**CSE 132A** Winter 2023
Assignment #1 (SQL)
Due February 6 (11:59pm)

*This is an* **individual** *assignment. Please review the academic integrity policy and contact the instructor or TA if you have any questions.*

The beer drinker's database has the following schema:

*frequents: drinker (string), bar (string)*
*serves: bar (string), beer (string), price (float)*
*likes: drinker (string), beer (string)*

The first relation indicates the bars each drinker frequents, the second tells what beers each bar serves and their price (in integer dollars), and the last indicates the beers that drinkers like. Assume that every drinker frequents at least one bar, every bar serves at least one beer, and every beer is served by some bar. However, there may be drinkers who do not like any beer, beers that are not liked by any drinker, and bars who are not frequented by any drinker. Assume that the database contains no nulls or duplicates.

In addition, the following constraints hold:

- $\{drinker, bar\}$ is the primary key of *frequents*
- $\{bar, beer\}$ is the primary key of *serves* (so a beer has just one price at a given bar)
- $\{drinker, beer\}$ is the primary key of *likes.*

A sample instance will be posted separately, together with outputs to the queries in the assignment.

**Write the following queries in SQL** (of course, the queries must work on all data, not just the sample one). Use only constructs covered in class so far (for example, do **not** use `case` statements, joins, or limit).

1. Find the bars that serve some beer that Molly likes, for strictly less than 3 dollars. The answer should have a single attribute *bar*.

2. List all pairs of drinkers for which there is at least one bar they both frequent, that serves some beer they both like. The answer should have attributes {*drinker1, drinker2*}. Avoid listing pairs of the form $< a, a >$, or listing both $< a, b >$ and $< b, a >$. Specifically, list only pairs $< a, b >$ where $a$ is strictly less than $b$ in alphabetical order.

3. Find the bars with the lowest average price of beers served at the bar. The answer should have a single attribute $\{bar\}$.

4. List, for each bar, the number of beers that the bar serves that are liked by all drinkers frequenting the bar. The answer should have attributes *bar, num*. Note that the value of *num* should be 0 for bars that serve no beer liked by all drinkers frequenting the bar.

5. For each beer that occurs in *serves*, find its average price and its popularity. The popularity of a beer is the number of drinkers who like the beer (if no drinker likes the beer, its popularity is zero). The answer should have attributes $\{beer, avg\_price, popularity\}$.

6. Find the bars that serve every beer Joe likes at the lowest price. The answer should have a single attribute *bar*. Note that, if Joe does not like any beer, all bars should be in the answer. Write an SQL query using nested sub-queries with NOT EXISTS tests only.

7. Update relation *likes* by swapping the drinkers who like Coors and the drinkers who like Bass, without explicitly naming the drinkers involved. Assume that all beer names are alphanumeric (consist of letters and/or numbers). You may use several update commands if needed. Do not create new relations.

**What to submit:**

- Submit a separate file for each problem, named accordingly (Q1.sql,..., Q7.sql).
- Each file should be formatted such that calling the command ".read Q*.sql" on a populated database in sqlite3 would give you the desired result (you can test this during submission).
- For each file, please only fill in the SQL statements needed for that problem. Do not include the statements to create the database schema or the database input.

**Submission instructions:**

- Zip the required files and upload to gradescope. Make sure to zip all the files together rather than just the parent directory.
- Once you submit your files on gradescope, the autograder will run your submission against some test cases. A working submission (i.e. correctly formatted, compressed) should pass the public test cases.

**Grading:**

- The assignment is graded based on correctness of the queries against some public and hidden test cases.
- The public test cases are derived from the posted sample database, and are visible for your testing purposes.
- The hidden test cases are derived from a different instance of the database, and won't be visible to you. Therefore, it's important to make sure the SQL statements work for all databases.