file by downloading it from MarkUs; new files will not be accepted after the due date.