Question 1. [12 MARKS]

Consider this schema for Twitter, a social media platform where users post messages called "tweets".

Consider this relational schema for Twitter data. Keys are underlined.

As this question considers relational algebra, assume all relations are sets containing no nulls.

User(<u>userID</u>, name, email)

A Twitter user.

Tweet(<u>tweetID</u>, userID, content, date)

The user with *userID* made a tweet containing *content* on *date*.

Follows(<u>a, b</u>)

User *a* follows user *b* on Twitter, which means that
a has subscribed to *b*'s tweets.

Likes(<u>who, what, when</u>, stars)
User *who* liked tweet *what* on date *when*.
Rating is recorded in stars.

Tweet[userID] ⊆ User[userID]

Follows[a] ⊆ User[userID]

Follows[b] ⊆ User[userID]

Likes[who] ⊆ User[userID]

Likes[what] ⊆ Tweet[tweetID]

Part (a) [2 MARKS]

Does the scheme enforce this constraint: a user cannot like the same tweet twice? Circle one:

Yes No.

If yes, explain; if not write a new constraint to enforce it. You cannot change any of the existing constraints, rather write a new constraint to enforce it:

Solution:

no

$Likes1(who1, what1, when1) := \Pi_{who,what,when}(Likes)$

$Likes2(who2, what2, when2) := \Pi_{who,what,when}(Likes)$

$Likes1 \bowtie_{who1=who2 \wedge what1=what2 \wedge when1<>when2} Likes2 = \emptyset$

Part (b) [2 marks]

Does the scheme enforce this constraint: You can only follow someone who has a tweet you like. Circle one:

Yes No.

If yes, explain; if not write a new constraint to enforce it:

Solution:

no

$follow(a, b) \subseteq \pi_{who,userID}(likes \bowtie_{what=tweetID} tweets)$

Part (c) [2 marks]

Suppose relation *User* has 100 tuples. How many tuples could *Follows* have? Circle all that apply:

0 1 100 10,000 100,000

not last one because max 100 x 100

Solution:

all are possible except last (100,000)

Part (d) [4 MARKS]

Which of the following pairs of queries are equivalent? Circle each pair that returns the same results on all database instances.

1. $\Pi_{userID}(Tweet \bowtie User) = \Pi_{userID}(Tweet)$

   Solution: Yes

2. $User = User \bowtie User$

   Solution: Yes

3. $\Pi_{name}(User) = \Pi_{name}(User \bowtie_{userID=who} Likes)$

   Solution: No – right could have fewer

4. $\Pi_{when}(Likes) \cap \rho_{date \rightarrow when}\Pi_{date}(Tweet) = \Pi_{when}(Tweet \bowtie_{when=date} Likes)$

   Solution: Yes

Part (e) [2 MARKS]

Which of the following queries can be expressed using the same form of relational algebra that we used in class and on Assignment 1, meaning: assignment, and the operators $\Pi, \sigma, \bowtie, \bowtie_{condition}, \times, \cap, \cup, -, \rho$? Circle all that apply.

1. Find the oldest Tweet(s). This means the Tweet(s) whose date is the oldest in the database instance.

   Solution: Yes

2. Find all tweets that no one has liked.

   Solution: Yes

3. Find all people who only like their own tweets.

   Solution: Yes

   cannot count

4. Find people who have made the most tweets. If there is a tie, report them all.

   Solution: No

5. The trust relation is defined as follows. If a follows b, then a trusts b. In addition, if b follows c and a trusts b then a trusts c. Find all users who the user 'Colonel Chris Hadfield' trusts.

   Solution: No

   no

Question 2. [8 marks]

Here is the schema from Assignment 1. A few attributes and relations have been omitted for simplicity.

Relations

Product(DIN, manufacturer, name, form, schedule)
A tuple in this relation represents a drug product.

Price(DIN, price)
The price of a drug product.

Prescription(RxID, date, patient, drug, doctor)
A prescription for *drug* was written on *date* for *patient* by *doctor*. Attribute *patient* is the patient's OHIP number.

Filled(RxID, date, pharmacist)
Prescription *RxID* was filled by *pharmacist* on *date*.
Attribute *pharmacist* is the pharmacist's OCP number.

Integrity constraints

Price[DIN] $\subseteq$ Product[DIN]

Prescription[drug] $\subseteq$ Product[DIN]

Filled[RxID] $\subseteq$ Prescription[RxID]

Π_{schedule}Product $\subseteq$ {"prescription", "narcotic", "OTC"}

Let's say a pharmacist has exhaustive experience if, for all drug products in the database, he or she has filled a prescription for it. Report the OCP number of every pharmacist with exhaustive experience.

Use only the basic operators $\Pi, \sigma, \bowtie, \times, \cap, \cup, -, \rho$, and assignment.

Solution:

$Should(pharmacist, drug) := \Pi_{pharmacist}(Filled)\Pi_{DIN}(Product)$

$NotAllDrugs := Should - \Pi_{pharmacist,drug}(Filled \bowtie Prescription)$

$AllDrugs := \Pi_{pharmacist}(Filled) - \Pi_{pharmacist}(NotAllDrugs)$

Question 3. [6 marks]

Suppose we have implemented the Twitter schema from Question 1 in SQL, and the tables currently contain the following:

User:

userid	name	email
adele	Adele Laurie Blue Adkins	
drizzy	Drake	
potus	Barack Obama	potus@gov.us
rjm	Renee Miller	rjm@cs

Follows:

a	b
potus	drizzy
drizzy	rjm
drizzy	adele
adele	drizzy

Tweet:

tweetid	userid	content	date
15	adele	Hello	2016-10-16
61	adele	It's me	2016-10-16
33	potus	6 weeks	2016-10-11
28	rjm	in the 6	2016-10-10

Likes:

who	what	when	stars
drizzy	15	2016-10-16	*
drizzy	61	2016-10-18	****
potus	33	2016-09-11	*****

Show the output of each of the following queries.

```
SELECT who
FROM Likes, Tweet
WHERE userID = 'adele';
```

```
SELECT userID, count(a)
FROM User, Follows
WHERE userID = b
GROUP BY userID;
```

```
SELECT count(*) AS num1, count(email) AS num2
FROM User;
```

```
SELECT userID
FROM User NATURAL JOIN Tweet;
```

```
SELECT tweetID, what
FROM Tweet FULL JOIN Likes ON date = when;
```

```
(SELECT date
FROM Tweet)
  EXCEPT
(SELECT when as date
FROM Likes);
```

Solution:

{ (drizzy), (drizzy), (potus) }

{ (drizzy, 2), (adele, 1), (rjm, 1) }

{ (4, 2) }

{ (adele), (adele), (potus), (rjm) }

{ (15, 15), (61, 15) }

{ (2016-10-16), (2016-10-11), (2016-10-10) }

Question 4. [4 MARKS]

Write a query to find the name of everyone with at least 2 followers each of whom (meaning each of these followers) has made at least one tweet. Ensure that your query would work on any instance of the database, not simply the one above.

Solution:

```
SELECT name
FROM User, Follows, Tweet
WHERE User.userID = b and a = Tweet.userID
GROUP BY User.userid
HAVING count(a) >= 2;
```