CompSci 316 Fall 2013: Homework #2

100 points (8.75% of course grade) + 10 points extra credit
Assigned: Tuesday, September 17
Due: Thursday, October 3

This homework should be done in parts as soon as relevant topics are covered in lectures. If you wait until the last minute, you might be overwhelmed.

For Problems 1, 2, 4, 6, and 7, you will need to use Gradiance. Please read http://www.cs.duke.edu/courses/fall13/compsci316/faq-gradiance.html for detailed information. There is no need to turn in anything else for these problems; your scores will be tracked automatically.

For the remaining problems, please submit your solutions using the course submission website at https://www.cs.duke.edu/csed/websubmit/app/. For Problems 3 and 5, submit the required files as instructed. For the other problems, submit text or PDF files with informative names; if your solution is handwritten, you may submit a photo or scanned image.

Note that Problems 3 and 5 require the virtual machine for the course on your personal computer. See http://www.cs.duke.edu/courses/fall13/compsci316/faq-vm.html for instructions.

Problem 1 (4 points)

Complete the Gradiance homework titled "Homework 2.1 (Relational Design Theory: MVD)."

Problem 2 (12 points)

Complete the Gradiance homework titled "Homework 2.2 (SQL Querying)."

Problem 3 (36 points)

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

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

Write the following queries in SQL. Log into your virtual machine, and then type "`psql`" at a command-line prompt to run PostgreSQL interpreter. For additional instructions, see http://www.cs.duke.edu/courses/fall13/compsci316/faq-psql.html. Record all your queries in a file named `beers.sql` (and use "`--`" to add comments in the file to indicate which problems they correspond to). When you are all done, run

```
    psql -af beers.sql &> beers.txt
```

to verify that everything works and to generate the final answers. Submit the files `beers.sql` and `beers.txt` through the submission website. If you cannot get a query to parse correctly or return the right answer, include your best attempt and explain it in comments, to earn possible partial credit.

(a) Find all beers served at *Satisfaction*.
(b) Find the names and addresses of bars frequented by *Amy*.
(c) Find the names of bars serving at least one beer liked by *Amy* for no more than $2.50.
(d) Find the names of bars frequented by both *Ben* and *Dan*.
(e) Find the names of bars frequented by either *Ben* or *Dan*, but not both.
(f) For each bar, find all beers served there that are liked by no one who frequents there.
(g) Find names of all drinkers who frequent ***only*** those bars that serve some beers they like.
(h) Find names of all drinkers who frequent ***every*** bar that serves some beers they like.
(i) Find, for each drinker, the bar(s) he or she frequents the most. The output should be list of (drinker, bar) pairs. If some drinker *D* does not frequent any bar, you should still output (*D*, `NULL`).
(j) For each bar, find the total number of drinkers who frequent it, as well as the average price of beers it serves. Sort the output by the number of drinkers (in descending order).

Problem 4 (8 points)

Complete the Gradiance homework titled "Homework 2.4 (SQL Constraints)."

Problem 5 (30 points)

Consider the following relational schema designed for startups in Problem 4 of Homework #1 (http://www.cs.duke.edu/courses/fall13/compsci316/duke-only/assignments/hw1.pdf). See the sample solution for additional explanation for this design.

VCFund (*vcid*, *name*, *number*, *size*, *closing_date*)
Industry (*name*, *market_size*)
Sector (*industry_name*, *sector_name*, *projected_growth*)
Startup (*sid*, *industry_name*, *startup_name*, *address*)
StealthCompany (*sid*, *buzz_factor*)
PrivateCompany (*sid*, *CEO*, *website*, *sector_name*)
Fund (*vcid*, *sid*)
TargetOf (*target_sid*, *sid*)

Your job is to write SQL `CREATE TABLE` and `CREATE TRIGGER` statements to create the above schema and enforce all constraints as described in Problem 4 of Homework #1. We have prepared a template file `startups.sql` for you—download it from https://www.cs.duke.edu/courses/fall13/compsci316/duke-only/assignments/hw2/startups.sql.

This file is incomplete; you need to edit it to fill in the missing parts. Use simple SQL constructs as much as possible, and only those supported by PostgreSQL. Note that:

- PostgreSQL does not allow subqueries in `CHECK`.
- PostgreSQL does not support `CREATE ASSERTION`.
- PostgreSQL's implementation of triggers deviates slightly from the standard. In particular, you will need to define a "UDF" (user-defined function) to execute as the trigger body. For syntax and examples, see http://www.postgresql.org/docs/9.1/static/plpgsql-trigger.html.

Your job involves the following main tasks:

(a) Edit the `CREATE TABLE` statements to enforce constrains.
(b) Note that `startups.sql` defines two additional views, `StealthStartup` and `PrivateStartup`, which "mediate" accesses to `Startup`, `StealthCompany`, and `PrivateCompany`. With help of triggers, these views also simplify constraint enforcement. `StealthStartup` and its trigger have already been implemented; you need to complete `PrivateStartup` and its trigger.
(c) If needed, add additional triggers to enforce other constraints.
(d) Verify that the section of `startups.sql` with data modification statements works as intended. If you enforce all constraints correctly, a number of statements should be accepted and others should be automatically rejected (see comments in the file).
(e) Complete the last part of `startups.sql`—for each constraint listed therein, write one SQL modification statement (`INSERT`, `DELETE`, or `UPDATE`) that would violate the constraint and therefore should be automatically rejected.

When you are all done, run

```
psql -af startups.sql &> startups.txt
```

to verify that everything works and to generate the final answers. Submit the files `startups.sql` and `startups.txt` through the submission website. If you cannot get a statement to work correctly, include your best attempt and explain it in comments, to earn possible partial credit.

Problem 6 (4 points)

Complete the Gradiance homework titled "Homework 2.6 (SQL Recursion)."

Problem 7 (6 points)

Complete the Gradiance homework titled "Homework 2.7 (SQL Triggers, Views)."

Extra Credit Problem X1 (10 points)

Consider the following "bag" algebra, a cousin of relational algebra that works on bags. We give (most of) its semantics below in terms of equivalent SQL queries:

- Bag selection $\sigma_{cond}\, R$: Same as
  `SELECT * FROM R WHERE cond;`
- Bag projection (duplicate-preserving) $\pi_{column\_list}\, R$: Same as
  `SELECT column_list FROM R;`
- Duplicate elimination δR: Same as
  `SELECT DISTINCT * FROM R;`
- Bag cross product $R \times S$: Same as
  `SELECT * FROM R, S;`
- Bag union $R \uplus S$: Same as
  `(SELECT * FROM R)UNION ALL(SELECT * FROM S);`
- Bag difference $R \dot{-} S$: Same as
  `(SELECT * FROM R)EXCEPT ALL(SELECT * FROM S);`
- Bag rename $\rho_{S(A_1,A_2,\ldots)}R$: Same as the relational rename; preserves original bag content.

Suppose we have a table $R(A)$ (with just a single integer column A).

(a) Write a bag algebra query that finds the ***median*** of all $R.A$ values, i.e., the value that appears right in the center of the list when you sort all $R.A$ values. For this problem, you may assume that R contains an odd number of rows and all $R.A$ values are distinct. For example, the median of the set $\{1, 4, 2, 8, 5, 7, 11\}$ is 5, because it is right in the center of the sorted list $(1, 2, 4, 5, 7, 8, 11)$.

(b) Write a bag algebra query that finds (the cutoff for) the ***upper quartile*** of $R.A$ values, i.e., the value that is greater than 75% of all $R.A$ values and less than or equal to 25% of them. For simplicity, you may assume that all $R.A$ values are distinct, and that the number of rows is an exact multiple of 4. For example, the upper quartile of the set $\{1, 4, 2, 8, 5, 7, 11, 3\}$ is 8, which can be easily seen by sorting the set to $(1, 2, 3, 4, 5, 7, 8, 11)$.