University of Toronto
CSC343, Fall 2017

Assignment 1: Relational Algebra
ZHANG Qianhao / 1004654377
LI XU / 1004654466

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 out it.

The characters "\\" create a line break and "[5pt]" puts in five points of extra vertical space. The algebra is easier to read 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 liked or viewed a post or story of a user that they do *not* follow. Report their user id and "about" information. Put the information into a relation attributes "username" and "description".

   – uid of liker and uid of the poster of the liked post.

   $LikerLiked(uid1, uid2) := \Pi_{liker,uid}(Likes \bowtie Post)$

   – uid of viewer and uid of the writer of the story viewed.

   $ViewerViewed(uid1, uid2) := \Pi_{viewerid,uid}(Saw \bowtie Story)$

   – uid of follower and uid of the followed user.

   $FollowerFollowed(uid1, uid2) := \Pi_{follower,followed} Follow$

   – uid of user who liked or saw from non-following users.

   $BroadReader(uid) := \Pi_{uid1}((LikerLiked - FollowerFollowed) \cup (ViewerViewed - FollowerFollowed))$

   – uid of user who never liked or saw from non-following users.

   $NonBroadReader(uid) := BroadReader - \Pi_{uid} User$

   – answer.

   $Answer(username, description) := \Pi_{uid,about}(NonBroadReader \bowtie User)$

2. Find every hashtag that has been mentioned in at least three post captions on every day of 2017. You may assume that there is at least one post on each day of a year.

   – Firstly we join the two table in order to get the data simultaneously

   $PostWithTag(pid, when, tag) := \Pi_{pid,when,tag}[HashTag \bowtie_{HashTag.pid=Post.pid} Post]$

   – Secondly we assume the date is 01/01/2017 and find all the tags that matched the conditon.

   $$Tags^{01/01/2017}(tag) := (\Pi_{tag}\sigma_{\substack{when.day=01 \\ \wedge when.month=01 \\ \wedge when.year=2017 \\ \wedge T1.tag=T2.tag \\ \wedge T2.tag=T3.tag}}[(\rho_{T1} PostWithTag) \times (\rho_{T2} PostWithTag) \times (\rho_{T3} PostWithTag)])$$

   –By the same process as above we can get $Tags^{01/02/2017} \cdots Tags^{10/11/2017}$ all the set

   – Then we intersect all the set from date 01/01/2017 til 10/11/2017.

   $Answer(tag) := Tags^{01/01/2017} \cap Tags^{01/02/2017} \cap \cdots \cap Tags^{10/11/2017}$

3. Let's say that a pair of users are "reciprocal followers" if they follow each other. For each pair of

reciprocal followers, find all of their "uncommon followers": users who follow one of them but not the other. Report one row for each of the pair's uncommon follower. In it, include the identifiers of the reciprocal followers, and the identifier, name and email of the uncommon follower.

– uid of follower and uid of the followed user.

$FollowerFollowed(uid1, uid2) := \Pi_{follower,followed} Follow$

– uid of followed user and uid of the follower.

$FollowedFollower(uid1, uid2) := \Pi_{followed,follower} Follow$

– uid of reciprocal user pair out duplicate.

$ReciprocalPair(uid1, uid2) := \sigma_{uid1<uid2}(FollowerFollowed \cap FollowedFollower)$

– uid of users following the first user in a reciprocal pair (user not from the pair).

$FollowFirst(uid, uid1, uid2) := \Pi_{uid,uid1,uid2}(Follows \bowtie_{Follows.follower \neq ReciprocalPair.uid1 \wedge Follows.follower \neq ReciprocalPair.uid2 \wedge Follows.followed = ReciprocalPair.uid1} ReciprocalPair)$

– uid of users following the second user in a reciprocal pair (user not from the pair).

$FollowSecond(uid, uid1, uid2) := \Pi_{uid,uid1,uid2}(Follows \bowtie_{Follows.follower \neq ReciprocalPair.uid1 \wedge Follows.follower \neq ReciprocalPair.uid2 \wedge Follows.followed = ReciprocalPair.uid2} ReciprocalPair)$

– uid of users following only one user from a reciprocal pair.

$UncommonFollower(uid, uid1, uid2) := (FollowFirst \cup FollowSecond) - (FollowFirst \cap FollowSecond)$

– answer.

$Answer(uid, uid1, uid2, name, email) := UncommonFollower \bowtie_{UncommonFollower.uid=User.uid} (\Pi_{uid,name,email} User)$

4. Find the user who has liked the most posts. Report the user's id, name and email, and the id of the posts they have liked. If there is a tie, report them all.

–Cannot be expressed.

–There is no way of getting the number of the likes of the user without using count function.

5. Let's say a pair of users are "backscratchers" if they follow each other and like all of each others' posts. Report the user id of all users who follow some pair of backscratcher users.

– uid of follower and uid of the followed user.

$FollowerFollowed(uid1, uid2) := \Pi_{follower,followed} Follow$

– uid of followed user and uid of the follower.

$FollowedFollower(uid1, uid2) := \Pi_{followed,follower} Follow$

– uid of reciprocal user pair out duplicate.

$ReciprocalPair(uid1, uid2) := \sigma_{uid1<uid2}(FollowerFollowed \cap FollowedFollower)$

– all necessary liker-pid-liked tuples for first user from reciprocal pair to back-scratch second user.

$Scratch1(liker, pid, liked) := \Pi_{uid1,pid,uid2}(ReciprocalPair \bowtie_{ReciprocalPair.uid2=Post.uid} Post)$

– liker-pid-liked tuples for first user from reciprocal pair who has actually liked second user's post.

$Tickle1(liker, pid, liked) := \Pi_{liker,pid,liked}(Scratch1 \bowtie_{\substack{Scratch1.liker=Likes.liker \\ \wedge Scratch1.pid=Likes.pid}} Likes)$

– liker-liked tuples where liker back-scratches liked (note it is trivial if liked never posted).

$Scratcher1(liker, liked) := \rho_{ReciprocalPair_{(liker,liked)}} ReciprocalPair - \Pi_{liker,liked}(Scratch1 - Tickle1)$

– all necessary liker-pid-liked tuples for second user from reciprocal pair to back-scratch first user.

$Scratch2(liker, pid, liked) := \Pi_{uid2,pid,uid1}(ReciprocalPair \bowtie_{ReciprocalPair.uid1=Post.uid} Post)$

– liker-pid-liked tuples for second user from reciprocal pair who has actually liked first user's post.

$Tickle2(liker, pid, liked) := \Pi_{liker,pid,liked}(Scratch2 \bowtie_{\substack{Scratch2.liker=Likes.liker \\ \wedge Scratch2.pid=Likes.pid}} Likes)$

– liker-liked tuples where liker back-scratches liked (note it is trivial if liked never posted).

$Scratcher2(liker, liked) := \rho_{ReciprocalPair_{(liker,liked)}} \Pi_{uid2,uid1} ReciprocalPair - \Pi_{liker,liked}(Scratch2 - Tickle2)$

– backscratchers, note that Scratch2's liker and liked are reversed in order to match Scratch1's notion where the first user in a reciprocal pair appears first int the tuple.

$BackScratcherPair(uid1, uid2) := (\rho_{Scratcher1_{(uid1,uid2)}} Scratcher1) \cap (\rho_{Scratcher2_{(uid1,uid2)}} \Pi_{liked,liker} Scratcher2)$

– uid of users following the first user in a backscratcher pair (user not from the pair).

$FollowFirst(uid, uid1, uid2) := \Pi_{uid,uid1,uid2}(Follows \bowtie_{\substack{Follows.follower \neq BackScratcherPair.uid1 \\ \wedge Follows.follower \neq BackScratcherPair.uid2 \\ \wedge Follows.followed = BackScratcherPair.uid1}} BackScratcherPair)$

– uid of users following the second user in a backscratcher pair (user not from the pair).

$FollowSecond(uid, uid1, uid2) := \Pi_{uid,uid1,uid2}(Follows \bowtie_{\substack{Follows.follower \neq BackScratcherPair.uid1 \\ \wedge Follows.follower \neq BackScratcherPair.uid2 \\ \wedge Follows.followed = BackScratcherPair.uid2}} BackScratcherPair)$

– answer.

$Answer(uid) := \Pi_{uid}(FollowFirst \cap FollowSecond)$

6. The "most recent activity" of a user is his or her latest story or post. The "most recently active user" is the user whose most recent activity occurred most recently.

   Report the name of every user, and for the most recently active user they follow, report their name and email, and the date of their most-recent activity. If there is a tie for the most recently active user that a user follows, report a row for each of them.

   – Get the post that not most recent of every user

   $NotMostRecentPost(uid, when) := (\Pi_{uid,P1.when}\sigma_{\substack{P1.uid=P2.uid \wedge \\ P1.when<P2.when}}[(\rho_{P1}Post)\times(\rho_{P2}Post)])$

   – Get the most recent post of every user

   $MostRecentPost(uid, when) := (\Pi_{uid,when}[Post]) - NotMostRecentPost$

   – Get the story that not most recent of every user

   $NotMostRecentStory(uid, when) := (\Pi_{uid,S1.when}\sigma_{\substack{S1.uid=S2.uid \wedge \\ S1.when<S2.when}}[(\rho_{S1}Post)\times(\rho_{S2}Post)])$

   – Get the most recent story of every user

   $MostRecentStory(uid, when) := (\Pi_{uid,when}[Story]) - NotMostRecentStory$

   –Find the users that their most recent Activities are Posts

   $MostRecentActivityPost(uid, when) := (\Pi_{uid,P.when}\sigma_{\substack{P.uid=S.uid \wedge \\ P.when \geq S.when}}[(\rho_{P}MostRecentPost)\times (\rho_{S}MostRecentStory)])$

   –Find the users that their most recent Activities are Stories

   $MostRecentActivityStory(uid, when) := (\Pi_{uid,S.when}\sigma_{\substack{P.uid=S.uid \wedge \\ P.when<S.when}}[(\rho_{P}MostRecentPost)\times (\rho_{S}MostRecentStory)])$

   – Union the two sets above together to get all most recent acitivities of all users

   $MostRecentActivity(uid, when) := MostRecentActivityPost \cup MostRecentActivityStory$

   –Join the MostRecentActivity table and Follows table together in order to get the data simultaneously

   $FollowsWithActivity(follower, followed, start, when) :=$
   $Follows \bowtie_{Follows.followed=MostRecentActivity.uid} MostRecentActivity$

   –After finding the most recent activity, we can then find the not most recent followed user by self-join

   $NotMostRecent(follower, followed, when) := (\Pi_{follower,F1.followed,F1.when}\sigma_{\substack{F1.follower=F2.follower \wedge \\ F1.when<F2.when}}$
   $[(\rho_{F1}FollowsWithActivity) \times (\rho_{F2}FollowsWithActivity)])$

   –Get the followed user of most recent activity

   $MostRecentFollowed(follower, followed, when) :=$

$(\Pi_{follower,followed,when}[FollowsWithActivity]) - NotMostRecent$

–Finally, we get the answer by joining the User table twice

$Answer(name1, name2, email, when) := \Pi_{U1.name,U2.name,U2.email,F.when}\sigma^{U1.uid=F.follower\wedge}_{U2.uid=F.followed}$
$[(\rho_{U1}User) \bowtie (\rho_F Follows) \bowtie (\rho_{U2}User)]$

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.

– Cannot be expressed because the number of likes of a user is undetermined. There is no way to get a proper ordering of the likes.

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.

–Cannot be expressed. Because there is no way to count the number of followers of a user without using the count function.

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 all; if there is a tie for the last story seen, report all.

– viewer with their earliest view sid.

$Earliest(viewerid, earliestsid) := \Pi_{viewerid,sid}(Saw -$
$\Pi_{S1.viewerid,S1.sid,S1.when}(\sigma_{S1.viewerid=S2.viewerid \wedge S1.when>S2.when}(\rho_{S1}Saw \times \rho_{S2}Saw)))$

– viewer with their latest view sid.

$Latest(viewerid, latestsid) := \Pi_{viewerid,sid}(Saw -$
$\Pi_{S1.viewerid,S1.sid,S1.when}(\sigma_{S1.viewerid=S2.viewerid \wedge S1.when<S2.when}(\rho_{S1}Saw \times \rho_{S2}Saw)))$

– answer.

$Answer(viewerid, earliestsid, latestsid) := Earliest \bowtie Latest$

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

– The first thing to do is certainly find all the posts that have at least 3 comments

$$PostOfAtLeastThreeComment(pid, commenter, when, text) := (\Pi_{pid}\sigma_{\substack{C1.pid=C2.pid\wedge \\ C2.pid=C3.pid\wedge \\ C1.when\neq C2.when\wedge \\ C1.when\neq C3.when\wedge \\ C1.when\neq C3.when}} [(\rho_{C1}Comment) \times (\rho_{C2}Comment) \times (\rho_{C3}Comment)]) \bowtie Comment$$

– Next we should choose the posts have sentiment shift from above

$$ContainSentimentShiftPost(pid) :=$$
$$\Pi_{pid}\sigma_{sentiment(C1.text)\neq sentiment(C2.text)}$$
$$[(\rho_{C1}PostOfAtLeastThreeComment) \times (\rho_{C2}PostOfAtLeastThreeComment)]$$

– Finally, we can join the pid table with other table to get the information complete

$$Answer(uid, commenter, when, sentiment) :=$$
$$(\Pi_{uid}[ContainSentimentShiftPost \bowtie_{ContainSentimentShiftPost.pid=Post.pid} Post])$$
$$\cup$$
$$(\Pi_{commenter,when,sentiment(Comment.text)}$$
$$[ContainSentimentShiftPost \bowtie_{ContainSentimentShiftPost.pid=Comment.pid} Comment])$$

Part 2: Additional Integrity Constraints

Express the following integrity constraints the notation $R = \emptyset$, where R is an expression of relational algebra. You are welcome to define intermediate results 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 simple <, >= etc.) – post associated with its like.

   $$PostLike(pid, postwhen, likewhen) := \Pi_{Post.pid,Post.when,Likes.when}(Post \bowtie_{Post.pid=Likes.pid} Like)$$

   – constraint.

   $$\sigma_{postwhen>=likewhen} Postlike = \emptyset$$

2. Each user can have at most one current story.

   – Story pairs from the ones from the same user.

   $$StoryPair(uid, sid1, sid2, current1, current2) := \Pi_{S1.uid,S1.sid,S2.sid,S1.current,S2.current}$$
   $$\sigma_{\substack{S1.uid=S2.uid \\ \wedge S1.sid<s2.sid}}(\rho_{S1}Story \times \rho_{S2}Story)$$

– constraint.

$\sigma_{current1=true \wedge current2=true} StoryPair = \emptyset$

3. Every post must include at least one picture or one video and so must every story.

   – set of actual post with url.

   $PostWithURL(pid) := \Pi_{pid}(Post \bowtie PIncludes)$

   – set of actual story with url.

   $StoryWithURL(sid, url) := \Pi_{sid}(Story \bowtie SIncludes)$

   – constraint.

   $(\Pi_{pid} Post - PostWithURL) \cap (\Pi_{sid} Story - StoryWithURL) = \emptyset$