Assignment 1: The Relational Model and Relational Algebra (100pts)

Due: Thursday, Jan. 14th 2AM PST (**morning**)

Before You Begin!

Writing relational algebra expressions in a program like MS Word or LaTeX can be a little annoying. We will be using the Canvas quiz interface for you to submit your solutions this quarter, which supports a variety of answer formats (including a built-in LaTeX editor). The quiz will be accessible on Canvas Thursday. You may also write your solutions by hand and provide them as images. The main requirement is that your answers be neat and understandable. If you hand-write your solutions, **please make sure your scanned/photographed hand-written solutions are legible!** Some scans are barely dark enough to read, and they require a lot of effort on our side. Failure to follow these guidelines will result in point deductions.

To ensure you don't lose your work when answering each question on Canvas, we recommend you save your answers on a Word document (or on paper). You can submit your answers as many times as you would like - the last submission before the deadline will be the one that is graded.

Finally, keep track of how long you spent on the assignment, and fill out the feedback survey on the course Canvas when you have finished it. You will receive +3 bonus points for this.

Part A: Relational Algebra Book Problems (65 points)

These problems come from Database System Concepts, 5th ed. Unfortunately, the 6th edition of the textbook doesn't have very interesting or challenging relational algebra problems, so we will use the 5th edition's problems. A PDF of the problems and supporting material from the 5th edition is provided on Canvas under the Assignments Module. Do the specified problems from the problems in the PDF file. Each of these problems requires you to write a number of queries or update operations in the relational algebra.

Hints:

- Like math problems, most of these problems have reasonably simple solutions, but it might take a while to find the "trick." If your solution to a problem gets unwieldy or convoluted, you might be on the wrong track.
- As a corollary to the above, unnecessarily complicated solutions will be penalized.
- If you see how to solve a part of the problem, write that part of the query and note what it does. It will help you figure out the solution, and you will be more likely to get partial credit if it's wrong.
- Familiarize yourself with the natural join operation – it makes many queries quite straightforward to write.
- If you find you are reusing a particular sub-expression, consider using a temporary relation to represent that result. Recall that we assign a temporary relation to a relvar with the assignment operator ←, not the rename operator ρ.

- **Note that <u>unlike SQL</u>, predicates for the select operation <u>cannot</u> contain relational algebra operations!** Select predicates may only contain comparisons and the logical operators AND, OR and NOT.

- Related to the previous point, note that an expression like $\mathcal{G}_{\mathbf{max}(A)}(r)$ produces a **<u>relation</u>** with a single attribute, containing a single tuple. Thus, to compare this value to other results, you must use either a Cartesian product or some kind of join operation to combine this relation with the rows you want to compare it against.

Problems:

Each problem's point value is specified below.

- <u>Problem 2.5</u>, a-e *(parts a-c are 3 points each; d-e are 4 points each; 17 points total)*
- <u>Problem 2.6</u>, a-c *(part a: 3 points; part b: 2 points; part c: 3 points; 8 points total)*
- <u>Problem 2.7</u>, a-c *(5 points per part; 15 points total)*

  Hints for part b:

  - To clarify, only apply the 10% raise if the *resulting* salary would not be over $100K. If the resulting salary would be > $100K, apply the 3% raise.
  - Use temporary relations and multiple steps to simplify the solution.
  - Consider how you might use the rename operation (or generalized projection) to easily compute the set of managers from the *works* relation.
- <u>Problem 2.8</u>, a-b *(5 points per part; 10 points total)*
  - Note that the problem says "<u>more than</u> two." In other words, three or more.
  - Accounts are stored in the *account* and *depositor* relations; pages 40-42.
  - Part b is a bit grungy. Implementing such queries without aggregate functions usually is.
- <u>Problem 2.9</u>, a-c *(5 points per part; 15 points total)*
  - Your queries should simply return the company name.
  - Note that the **min** and **max** aggregate functions cannot be used in a select predicate to find a particular tuple. If you want to find tuples with a max or min value, you must generate a tuple with that min/max value for one of its attributes, and then compare candidate tuples to that tuple. In other words, a Cartesian product will be involved at some point.
  - You might want to use temporary relations on these problems to simplify things.

Part B: Relational Division Operations (15 points)

Given these relations:

monkey_likes(*name*, *food*)

name	food
Bobo	apples
Bobo	bananas
Jojo	apples
Jojo	oranges
Jojo	bananas
Lulu	oranges
Lulu	bananas
Guenter	apples
Guenter	oranges
Guenter	bananas
Guenter	tofu

monkey_foods(*food*)

food
apples
oranges
bananas

The relational division operation can be defined in multiple ways, depending on exactly how it should behave in different circumstances. For example, the relational division operation in the book will ignore the tuple (Guenter, tofu) when dividing *monkey_likes* by *monkey_foods*; thus, *monkey_likes* ÷ *monkey_foods* will be {(Jojo), (Guenter)}.

(Each part is worth 5 points.)

a) Using the definition of relational division given in the book, repeated below, explain why the above is true. (Don't compute the entire result for every single step; just clearly show or explain why Guenter would appear in the result.)

$$r \div s = \Pi_{R\text{-}S}(r) - \Pi_{R\text{-}S}((\Pi_{R\text{-}S}(r) \times s) - \Pi_{R\text{-}S,S}(r))$$

Note that the expression $\Pi_{R\text{-}S,S}(r)$ doesn't actually remove any attributes; it simply ensures that the *order* of the attributes is identical to that produced by the expression $(\Pi_{R\text{-}S}(r) \times s)$.

b) Define an "exact" version of the division operator $r \div_E s$, where the result only contains tuples from *r* that exactly match the contents of *s*. (e.g. *monkey_likes* $\div_E$ *monkey_foods* should simply produce {(Jojo)}, since Guenter also likes a food that doesn't appear within *monkey_foods*.) To obtain full credit, you must also explain how your definition works. *(Hints: feel free to start with the book version, and feel free to use ÷ in your definition of* $\div_E$*)*

SQL doesn't provide a relational division operation directly, but it is straightforward to implement using grouping and aggregation operations, which SQL does provide. Since we will begin working with SQL in the next assignment, let's look at how we might implement relational division with the tools that SQL provides.

c) Write a relational algebra expression that computes $r \div s$ using grouping and aggregation instead of the approach used above. You can of course use other relational algebra operations as needed, except do not use the set-difference relational operator for this solution. (You can use set-difference for manipulating schemas, but not relations.)

Make sure your solution doesn't give false positives; for example, if the tuples {(Henry, apples), (Henry, bananas), (Henry, tofu)} were added to *monkey_likes*, make sure that Henry won't appear in the result. (Henry likes three foods, and three foods are listed in *monkey_foods*…)

Part C: Query Optimization and Equivalence Rules (20 points)

Using the relational algebra to specify queries can be tedious at times, but it also has a big benefit that there are many different ways to state a particular query. Although different versions will produce identical results, some versions can be evaluated much more quickly than others.

For example, given this schema:

store(<u>*store_id*</u>, *store_city*, *store_state*)
employee(<u>*emp_id*</u>, *emp_name*, *store_id*, *salary*)

Let's look at a query for finding the names of all employees who work in stores in Idaho, and make at least $70,000 a year. The relational algebra expression would look something like this:

$$\Pi_{emp\_name}(\sigma_{store\_state = \text{"ID"} \wedge salary \geq 70000}(store \bowtie employee))$$

However, if *employee* is a large relation then this query will be very slow to compute, even if very few tuples actually satisfy the selection criteria. What we would like the database to do is rearrange the query, producing an equivalent expression that will evaluate much faster, such as this:

$$\Pi_{emp\_name}(\sigma_{store\_state = \text{"ID"}}(store) \bowtie \sigma_{salary \geq 70000}(employee))$$

Note that we have broken the select operation into two separate operations, and we have rearranged the query so that these operations are applied before the join takes place. This ensures that the join will receive the smallest number of inputs possible.

Good database engines will perform optimizations like this to make queries run faster; the better the database engine, the more kinds of optimizations it will know how to apply.

Optimizations like these are driven by **equivalence rules**, which state that two relational algebra expressions are equivalent. In other words, given any legal database instance (that is, a database that satisfies all primary/candidate and foreign key constraints), the two equivalent expressions would generate the exact same results. An example equivalence rule would be: $\sigma_{P1 \wedge P2}(E) = \sigma_{P1}(\sigma_{P2}(E))$ ("conjunctive select operations can be deconstructed into a sequence of individual selections"). The book lists a number of equivalence rules in section 13.2.1. (Rule 7b in this section shows that the two queries given above are in fact equivalent.)

Here are a number of potential equivalence rules; you must determine whether each pair of expressions is in fact equivalent. If the expressions are equivalent, give proof. (Your proof doesn't need to be rigorous, but you should be able to make it clear exactly why the expressions are equivalent.) If they are not equivalent, give a counterexample. Answers will only receive credit if they are proven one way or another.

(Each part is worth 4 points.)

a) Are $\sigma_\theta({}_A\mathcal{G}_F(r))$ and ${}_A\mathcal{G}_F(\sigma_\theta(r))$ equivalent?
- A is a set of grouping attributes
- F is a set of aggregation functions
- θ is a predicate using only attributes from A

b) Are $\Pi_A(r - s)$ and $\Pi_A(r) - \Pi_A(s)$ equivalent?
- Relations r and s have compatible schemas, e.g. $r(a, b)$ and $s(a, b)$.

c) Are $(r \bowtie s) \bowtie t$ and $r \bowtie (s \bowtie t)$ equivalent?
- r is a relation with schema $(a, b1)$
- s is a relation with schema $(a, b2)$
- t is a relation with schema $(a, b3)$

d) Are $\sigma_\theta(r \bowtie s)$ and $\sigma_\theta(r) \bowtie s$ equivalent?
- θ is a predicate using only attributes from r

e) Are $\sigma_\theta(r \bowtie s)$ and $r \bowtie \sigma_\theta(s)$ equivalent?
- θ is a predicate using only attributes from s

Feedback Survey (+3 bonus points)

Complete the feedback survey for this assignment on the course website, and 3 points will be added to your score (max of 100/100).