CompSci 316 Fall 2017: Homework #1

100 points (8.75% of course grade) + 10 points extra credit
Assigned: Tuesday, September 5
Due: Tuesday, September 19

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, 3, and 5, you will need to use Gradiance. Gradiance is an online testing system that provides immediate feedback to your answers, and allows you to retry a problem multiple times until you get it right. Please read the "Help → Getting Started with Gradiance" section of the course website for instructions on how to get started. There is no need to turn in anything else for these problems; your scores will be tracked automatically.

For other problems, you will need to turn in the required files electronically. Please read the "Help → Submitting Non-Gradiance Work" section of the course website for instructions. For Problems 4 and X1, you may prepare your answers electronically or on paper (handwritten). In the latter case, scan or photograph the pages and submit the resulting PDF (preferred) or JPG files. Please name your files informatively, e.g., as `n.pdf`, where `n` is the problem number.

Problem 2 should be completed on a virtual machine (VM). Please read the VM-related section under "Help" on the course website, and follow the instructions therein to get your VM running. (You should have received an email from the instructor by now concerning Google Cloud credits, if you opt to run your VM on Google Cloud.)

Problem 1 (13 points)

Complete the Gradiance homework titled "Homework 1.1 (Relational Algebra Basics)." Note that some of the online exercises use English names of relational algebra operators instead of symbols.

Problem 2 (32 points)

Consider a database containing information about bars, beers, and bar-goers.

drinker(*name*, *address*)
bar(*name*, *address*)
beer(*name*, *brewer*)
frequents(*drinker*, *bar*, *times_a_week*)
likes(*drinker*, *beer*)
serves(*bar*, *beer*, *price*)

Write the following queries in relational algebra using RA, our homegrown relational algebra interpreter. To set up the sample database called `beers`, issue this command in your VM shell:

```
/opt/dbcourse/examples/db-beers/setup.sh
```

Then, type "`radb beers`" to run RA. See "Help → RA: A Relational Algebra Interpreter" on the course website for additional instructions on using RA, including syntax of relational operators.

When you run RA, as soon as you get a working solution, record the query in a plain-text file named `2-queries.txt` (use Java/C++-style comments in the file to indicate which problem each query corresponds to). When you are done with all queries, run

```
radb beers -i 2-queries.txt -eo 2-answers.txt
```

to generate the final answers. Submit the files `2-queries.txt` and `2-answers.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 the names of bars serving *Corona*.
(b) Find the beers served at those bars that Ben frequents only once a week. (Just list the beers, not the bars.)
(c) Find names and addresses of drinkers who like some beer served at *Talk of the Town*.
(d) Find pairs of drinkers who live at the same address. (Just list the drinker names. Don't list (*drinkerA*, *drinkerA*). If you list (*drinkerA*, *drinkerB*) in the answer, don't list (*drinkerB*, *drinkerA*) again.)
(e) Find beers that are liked by nobody.
(f) For each beer, find the bar that serves it at the highest price. (Your output should be a list of (*beer*, *bar*) pairs. If multiple bars tie for the most expensive for a beer, list them all as different pairs.)
(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.

Problem 3 (15 points)

Complete the Gradiance homework titled "Homework 1.3 (E/R Design)."

Problem 4 (20 points)

It's time that wizardry embraces technology! You are commissioned by the Ministry of Magic to design a database for the Hogwarts School of Witchcraft and Wizardry. Your database should capture the following information:

- The database only needs to record information about current students and teachers. Therefore, you may assume that each person is either a student or teacher, but not both.
- Each person has a unique id assigned by the School, a name, a pet, and a wand. The Ministry is interested in keeping only a short description of one's current pet (e.g., *Hedwig the Snowy Owl*) and the core of one's wand (e.g., *phoenix feather*). You may assume that all wands are single-core.
- The School offers different subjects of study. Each subject has a unique name. In any given school year, a given subject can be taught by only one teacher (e.g., *Severus Snape* taught *Defense Against the Dark Arts* during 1996-97). Each student studying this subject receives a grade at the end of the school year. A subject can be taught over multiple years and a student may study the same subject multiple times.
- For each student, you need to additionally record the year when he or she entered the School, as well as his or her favorite subjects.

- Students are divided into Houses with unique names (e.g., *Gryffindor* and *Slytherin*). You need to track each House's head teacher (e.g., *Gryffindor*'s head is *Minerva McGonagall*) and its current students.
- Students earn points for their Houses: points are rewarded for good deeds, winning a Quidditch match, etc.; they are taken away for breaking rules. The Ministry wants to you to track the points earned and lost by each deed of each student over time. From your database, it should be possible to, for example, tally the points accumulated by each House since the beginning of the school year, calculate each student's contribution during any time period, and look up a particular deed.

If any of the above deviates from the "canon," use the assumptions above (with apologies to Harry Potter fans). If you think some aspects of the above are unclear, feel free to make additional, reasonable assumptions, but state them clearly in your answer. Also, keep in mind that there is no single "correct" design; if you think you are making a non-obvious design decision, please explain it briefly.

(a) Design an E/R diagram for this database. Very briefly explain the intuitive meaning of any entity and relationship sets as needed. Do not forget to indicate keys and multiplicity of relationships, as well as ISA relationships and weak entity sets (if any), using appropriate notation.
(b) Design a relational schema for this database. (You can start by translating the E/R design.) You may ignore attribute types, and you do not need to show any sample data. Indicate all keys and non-trivial functional dependencies in the schema. Check if the schema is in BCNF. If not, decompose the schema into BCNF.

Problem 5 (20 points)

Complete the Gradiance homework titled "Homework 1.5 (Relational Design Theory: FD)."

Extra Credit Problem X1 (5 points)

As discussed in class, the core operators in relational algebra are selection (σ_p), projection (π_L), cross product ($\times$), union ($\cup$), and difference ($-$). Prove that the selection operator is necessary; that is, some queries that use this operator cannot be expressed using any combination of the other operators.

Hint: We are looking for a rigorous proof, not just intuition. For example, consider the argument that the union operator is necessary. The intuition is that union the only operator that lets you "add" tuples to a relation without widening it, but this argument alone is not rigorous, because one could use cross product to create more tuples and then use projection to get a narrower relation. A rigorous argument would be the following. Consider a database instance with two single-attribute relations $R = \{\langle 0 \rangle\}$ *and* $S = \{\langle 1 \rangle\}$. $R \cup S$ *would yield a relation with two tuples, but starting with* R *and* S*, repeated uses of other operators can never obtain a result relation with more than one tuple (using an inductive argument).*

Extra Credit Problem X2 (5 points)

A parity query returns the empty relation $\emptyset$ when the input relation has an odd number of tuples, or otherwise the singleton relation $\{\langle\rangle\}$ (whose only tuple has no attributes, which can be created by projecting any non-empty relation onto an empty set of attributes). Prove that the parity query over a relation $R(A)$ cannot be written in relational algebra, where selection and join conditions are restricted to comparing attributes and/or constants using $=$ and $\neq$, and projections are restricted to subsets of input attributes.