CS 4320/5320 Homework 1

Fall 2015

Due September 16th, 2015

This assignment is due on Wednesday, September 16th at 11:59pm. It is out of 80 points and counts for 10% of your overall grade.

Before you start this assignment, make sure you have **read the course policies document (available in CMS) and completed the CMS quiz on the course policies. You must complete this quiz to pass the course.** We assume you are familiar with the course policies. Therefore, we don't repeat info about group work, late submissions, academic integrity etc in the homework instructions.

To complete the code portions of this assignment, you will need to install both MySQL (`http://dev.mysql.com/downloads/mysql/`) and PostgreSQL (`http://www.postgresql.org/download/`). By default we will grade your code on the current release (stable) versions of the respective platforms.

Using the right DBMS: Question 1 will be graded on MySQL, question 2 on PostgreSQL. Please be sure your queries run on the correct system. In particular, if you write queries for Question 1 that use PostgreSQL-only functionality, you are disregarding explicit instructions and will get zero points; you will not receive any consideration under our breaking automation policy.

1 SQL and Relational Algebra queries (35 points)

Consider the schema given by the following SQL `CREATE` statements. (We will provide these statements to you as a .txt file so you can avoid any typos while pasting them in.)

```
CREATE TABLE Student (sid INTEGER PRIMARY KEY,
                      sname VARCHAR(20));
CREATE TABLE CourseOffering (cid INTEGER NOT NULL,
                             semesterid INTEGER NOT NULL,
                             profid INTEGER,
                             roomid INTEGER,
                             PRIMARY key (cid, semesterid));
CREATE TABLE Enrolled (sid INTEGER NOT NULL,
                       cid INTEGER NOT NULL,
                       semesterid INTEGER NOT NULL,
                       grade INTEGER,
                       PRIMARY KEY (sid, cid, semesterid));
```

This schema describes a simple university setting. The Student table keeps track of students; each student has a student id (sid) and a name. The CourseOffering table keeps track of course offerings – a course offering has a course id (cid), semester id and associated info such as the profid of the professor teaching the course and the roomid of the classroom where the course meets. Finally, the Enrolled table keeps track of which students are enrolled in which courses in a given semester, and each (sid, cid, semesterid, grade) triple states that student *sid* takes course *cid* in semester *semesterid* and receives grade *grade* (for simplicity we represent grades as integers). It is possible that a course has no offerings, that a student enrolls in no classes, or that a course offering in a given semester has no enrollments.

Write the following queries in **SQL**. Each of your answers must be a single SQL query. Your SQL queries will be graded by running them in MySQL on test cases; queries that return the wrong answer will receive 0 points, i.e. there is no partial credit. In particular, please make sure your query returns the right column(s).

(a) (5 points) **For each course, find the course offering with the highest enrollment.** Display (cid, semesterid) pairs. That is, if you return $(4320, 10)$ that means that course 4320 had its highest enrollment when it was offered in semester number 10. If multiple course offerings have the same highest enrollment, return all appropriate (cid, semester) pairs. If there is a course with the property that no students ever enrolled in any offering of this course, you may ignore that course (i.e. not include it in the answer in any way).

(b) (5 points) **Find all pairs of students who have taken NO courses in common** - not just in a given semester, but who never ever enrolled in a course with the same course number. For every such pair, output their sids. Note that each pair of students should appear only once in the result, so if $(1, 3)$ is in the result $(3, 1)$ should not be.

(c) (5 points) **Find all pairs of students who have taken the EXACT SAME SET of courses in their career.** That is, if X is the set of all courses that student 1 has ever enrolled in, Y is the set of all courses that student 2 has ever enrolled in, and X and Y are identical, return $(1, 2)$. Semesters do not matter for this question - i.e. the students do not need to have taken the same course in the same semester. For every qualifying pair of students, output their sids, with no duplicates like in (b).

(d) (5 points) **Find all students who have received the highest (or highest-equal) grade for every course (offering) they have ever taken.** That is, every time they took a course they were the top student. Return the sids of such students with no duplicates. If a student has enrolled in no courses at all they should not be included; but if they have taken a course multiple times they need to be the top student in all offerings which they have taken.

Write the following queries in **relational algebra**. To enhance readability, you are encouraged to split off subexpressions and name them using the ρ operator. You must use only the operators introduced in lectures, although you may define “shorthand” operators that can be expressed in terms of the basic ones as long as you explain how they can be expressed. You may NOT use the extended notation introduced in Chapter 15 of your textbook (15.1.2) that uses operators such as MAX/MIN, GROUP BY and HAVING. (Seriously - if you use one of these operators, you will get zero points.)

(e) (5 points) the same query as part (b) above

(f) (5 points) the same query as part (c) above

(g) (5 points) the same query as part (d) above

2 Constraints and Triggers (15 points)

This question has you write some constraints and triggers in PostgreSQL. We start with the following database schema, which we also provide for you in a .txt file so you can copy/paste it into your system:

```
CREATE TABLE Ships (name VARCHAR(20) PRIMARY KEY,
                    launched DATE);

CREATE TABLE Battles (name VARCHAR(20) PRIMARY KEY,
                      battledate DATE);

CREATE TABLE Outcomes (shipname VARCHAR(20),
                       battlename VARCHAR(20),
                       result VARCHAR(20),
                       FOREIGN KEY (shipname) REFERENCES Ships (name),
                       FOREIGN KEY (battlename) REFERENCES Battles (name),
                       PRIMARY KEY (shipname, battlename));
```

These tables contain historical information about battleships. Every ship has a name and date it was launched - this is found in the Ships table. Every battle has a name and a date, in the Battles table. When a ship fights in a battle, the result is recorded in the Outcomes table.

(a) (4 points) Add a `CHECK` constraint to the table to enforce that the result attribute in Outcomes must have value either 'ok', 'damaged', or 'sunk' (without the quotes). That is, provide an `ALTER` statement that we can paste into the console, so that if we subsequently try to insert values with invalid "result" attributes, the insert will be rejected, but inserts with valid "result" attributes will be allowed.

(b) (3 points) No ship can be in battle before it is launched. Express this as a `CHECK` clause. That is, again give an `ALTER` statement on the `Outcomes` table that would enforce this constraint. Note that your statement will not actually run because PostgreSQL does not allow nested subqueries in `CHECK` clauses, so we will grade this just by looking at your code.

(c) (8 points) Now that we have established `CHECK` functionality is not sufficient for what we want, use triggers to enforce our constraint (no ship should be in battle before it is launched). Provide code that we can paste into the console – we will paste it in and then attempt to make various inserts/deletes/updates to produce a database where a ship is in battle before its launch date. Think about all possible "evil" things we could do and handle them all. Of course, all modifications that do respect this constraint (and other constraints defined above such as primary keys) should be allowed.

We will grade parts a) and c) by running your code literally as described.

3 Set and Bag Relational Algebra (30 points)

Consider each of the following equivalences. Some of them hold in both set and bag relational algebra, while others hold only in set relational algebra. For each equivalence below,

- state whether it holds for both set and bag RA or only for set RA
- if it holds for both sets and bags, prove that fact
- if it holds only for sets, prove that it holds for sets and give a counterexample to show that it does not hold for bags.

Remember, to prove a bag RA equivalence, you must show that if a tuple t appears with multiplicity k on the LHS, then it appears on the RHS with the same multiplicity k and vice versa. If you want more clarification, a sample proof of a bag RA equivalence is provided in Section 5.

R and S are arbitrary sets or bags (as appropriate), C and D are arbitrary selection or join conditions unless otherwise specified, and A is an arbitrary set of attributes. If considering the set RA, assume the operators are the set versions, if considering the bag RA assume they are bag versions.

(a) $(R \cap S) - T \equiv R \cap (S - T)$

(b) $R \bowtie_C (S \bowtie_D T) \equiv (R \bowtie_C S) \bowtie_D T$, assuming C contains only attributes from R and S and D contains only attributes from S and T.

(c) $\pi_A(R \cup S) \equiv \pi_A(R) \cup \pi_A(S)$

(d) $\sigma_{C \vee D}(R) \equiv \sigma_C(R) \cup \sigma_D(R)$

(e) $R \cup (S \cap T) \equiv (R \cup S) \cap (R \cup T)$

(f) $R \cap (S \cup T) \equiv (R \cap S) \cup (R \cap T)$

In grading, each equivalence is worth 5 points.

4 Submission Instructions

Submit a .zip archive containing:

- a .pdf file containing the **relational algebra** queries for question 1 and your answers to question 3. All answers must be typeset, a scan of handwritten work is not acceptable and will receive an automatic zero points. Clearly label which answer corresponds to which question.
- a .txt file containing the answers to all the remaining questions. Clearly label which answer corresponds to which question.
- if you consulted any external sources, an acknowledgments.txt file as required by the academic integrity policy.

5 Appendix: sample proof of a bag RA equivalence

This section contains an example of how to prove a bag RA equivalence – obviously, one that is not featured on the homework itself. Two proofs are given, one is more informal and one is more formal. Either approach is acceptable in your answers.

Equivalence to prove: $\sigma_{A \wedge B}(R) \equiv \sigma_A(\sigma_B(R))$

5.1 Informal version

First we show $\sigma_{A \wedge B}(R) \subseteq \sigma_A(\sigma_B(R))$ for arbitrary R.

Fix an arbitrary R and suppose t appears in the result of the LHS query, i.e. in $\sigma_{A \wedge B}(R)$, with multiplicity k. Then from the semantics of bag selection, we know that t appears in R with multiplicity k and that $A \wedge B$ holds on t.

Let us now consider what happens to t in computing $\sigma_A(\sigma_B(R))$. t satisfies $A \wedge B$ so it also satisfies B. From the semantics of bag selection, it will therefore appear in $\sigma_B(R)$ with multiplicity unchanged (i.e. k). We also know t satisfies A, so therefore it appears in $\sigma_A(\sigma_B(R))$ with multiplicity still unchanged (i.e. k). Thus t appears in the result set on the RHS with multiplicity k as required.

We also need to show that $\sigma_{A \wedge B}(R) \supseteq \sigma_A(\sigma_B(R))$ for arbitrary R.

I haven't given you this direction because it is pretty similar to the above and you should be able to figure it out yourselves. However, in your homework solutions you should state both directions fully and explicitly.

5.2 Formal version

We introduce some notation: we will use the notation $t : k$ to mean that tuple t appears with multiplicity k (i.e. there are k copies of t).

Then the formal definition of bag selection is as follows for an arbitrary bag R and selection condition C: $\sigma_C(R) = \{t_k : m_k \mid t_k : m_k \in R \text{ and } C \text{ holds on } t_k\}$.

First we show $\sigma_{A \wedge B}(R) \subseteq \sigma_A(\sigma_B(R))$ for arbitrary R.

Fix an arbitrary R and suppose $t : k \in \sigma_{A \wedge B}(R)$. Then from the definition of bag selection, we know that $t : k$ is in R and that $A \wedge B$ holds on t.

Let us now consider what happens to t in computing $\sigma_A(\sigma_B(R))$. t satisfies $A \wedge B$ so it also satisfies B. From the definition of bag selection, we know $t : k$ is in the result of evaluating $\sigma_B(R)$. We also know t satisfies A, so therefore $t : k$ is in the result $\sigma_A(\sigma_B(R))$. Thus t appears in the result set on the RHS with multiplicity at least k as required.

Again the other direction is basically symmetric (but you should write it down).