CS 61A Fall 2019 — Streams, SQL

Guerrilla Section 05: December 7, 2019 Solutions

1 Streams

1.1 What Would Scheme Display?

```
> (define a (cons-stream 4 (cons-stream 6 (cons-stream 8 a))))
> (car a)

4

> (cdr a)

[promise (not forced)]

> (cdr-stream a)

(6 . [promise (not forced)])

> (define b (cons-stream 10 a))
> (cdr b)

[promise (not forced)]

> (cdr-stream b)

(4 . [promise (forced)])

> (define c (cons-stream 3 (cons-stream 6)))
> (cdr-stream c)

Error: too few operands in form
```

1.2 Write a function **merge** that takes in two sorted infinite streams and returns a new infinite stream containing all the elements from both streams, in sorted order.

```
(define (merge s1 s2)





)
```

```
(define (merge s1 s2)
(if (null? s1)
s2
(if (<= (car s1) (car s2))
(cons-stream (car s1) (merge (cdr-stream s1) s2))
(cons-stream (car s2) (merge s1 (cdr-stream s2)))
)
)
)

; Alternate solution
(define (merge s1 s2)
(if (null? s1)
s2
(if (<= (car s1) (car s2))
(cons-stream (car s1) (merge (cdr-stream s1) s2))
(merge s2 s1)
)
)
)
```

2 SQL

2.1 We will be working with a Facebook-like website called Fakebook. There are four tables in our Fakebook data, described below:

people(name, age, state, hobby): a person on Fakebook

posts(post_id, poster, text, time): a post with its creator and creation time (in minutes, starting at 0)

likes(post_id, name, time): a like – post_id of the post that was liked, name of person who liked the post, and time (in minutes) of like

requests(friend1, friend2): a friend request from friend1 to friend2

If you wish to try these questions out on your sqlite3 interpreter, you can download a .sql file with the data we'll be working with can be found here:
https://links.cs61a.org/guer05

Write a query to find the name and age for each person on Fakebook who is 26 years old or younger

```
SELECT name, age FROM people
WHERE age <= 26;
```

2.2 Write a query to find the name of the poster and the time of each post on Fakebook before minute 230

```
SELECT poster, time FROM posts
WHERE time < 230;
```

2.3 Write a query to find the names of users who have liked their own post

```
SELECT poster from posts, likes
WHERE name = poster AND posts.post_id = likes.post_id;
```

2.4 The requests table stores all requests from one person to another. Two people are only friends if both people requested to be friends with the other. Create a table **friends** that has two columns, **(friend1, friend2)**, which contain the names of each friend pairing. For example, if Hali sends a friend request to Joe and Joe sends a friend request to Hali, both Joe—Hali and Hali—Joe should appear in the table.

```
CREATE TABLE friends AS
SELECT a.friend1 as friend1, a.friend2 as friend2
FROM requests AS a, requests AS b
WHERE a.friend1 = b.friend2 AND a.friend2 = b.friend1;
```

2.5 Recall: The aggregate functions MAX, MIN, COUNT, and SUM return the maximum, minimum, number, and sum of the values in a column. The GROUP BY clause of a select statement is used to partition rows into groups. HAVING filters through the groupings from the GROUP BY clause.
Write a query that outputs all names of people who have at least 4 friends

```
SELECT friend1 FROM friends
GROUP BY friend1
HAVING COUNT(*) >= 4;
```

2.6 Write a query that outputs the states that Will's friends live in, and how many friends are in each state

```
SELECT state, COUNT(*)
FROM friends as f, people as p
WHERE f.friend1 = "Will" AND f.friend2 = name
GROUP BY p.state;
```

2.7 Write a query that outputs the text from every post that was liked within 2 minutes of post time

```
SELECT posts.text FROM posts, likes
WHERE posts.post_id = likes.post_id AND likes.time <= posts.time + 2;
```

2.8 Write a query that outputs every pair of people that share the same hobby, as well as that shared hobby. Make sure your output doesn't have duplicate pairs (e.g. A—B and B—A should not both appear in the output).

```
SELECT a.name, b.name, a.hobby
FROM people AS a, people AS b
WHERE a.hobby = b.hobby AND a.name < b.name;
```

2.9 Write a query that outputs the counts of the number of people that live in each state, with each state listed in descending order of count

```
SELECT state, COUNT(*) FROM people
GROUP BY state
ORDER BY -COUNT(*);
```

2.10 Send a friend request by inserting a new friend request from 'Denero' to 'Hilfy'

```
INSERT INTO requests(friend1, friend2) VALUES(Denero, Hilfy);
```

2.11 Help Fakebook user 'Denero' send a friend request to every person who liked post 349 by inserting into requests

```
INSERT INTO requests(friend1, friend2) SELECT Denero, name FROM likes WHERE
post_id = 349;
```

2.12 Change the hobby of every person whose name is Joe to CS

```
UPDATE people SET hobby = CS WHERE name = Joe;
```

2.13 Create a table num_likes with columns: name, post_id, number. Each row should contain a posters name, a post_id, and number of likes for that post

```
CREATE TABLE num_likes AS
SELECT posts.poster AS name, posts.post_id AS post_id,
COUNT(likes.name) AS number
FROM posts, likes
WHERE posts.post_id = likes.post_id
GROUP BY posts.post_id;
```

2.14 Carolyn is a bit shy. Delete all of her posts in the num_likes table with fewer than 4 likes

```
DELETE FROM num_likes WHERE number < 4 AND name = Carolyn;
```

2.15 Create an empty table called privacy with columns name and visibility which should hold the default to everyone.

```
CREATE TABLE privacy(name, visibility DEFAULT everyone);
```

2.16 Add Hermish to privacy using the default value.

```
INSERT INTO privacy(name) VALUES (Hermish);
```