# Assignment #6
## Due Tuesday November 21, 5:00 PM

### Problems to Work and Hand In

The following questions relate to the "Beer Drinkers' Relational Database Schema" below. It has been augmented with some additional attributes to make things more interesting. We may suppose that the "rating" a drinker gives to a beer is a real number reflecting the degree of approval of that beer; the price charged by a bar for a glass of beer is another real number with the obvious meaning.

```
Likes(drinker, beer, rating)
Serves(bar, beer, price)
Frequents(drinker, bar)
LocatedAt(bar, addr)
```

1. (12 pts.) Write the following constraints either as the containment of two relational-algebra expressions or as the equation of a relational-algebra expression to $\emptyset$.

   (a) Every bar that any drinker frequents must appear as a bar in the `Serves` relation.

   (b) No bar may sell Corona beer for more than it sells Budwieser.

   (c) Two bars may not be located at the same address.

   (d) If some drinker likes a beer, and a bar serves that beer, then the drinker must frequent that bar.

2. (24 pts.) Write SQL queries for the following:

   (a) Find all drinkers who frequent a bar that serves Coors.

   (b) Find all drinkers that frequent a bar that serves a beer that drinker `I.M.Potted` likes.

   (c) Find all pairs of beers that are served at one bar. Do not include pairs consisting of one beer mentioned twice, and do not list the same pair of beers twice, even in different orders. That is, do not list $(b, b)$, and if you list $(b_1, b_2)$, do not list it again or list $(b_2, b_1)$ in your output.

   (d) Find the average rating of Anchor Steam beer. You may assume that there are no duplicates in the `Likes` relation.

   (e) Create a table giving for each beer its average rating. Again assume no duplicates in `Likes`.

   (f) Create a table that gives for each beer the maximum price charged.

   (g) Create a table that gives for each beer the bar(s) charging the least for that beer.

   (h) Create a table that gives for each beer the drinkers that give that beer a rating of at least 0.8 and also frequent at least one bar that serves that beer.

3. (9 pts.) Write the following database modifications in SQL.

   (a) Add the fact that drinker `N.E.Briated` Likes Pete's Wicked Ale (no rating is known).

   (b) If a bar serves a beer that at least one drinker who frequents the bar likes, then double the price of the beer at that bar.

   (c) Delete all `Serves` tuples for beers that no one likes.

## Sixth Step of Your Personal Database Application

1. (40 pts.) In this problem you will finally have the pleasure of running queries over the data in your Personal Database Application (PDA). Develop and test:

   (a) At least five SQL data retrieval commands.

   (b) At least one each of the four types of data modification commands (`INSERT` a single tuple, `INSERT` a subquery, `DELETE`, `UPDATE`).

   There are a number of important things to keep in mind for this problem:

   - For this assignment you will be invoking SQL commands interactively, using the `isql` command described in Handout 10: *Getting Started with Sybase*. Of course you should certainly use a script (file), rather than typing the queries in each time you run them.
   - Please write "interesting" queries. You should try to use most or all of the SQL constructs discussed in class and in the reader (subqueries, aggregates, etc.). You will not receive full credit if your queries, modifications, or views are extremely simple.
   - It is strongly suggested that you experiment with your SQL commands on a small database (e.g., the database you hand-created for Assignment #4), before running them on the big database you generated for Assignment #5. Initial debugging is much easier when you're operating on small amounts of data. Once you're confident that your commands are working, run them on your complete database.
   - Turn in a copy of all of your SQL commands, along with a script illustrating their execution. Your script should be sufficient to convince us that your commands run successfully. In particular, you should not simply hand in a copy of your commands and a separate copy of the output. Please do not, however, turn in query results that are thousands (or hundreds of thousands) of lines long!

2. (15 pts.) In part 1 you probably discovered that some queries run very slowly over your large database. The principal technique for improving the performance of queries is to create *indexes*. An *index* on an attribute $A$ of relation $R$ allows the database to quickly find all tuples in $R$ with a given value for attribute $A$. An index can be created on any attribute of any relation. (Often, but not always, indexes are created on key attributes.) The syntax for creating an index in Sybase is:

   `CREATE INDEX` *index-name* `ON` *relation-name* (*attribute-name*)

   If you know that the attribute is a key, you can speed things up even more by declaring:

   `CREATE UNIQUE INDEX` *index-name* `ON` *relation-name* (*attribute-name*)

   You can get rid of an index using the command: `DROP INDEX` *index-name*.

   Create at least two useful indices for your PDA. Run your queries from part 1 on your large database with the indices and without the indices. Note that each time you run an SQL command in Sybase, afterwards you get statistics about how long it took to execute the command. Naturally these times may be affected by external factors such as system load, etc. Still, you should see a dramatic difference between the execution times with indices and the times without. Turn in a script showing your commands to create indices, and showing the relative times of query execution with and without indices.