Guerrilla Section 7: Macros, SQL

Instructions

Form a group of 3-4. Start on Question 1. Check off with a staff member when everyone in your group understands how to solve the questions up to the first checkpoint. Repeat for the second checkpoint, the third checkpoint, and so on. **You're not allowed to move on after a checkpoint until you check off with a staff member.** You are allowed to use any and all resources at your disposal, including the interpreter, lecture notes and slides, discussion notes, and labs. You may consult the staff members, **but only after you have asked everyone else in your group. The purpose of this section is to have all the students working together to learn the material.**

SQL

Loading Data: Fakebook

We will be working with a Facebook-like website called Fakebook. The data we will be using will be in fakebook.sql (Google Drive link). Download it and load it in your interpreter with

```
sqlite> .read fakebook.sql
```

OR, if you don't have `sqlite3` installed, you can use an online SQL interpreter to test your solutions. If you're using `sqlite3`, edit your queries in some text editor (e.g. Sublime) and read them in so you can easily change them.

Data Description

There are four tables in the provided Fakebook data, summarized below:

Table Name and Columns	**Table Information Description: Each row represents...**
`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`

Question 1: Fill in the blanks! (Part I)

Fill in the table below with the query that would produce the expected output

Desired Information	**Expected Output**	**Query**

The name and age for each person on Fakebook who is 26 years old or younger	`Hali|25`
	`Jenn|22`
	`Joe|25`
	`Lindsey|24`
	`Rodney|24`
		`SELECT name, age FROM people`
		`WHERE age <= 26;`

The name of the poster and the time of each post on Fakebook before minute 230	`Mike|104`
	`Jenn|124`
	`So|134`
	`Nina|229`
		`SELECT poster, time FROM posts`
		`WHERE time < 230;`

The names of users who have liked their own post	`Mike`
	`Vince`
	`Jenn`
	`Mike`
	`Shirin`
	`Vince`
	`Rodney`
	`Max`
	`Rodney`
	`Mike`
	`Will`
		`SELECT poster from posts, likes`
		`WHERE na`me = poster AND`
		`posts.post_id = likes.post_id;`

Question 2: Friend Requests

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` and `friend2`) that contains 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;
```

If you have created the table correctly, the sample query below should work.

```
      > SELECT * FROM friends WHERE friend1 = "Hali";
      Hali|Jenn
      Hali|Joe
      Hali|Shirin
```

STOP!

Don’t proceed until everyone in your group has finished and understands all exercises in this section, and you have gotten checked off for **Check-in #1**

Question 3: Write More Queries!

Hint: 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.

Desired Information	Expected Output	Query

all names of people who have at least 4 friends	`Carolyn`
	`Kelly`
	`Mike`
	`Tyler`
	`Will`
		`SELECT friend1 FROM friends`
		`GROUP BY friend1`
		`HAVING COUNT(*) >= 4;`

the states that Will's friends live in, and how many friends in each state	`Arizona|1`
	`California|2`
	`Massachusetts|1`
	`Texas|1`
		`SELECT state, COUNT(*)`
		`FROM friends as f, people as p`
		`WHERE f.friend1 = "Will"`
		`AND f.friend2 = name`
		`GROUP BY p.state;`

Text from every post that was liked within 2 minutes of post time	`Scorpions`
	`Winner winner chicken dinner`
	`Snickers`
	`Sandwiches`
		`SELECT posts.text FROM posts, likes`
		`WHERE posts.post_id = likes.post_id`
		`AND likes.time <= posts.time + 2;`

Every pair of people that share the same hobby, as well as that shared hobby.
Make sure your output doesn’t have duplicate pairs
	`Carolyn|Will|karaoke`
	`Dan|Mike|disc golf`
	`Hali|Jenn|surfing`
	`Joaquin|Shirin|traveling`
	`Joaquin|So|traveling`
	`Kelly|Tyler|football`
	`Shirin|So|traveling`
		`SELECT a.name, b.name, a.hobby`
		`FROM people AS a, people AS b`
		`WHERE a.hobby = b.hobby AND a.name < b.name;`

The counts of the number of people that live in each state, with each state listed in descending order of count	`California|9`
	`Florida|2`
	`New York|2`
	`Arizona|1`
	`Maine|1`
	`Massachusetts|1`
	`Texas|1`
	`Utah|1`
		`SELECT state, COUNT(*) FROM people`
		`GROUP BY state`
		`ORDER BY -COUNT(*);`

Question 4: Mutation! Insert stuff! Update stuff! Delete stuff!

Directions	**Query**
Send a friend request by inserting a new friend request from Denero to Hilfy	`INSERT INTO requests(friend1, friend2) VALUES('Denero', 'Hilfy');`
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;`
Change the hobby of every person whose name is Joe to CS	`UPDATE people SET hobby = 'CS' WHERE name = 'Joe';`

Create a table **`num_likes`** with the columns **`name`**, **`post_id`**, **`number`**. Each row should contain a poster's **`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;`

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';`
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');`

Add Hermish to privacy using the default value.	INSERT INTO privacy(name) VALUES ('Hermish');

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section, and you have gotten checked off for **Check-in #2**

Macros

Question 0

What will Scheme output? If you think it errors, write Error.

```
scm> (define-macro (doierror) (/ 1 0))
doierror
scm>(doierror)
Error
scm> (define x 5)
x
scm> (define-macro (evaller y) (list (list 'lambda '(x) x)) y)
evaller
scm> (evaller 2)
2
```

Question 1

Let's try to implement a version of cons-stream and cdr-stream using macros! We'll call them stream-cons and stream-cdr; you should implement them so that they will follow the behavior given by the doctests below.

You do not need to worry about multiple evaluations; in other words, stream-cdr may cause the value to be recomputed (unlike actual streams which the cdr can only be forced / evaluated once). Again: **streams-cdr is allowed to recompute the value**; we're mainly focused on the fact that streams allows to delay evaluation of expressions. In your implementation, you may not use cons-stream or cdr-stream.

Hint: In most cases, e.g. with expressions like (cons 1 (/ 1 0)) or (define x (print 2)), we evaluate an entire expression immediately, violating the properties of lazy evaluation that a stream uses. What's one special form or function that we've learned before macros which can delay the evaluation of an expression?

```
scm> (define (naturals-from n) (stream-cons n (naturals-from (+ n
1))))
naturals-from
scm> (define naturals (naturals-from 0))
naturals
scm> (car (stream-cdr (stream-cdr (stream-cdr (stream-cdr
naturals)))))
4

(define-macro (stream-cons x xs)
    `(cons ,x (lambda () ,xs)))

(define (stream-cdr xs)
    ((cdr xs)))
```

Question 2

Define a macro while that processes a while loop by converting it to a tail recursive function
The goal of this question is to define a macro that represents a while loop. Since this is a difficult task we will break it into parts.

2a

Write tail-recursive factorial:

```
(define (fact n)
      (define (fact-tail n result)
            (if (= n 0)
            result
            (fact-tail (- n 1) (* n result))))
      (fact-tail n 1)
  )
```

2b

Using the above problem to assist implementation, create the while macro. This macro will accept 4 arguments:

- initial-bindings: this will represent initialization values for variables in the loop
- condition: this will represent the condition which the while loop should continue to check to see if the loop should continue
- return: after the loop has ended this represents the value that should be returned

You may find the built-in map function useful for this problem:

```
scm > (map (lambda (x) (* 2 x)) '(1 2 3))
(2 4 6)
```

And here's an example of the while macro being used to calculate the factorial:

```
scm > (define (fact n)
      (while
            ((acc 1) (n n))
            (> n 0)
            ((* acc n) (- n 1))
           acc))
fact
scm> (fact 4)
24
```

Fill in the following macro definition:

```
(define-macro (while initial-bindings condition updates return)
    (define helper-vars (map car initial-bindings))
    (define initial-vals (map (car (cdr initial-bindings))))
    (list 'begin
        (list 'define (cons 'helper helper-vars)
            `(if ,condition
                ,(cons 'helper updates)
                ,return))
        (cons 'helper initial-vals)))
```

CONGRATULATIONS!

You made it to the end of the worksheet! Great work :)