University of California, Los Angeles
Department of Computer Science

Computer Science 143
Spring 2020

Prof. Ryan Rosario

Sample Midterm Exam

Adapted from Forms 19S, 1-5

Solutions.

1. **It's So Logical.** Answer the following questions about the relational algebra.

   (a) (5 points) We learned some basic relational algebra operators and discussed that more complex operators could be constructed based on them. Let R and S be relations. Write a relational algebra expression for R left semijoin S. The left semijoin essentially uses values of S to filter and keep values in R that match on the join key, and then restricts the result to a subset of attributes in R. Thus, the semijoin is sometimes called *restriction*. Write the relational algebra expression for the left semijoin using the operators we learned in class.

   $R \ltimes S = \Pi_{\text{subset of attributes in } R}(R \bowtie S)$

   We were somewhat lenient on the subscript of Π but it must refer to attributes of R, and not R itself. If a theta join was used, it must specify some kind of constraint.

   (b) (5 points) In class, we discussed the left join. Let R and S be two relations with a single common attribute name between them. Using the basic relational algebra operators we discussed in class ($\Pi, \sigma, \times, \bowtie, \rho, -, \cup, \cap$), write the relational algebra expression for the left join of R and S. Let any null values be represented by ω. Be careful: We cannot test for nulls as a subscript on an operator, and we are asking for a relational algebra expression, not a pure set theory representation.

   $R \bowtie\!\!\!\!\!\!\times\, S = (R \bowtie S) \cup (R - \Pi_{R_1,\ldots,R_n}(R \bowtie S)) \times (\omega, \ldots, \omega)$

   The left join starts with the natural join and adds some more tuples to it. This second set contains all tuples in R but excludes from it those that are contained in the result of the natural join (tuples only in R). This leaves us with a set of incomplete tuples that an RDBMS would fill with null values. We can concatenate a null to these tuples using the Cartesian product $\times$.

   (c) In class, we discussed that the projection operator is the closest operator to a SQL `SELECT`. Explain why it does not do the *exact* same thing as the `SELECT` statement.

   The projection operator Π removes duplicates, so it is actually the same as the `SELECT DISTINCT`, but of all the relational operators we learned, Π is the closest operator to the `SELECT` statement in SQL.

2. **The Not So Friendly Skies.** Suppose you work as a safety analyst for Southwest Airlines. We have a few tables to study. Due to the number of flights operated each day, flight numbers may be recycled, including on the same day. For example, flight 528 on Wednesdays starts in Albany and stops in Orlando, Raleigh, Atlanta, Houston, El Paso, Los Angeles and terminates in San Jose. Assume that each flight number uses the same aircraft (denoted by `tail`, which is like a license plate) throughout its journey. Assume that no flights span two days (depart before midnight, arrive after midnight).

   You will notice that there are no foreign keys defined. We ignore them for simplicity, and assume that these tables are static with no inserts, updates or deletes.

   Special Note: *NONE of the queries in this section require using more than one subquery, or more than one level of subquery. Complicated queries that use additional subqueries will not receive credit. Please use standard `JOIN` queries, as they explicitly capture the join condition. `NATURAL JOIN` and `USING` will not receive credit.*

```
CREATE TABLE eq_flt (
    fltno          smallint,
    tail           char(6) NOT NULL,
    -- i.e. N1234A, N789SW
    PRIMARY KEY(fltno)
    -- minimal key, no flight number can depart at same time to different destination.
);
-- which individual aircraft (tail) services a particular flight on a particular date/time.

CREATE TABLE flt (
    fltno          smallint,
    -- i.e. 424, 1297
    sch_dep        timestamptz,  -- scheduled departure
    sch_arr        timestamptz,  -- scheduled arrival
    orig           char(3) NOT NULL,
    dest           char(3) NOT NULL,
    -- i.e. DEN, SJC, LAX, BUR, LGA, MMH
    dist           smallint,  -- distance in nautical miles (redundancy for simplicity)
    PRIMARY KEY(fltno, sch_dep)
);
-- flights and their routes

CREATE TYPE model AS enum ('B737', 'B738', 'B38M');
-- B737 = 737-700, B738 = 737-800, B38M = 737 MAX 8
CREATE TABLE actype (
    tail          char(6) PRIMARY KEY,
    ac            model NOT NULL
);
-- mapping tails to aircraft model type
```

REST OF PAGE INTENTIONALLY LEFT BLANK

(a) (5 points) Management has decided to cut costs and only restock snacks and drinks on an aircraft when it either has no more flights that day, or once it flies more than 2,000 nautical miles in a day. We want to get a head start, so we want to find out which aircraft (tails) will exceed the 2,000 nautical mile mark tomorrow. Write a query to answer this question. Assume you have a function `TOMORROW()` that returns tomorrow's date and a function `DATE()` that extracts the date part of a timestamp.

```
SELECT
    tail,
    SUM(dist) AS total_distance
FROM flt f
JOIN eq_flt e
ON e.fltno = f.fltno
WHERE DATE(sch_dep) = TOMORROW()
GROUP BY tail
HAVING SUM(dist) > 2000;
```

(b) (10 points) A crew fuels an aircraft before it takes off, and records the amount of fuel, and a separate crew checks the fuel level after a flight arrives at the gate. Suppose it's not uncommon for a mechanic to forget to record the landing fuel level, particularly if the tanks are known to be near empty at arrival. These recordings are stored in two tables shown below. Write a query that computes the average liters of fuel consumed per route (origin, destination pair). Only consider routes that the airline actually flies (not all pairs of origin/destination). You can assume that if the mechanic did not enter a fuel reading when the plane arrived, then the total fuel consumed shall be the amount of fuel that was present before takeoff for that particular journey. **Hint:** Be very careful. You may want to sketch out this query on the back of the page before writing a response.

```
CREATE TABLE to_fuel (
    fltno   smallint,
    sch_dep timestampz,
    orig    char(3),
    dest    char(3),
    liters  integer,
    PRIMARY KEY(fltno, sch_dep)
);
```

```
CREATE TABLE arr_fuel (
    fltno   smallint,
    sch_dep timestampz,
    orig    char(3),
    dest    char(3),
    liters  integer,
    PRIMARY KEY(fltno, sch_dep)
);
```

```
SELECT
    q.orig,
    q.dest,
    AVG(fuel_consumed)
FROM (
    SELECT
        l.orig,
        l.dest,
        l.liters - COALESCE(r.liters, 0) AS fuel_consumed
    FROM takeoff_fuel l
    LEFT JOIN arrival_fuel r
    ON l.fltno = r.fltno
) q
GROUP BY q.orig, q.dest;
```

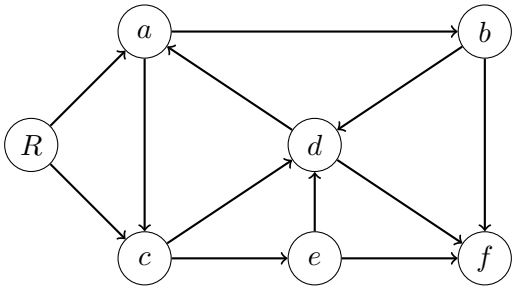
Other variations were also accepted.

(c) (5 points) In this problem we stored the fuel measurements in two different tables. An alternative approach would be to have one record per takeoff/landing. Briefly explain which should be preferred and why. (*Does not require more than 50 words.*)

We accepted 3 possible answers: use two tables, use one table, or "it depends." If the response was **one table**, the response needed to address redundancy, load on the table (read/write) due to updating rows on landing as well as the undesired presence of nulls. If the response was **two tables**, the response needed to discuss the ease of appends and the lack of nulls. A response of **it depends** needed to describe the aforementioned issues. Any of the three methods was acceptable, but it must be defended well.

EXAM CONTINUES ON THE NEXT PAGE

3. **Access Denied.** We have the following authorization graph for some privilege p granted by root user R.



(a) (2 points) Suppose we revoke authorization from user a. The user(s) that lose their authorization with the `CASCADE` option is/are (mark all that apply):

[a] [b] [c] [d] [e] [f] [R]

(b) (2 points) After the previous revoke succeeds, suppose we then revoke authorization from user c. The user(s) that lose their authorization with the `CASCADE` option after **all (including those in part a)** of these revocations is/are (mark all that apply):

[a] [b] [c] [d] [e] [f] [R]

(c) (6 points) **Yo Ho, Yo Ho a Hacker's Life for Me**. It's time to put on the pirate hat. Suppose we have a web app that retrieves contact info for a particular UCLA student using a table `directory`. Suppose in the same schema there is a relation `bol` containing BOL usernames and passwords (with attributes `username` and `password`). Assume the user has read access to both `bol` and `directory` tables and, for simplicity, that both columns in both tables are `text`. Perform a SQL injection to steal all usernames and passwords using the following query. The query must **only** returns usernames and passwords. Note that you may not need all of the lines provided.

```
SELECT
    name,
    email
FROM directory
WHERE name = '' AND 1=2 UNION SELECT username, password FROM bol;
```

Replacing `UNION` with a semicolon was also acceptable. Safe queries did not receive any credit.

4. **More Queries and Joins**. Write a *succint* response to each question. Only correct responses will receive credit. Correct responses should not be long. There is a word maximum for each response and responses should not exceed the provided lines. Exceeding this maximum may result in 0 points.

   (a) (5 points) **Can I Join You Two?** Give a specific example of an application when we would want to intentionally compute a `CROSS JOIN` between two distinct relations R and S. *A response that simply repeat the definition of a `CROSS JOIN` without a specific application will not receive credit.* (*Does not require more than 50 words.*)

   Matching problems are usually good examples. Matching shirts and shoes, roommates, dating etc. The response then had to describe what to do with each match, for example, compute compatibility.

   (b) (5 points) **The Return of FOAF.** We can solve certain kinds of graph problems using self joins in SQL. For the graph types listed below, either explain why using a self join cannot be used to solve the problem, or give a specific condition under which a self join cannot be used to solve the problem, or solves it very inefficiently.

   An undirected graph: the relational model is inherently directed. To simulate an undirected graph, we would need to make sure we insert both (A, B) and (B, A), otherwise not possible
   A directed graph: cycles would either cause an infinite recursion, or will at least generate a bunch of extra rows which is inefficient in space and RAM.

END OF EXAM

1. Make sure that you have written your name and UID on page 1.
2. Make sure you marked all of your answers for questions 1 and 2 in the right hand margin of pages 2 and 3. Marks should be distinct, with errors erased completely so we do not mistake them as intentional responses.
3. Make sure your responses are legible and that errors are cleanly erased or crossed out.
4. **Prepare to show a photo ID when handing in your exam.**