CS 4320/5320 Homework 1

Spring 2016

Due February 19th, 2016 at 11:59 pm

This assignment is out of 100 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 3 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 (42 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 Customer (cid INTEGER PRIMARY KEY,
                      cname VARCHAR(20) NOT NULL);
CREATE TABLE Product  (pid INTEGER PRIMARY KEY,
                       pname VARCHAR(20) NOT NULL,
                       msrp INTEGER NOT NULL);
CREATE TABLE Purchase (cid INTEGER,
                       pid INTEGER,
                       date DATE,
                       price INTEGER NOT NULL,
                       PRIMARY KEY (cid, pid, date),
                       FOREIGN KEY (cid) REFERENCES Customer(cid),
                       FOREIGN KEY (pid) REFERENCES Product(pid));
```

This schema describes a simple retail setting. The Customer table keeps track of customers; every customers has a cid and a name. The Product table keeps track of products; every product has a pid, a name, and a MSRP (manufacturer's suggested retail price), which is a standard/recommended price for that product. The Purchase table keeps track of which customers purchased which products and when. For every purchase, it also records the price paid, which may be higher than, lower than or equal to the product's MSRP. You may assume the database contains no null values.

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. In particular, please make sure your query returns the right column(s).

(a) (5 points) Find all the customers who have never paid MSRP or more for any product. That is, for every purchase the customer made, the price paid was below the MSRP for the product. Display a (cid, cname) pair for each such customer, with no duplicates. If a customer has purchased no products do not include them in the result.

(b) (5 points) Find and display all the customer/product pairs (cid, pid) such that customer cid has purchased product pid exactly three times.

(c) (5 points) For each product, find the highest price at which it has ever been purchased. Display (pid, price) pairs. If a product has never been purchased, do not include it in the result.

(d) (5 points) For each customer including those who have made no purchases, list their cid, cname and number of purchases they have made, **without using any outer join operator**. That is, give a query that gives the same answer as the following but uses no outer joins.

```
SELECT cid, cname, COUNT(pid)
FROM (SELECT * FROM Customer LEFT OUTER JOIN Purchase USING (cid)) AS TEMP
GROUP BY cid, cname;
```

Queries using an outer join will get zero points.

(e) (7 points) Consider only customers who have made at least one purchase. For each pair of such customers compute their Jaccard similarity with respect to the products they have purchased. That is, if customer A has purchased set S_A of products and customer B has purchased set S_B of products, their Jaccard similarity is $\frac{|S_A \cap S_B|}{|S_A \cup S_B|}$, the size of the intersection of the two sets divided by the size of their union. For every pair of customers, output the two cids and the Jaccard similarity. Each pair should be output only once, i.e. if $(1, 3, j)$ is a matching pair of cids with Jaccard similarity j, you should return only $(1, 3, j)$ and not $(3, 1, j)$. Sort your output in descending order by Jaccard similarity, i.e. the most similar pair should be listed first. Note that this question asks about products and not product purchases: if a customer has purchased a product more than once, this only counts as one product for the purposes of this computation.

Write the following queries in **set 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.)

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

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

(h) (5 points) the same query as part (c) above.

2 Bag Relational Algebra (28 points)

In class, we briefly discussed the bag relational algebra, with operators that work on relations that are bags (multisets). In this question, you will further explore the bag relational algebra, with a focus on the *duplicate elimination operator* δ. In simple terms, the δ operator retains all the tuples in the original relation but resets the multiplicity to 1. If R consists of the tuples $\{(1,2),(1,2),(2,3)\}$, then $\delta(R) = \{(1,2),(2,3)\}$. Formally, $\delta(R) = \{t \mid t \in R\}$.

The questions below ask you whether δ commutes with various relational operators. In each case below, state whether the proposed equivalence is true or false. If it is true, prove it. If it is false, provide a counterexample where the equivalence fails. In grading, each equivalence is worth 4 points.

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 bags, C is an arbitrary selection or join condition, A is an arbitrary set of attributes, and $\cup_B$, $\cap_B$ and $-_B$ are bag union, intersection and difference as introduced in class. All other operators (σ, π etc.) are the bag versions as well.

(a) $\delta(\sigma_C(R)) \equiv \sigma_C(\delta(R))$

(b) $\delta(\pi_A(R)) \equiv \pi_A(\delta(R))$

(c) $\delta(R \times S) \equiv \delta(R) \times \delta(S)$

(d) $\delta(R \bowtie_C S) \equiv \delta(R) \bowtie_C \delta(S)$

(e) $\delta(R \cup_B S) \equiv \delta(R) \cup_B \delta(S)$

(f) $\delta(R \cap_B S) \equiv \delta(R) \cap_B \delta(S)$

(g) $\delta(R -_B S) \equiv \delta(R) -_B \delta(S)$

3 Constraints and Triggers (30 points)

This question uses PostgreSQL. Consider the following database table, which we also provide for you in a .txt file so you can copy/paste it into your system:

```
CREATE TABLE Reserves (resnum INTEGER PRIMARY KEY,
                       sid INTEGER NOT NULL,
                       bid INTEGER NOT NULL,
                       startdate DATE NOT NULL,
                       starttime TIME(0) NOT NULL,
                       enddate DATE NOT NULL,
                       endtime TIME(0) NOT NULL);
```

This is an extension of the `Reserves` example we used in class. Each reservation has a reservation number `resnum`, as well as a starting and ending date and time. Note that `sid` and `bid` should theoretically be foreign keys into the `Sailors` and `Boats` tables, but we have omitted those tables for simplicity.

In this question, you will specify a number of constraints on `Reserves`:

1. Every reservation must start before it ends. Also, reservations can only start and end on whole hours (you can book a boat from 9:00 to 10:00, but not from 9:30 to 10:30, for example).
2. We cannot have overlapping reservations for any boat. For example, if boat 101 is reserved from 10 to 12 on a given day, there cannot be a second reservation from 11 to 15 for the same boat on the same day. However, a reservation from 12 to 14 is allowed, that is, the start time of one reservation may be the same as the end time for another reservation. You don't have to worry about overlapping reservations *for the same sailor*, only for the same boat.
3. Any boat may only be reserved for a maximum of 8 hours in any consecutive 24-hour period. This also means the maximum length for any single reservation is 8 hours.

(a) (4 points) Add a `CHECK` constraint to the table to enforce constraint (1). That is, provide an `ALTER` statement that we can paste into the console, so that if we subsequently try to insert reservations that violate any of the above constraints (or modify an existing reservation to violate constraints), our operation will be disallowed.

(b) (6 points) Provide trigger code to enforce constraint (2). We will grade by 1) creating `Reserves` using the `CREATE` statement above, 2) copying and pasting your provided code into the console, and then 3) attempting to make various changes to `Reserves`. Changes that violate the overlap constraint should be rejected but all changes that do not should be accepted. You may assume we will not make insertions/modifications that would violate constraint (1).

(c) (20 points) Provide trigger code to enforce constraint (3). We will grade as in (b), and we will not make insertions/modifications that would violate constraints (1) or (2).

As you write your code, be aware that PostgreSQL supports data types related to dates, times, and even time intervals; a study of the relevant documentation will uncover a lot of useful information.

If you want to use print statements for debugging, you can use `RAISE NOTICE`, as follows:

`RAISE NOTICE 'I am a debugging message!'`.

Finally, when working with triggers, it is good to know how to create a fresh database. When at the console, you can type in `CREATE DATABASE mydb;` This will create a new database called `mydb`. You can see all the available databases by using the command `\l`.

Once your database is created, you can connect to it by typing `\connect mydb` You can now do whatever work you want in this database. When you are done, you can delete the database as follows:

`\connect postgres` (this disconnects you from `mydb`)
`DROP DATABASE mydb;`

4 Submission Instructions

Submit a .zip archive containing:

- a .pdf file containing the **relational algebra** queries for question 1 and your answers to question 2. 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.

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).