CPS 116 Fall 2004
Homework #2 (8.75% of course grade: 110 points)
Assigned: Thursday, September 16
Due: Tuesday, September 28

Problem 1 (15 points).

Complete the Gradiance homework titled "Homework 2.1 (SQL Basics)."

Problem 2 (50 points).

Consider again the beer drinker's database from Homework #1. Key columns are underlined.

Drinker (*name*, *address*), *Bar* (*name*, *address*), *Beer* (*name*, *brewer*),
Frequents (*drinker*, *bar*, *times_a_week*), *Likes* (*drinker*, *beer*), *Serves* (*bar*, *beer*, *price*).

Run `/home/dbcourse/examples/db-beers/setup.sh` to setup a database with some sample data. For the SQL database schema, please refer to the file `create.sql` in the same directory. Write SQL statements to answer the following queries.

Write all your queries in a file named `hw2-2.sql`. When you are done, run "`db2 -tf hw2-2.sql > hw2-2.out`" (you may need to run "`db2 connect to dbcourse`" before that and "`db2 disconnect all`" afterwards). Then, print out files `hw2-2.sql` and `hw2-2.out` and turn them in together with the rest of the assignment.

(a) Find all drinkers who frequent James Joyce Pub.
(b) Find all bars that serve both Amstel and Corona.
(c) Find all bars that serve at least one of the beers Amy likes for no more than $2.50.
(d) For each bar, find all beers served at this bar that are liked by none of the drinkers who frequent that bar.
(e) Find all drinkers who frequent *only* those bars that serve some beers they like.
(f) Find all drinkers who frequent *every* bar that serves some beers they like.
(g) Find those drinkers who enjoy exactly the same set of beers as Amy.
(h) For each beer, find the bars that serve it at the lowest price.
(i) For each beer, find its average price and popularity (measured by the number of drinkers who like it). Sort the output by average price.
(j) Every time when Dan goes to a bar, he buys a bottle of the most expensive beer he likes that is served at this bar. If there is more than one such beer, he buys just one of them. If the bar does not serve any beer he likes, he will not buy any beer. Find the amount of money Dan spends every week buying beers in bars.

Problem 3 (15 points).

Assume that in relational algebra, you can use built-in SQL predicates on strings, times, etc. in selection and join conditions; however, no SQL aggregation functions are allowed. Consider Parts (g)-(j) of Problem 2:

(a) Which queries *cannot* be formulated in relational algebra?
(b) For each query that can be formulated in relational algebra, show the equivalent relational algebra query.

Problem 4 (14 points).

Below is the basic design for a used-car sales database. Key columns are underlined. Each automobile has a VIN (vehicle identification number), a model (e.g., Camero), a make (e.g., Chevrolet), a year (e.g., 1999), a color (e.g., red), a mileage (e.g., 50,000 miles), and a body style (e.g., coupe). Each automobile has a seller, which may be either a dealer or an individual. For each dealer, the database stores name, address, phone number. For each individual, only phone number and email address are recorded.

Automobile (*VIN*, *model*, *make*, *year*, *color*, *mileage*, *body_style*, *sellerID*)
Dealer (*sellerID*, *name*, *address*, *phone*)
IndividualSeller (*sellerID*, *phone*, *email*)

Keep all SQL statements you write for this problem in a file named `hw2-4.sql`. For this problem, you should use "`@`" instead of "`;`" as the statement termination character because of the triggers you are going to write in Part (b). When you are done, run "`db2 –td@ -f hw2-4.sql > hw2-4.out`". Then, print out files `hw2-4.sql` and `hw2-4.out` and turn them in together with the rest of the assignment.

(a) Create the schema according to the given basic design, using `CREATE TABLE` statements. Choose appropriate data types for your columns, and remember to declare any keys, foreign keys, `NOT NULL`, and `CHECK` constraints when appropriate.

(b) Note that any *Automobile.sellerID* must be a *Dealer.sellerID* or *IndividualSeller.sellerID*. Also, a *Dealer.sellerID* cannot be an *IndividualSeller.sellerID*, and vice versa. It is not possible to declare these constraints as straightforward key and foreign key constraints. Instead, write triggers to reject any database modification that could violate these constraints.

Syntax for creating triggers in DB2 differs slightly from the standard SQL syntax presented in lecture. For details, please refer to `http://www.cs.duke.edu/courses/fall04/cps116/faqs/sql.html`.

(c) Start with empty tables. Write `INSERT`, `UPDATE`, and `DELETE` statements to illustrate that the triggers you wrote for (b) are working properly. More specifically:
   1) The first statement should attempt to insert a row into *Automobile* but should be rejected by your triggers.
   2) The second statement should insert a row into *Dealer* successfully.
   3) The third statement should attempt to insert a row into *IndividualSeller* but should be rejected by your triggers.
   4) The fourth statement should insert a row into *IndividualSeller* successfully.

(Continue on the next page.)

5) The fifth statement should insert a row into *Automobile* (with *sellerID* referring to a *Dealer*) successfully.
6) The sixth statement should update the *Automobile* row's *sellerID* to refer to an *IndividualSeller* successfully.
7) The seventh statement should attempt to update the *Automboile* row's *sellerID* but should be rejected.
8) The eighth statement should attempt to delete the *IndividualSeller* but should be rejected.
9) The ninth statement should delete the *Automobile* row successfully.
10) The tenth statement should delete the *IndividualSeller* row successfully.
11) The eleventh statement should delete the *Dealer* row successfully.

Problem 5 (16 points).

Complete the Gradiance homework titled "Homework 2.5 (SQL Advanced Features)."