CSE 414 Midterm

April 28, 2017

Name: _____Solution_____

Question	Points	Score
1	35	
2	15	
3	30	
4	21	
Total	101	

- **Do not open** the test until instructed to do so.
- The test is closed book and electronics. You are allowed only one page of notes (double sided).
- You have **50 minutes** to complete the test.

Problem 1 (SQL)

Part A: Schema

You would like to create a database to store recipes along with information about local grocery stores that can supply the ingredients of those recipes. In more detail, you would like your database to contain the following relations:

- `Recipe(rid, rname)`, where `rid` is a unique integer ID for the recipe and `rname` is a string name.
- `Ingredient(rid, pname, amount)`, where `rid` is the ID of a known recipe, `pname` is the name of the ingredient, and `amount` is the weight in ounces of that ingredient in the recipe.
- `Grocer(gid, gname, city)`, where `gid` is a unique ID for the grocer, `gname` is their name, and `city` is a name of the city where the grocer is located.
- `Sells(gid, pname, price)`, where `gid` is the ID of a known grocer, `pname` is the name of an ingredient, and `price` is the price per ounce of the ingredient at that grocer.

Most of the SQL to create these relations is provided below. However, some important parts are still missing. **Fill in the remaining parts** so that the SQL will create a schema for these relations that properly reflects all parts of the description above.

```
CREATE TABLE Recipe(
  rid INT,
  rname VARCHAR(100),
  PRIMARY KEY (rid));
```

```
CREATE TABLE Grocer(
  gid INT,
  gname VARCHAR(100),
  city VARCHAR(100),
  PRIMARY KEY (gid));
```

```
CREATE TABLE Ingredient(
  rid INT,
  pname VARCHAR(100),
  amount FLOAT,
  FOREIGN KEY (rid)
    REFERENCES Recipe);
```

```
CREATE TABLE Sells(
  gid INT,
  pname VARCHAR(100),
  price FLOAT,
  FOREIGN KEY (gid)
    REFERENCES Grocer);
```

Part B: Queries

Each item below describes, in English, a query to perform on the database using the schema from the previous page. Translate each query into SQL.

1. Get the names of recipes that use ‘condensed milk’ as an ingredient. You may return them in any order.

```
SELECT DISTINCT rname
FROM Recipe NATURAL JOIN Ingredient
WHERE pname = ‘condensed milk’;
```

2. List each ingredient by name along with the cheapest price at which it is sold by any grocer. Order the results alphabetically by ingredient name.

```
SELECT pname, min(price)
FROM Ingredient NATURAL JOIN Sells
GROUP BY pname
```

3. List each recipe by name along with the total cost to purchase all its ingredients, in the required amounts, assuming each ingredient is purchased at the cheapest price at which it is sold by any grocer. You can return the rows in any order. (It is okay to use a subquery in this case.)

```
SELECT rname, sum(amount * price)
FROM Recipe NATURAL JOIN Ingredient NATURAL JOIN
  (SELECT pname, min(price) price
   FROM Ingredient NATURAL JOIN Sells
   GROUP BY pname)
GROUP BY rid, rname
```

Part C: Indexes

Kevin's life coach told him that he can increase his flow of positive energy by eating foods containing ingredients that start with the letter 'c'. Each time he gets hungry, he uses a SQL query on your database to find such foods.

The first time he ran the query he found a recipe for Jamaican Guinness Punch. Unfortunately, the recipe calls for 10 pounds (160 ounces) of condensed milk, and Kevin's weak arms were unable to carry such a heavy load of ingredients.

Kevin modified his query to exclude ingredients he cannot lift. Fortunately, the punch still tasted great, even without condensed milk, and no recipe other than the punch contained any ingredient that heavy.

Here is Kevin's modified query:

```
SELECT DISTINCT rname
FROM Recipe R, Ingredient I
WHERE R.rid = I.rid AND I.amount < 160 AND
      ('c' <= pname and pname < 'd')
```

(Note that all ingredient names use only lower case letters.)

1. Based on your `CREATE TABLE` statements from Part A, what index (on `Recipe` or `Ingredient`) should already exist in the SQL Server database?

   There is a clustered index on `Recipe(rid)`.

2. Write a SQL statement to create an index that is **likely to speed up** this query. (If you would want the index to have additional properties you cannot describe with standard SQL, just write them in English below your SQL statement.[1])

   ```
   CREATE CLUSTERED INDEX ON Ingredients(pname);
   ```

[1] You do not need any such properties to get full credit for this problem. If you want to add additional information just to be clear, feel free to do so… but be brief.

Problem 2 (Query Optimization)

Below, we consider possible execution plans for the following query, which returns the names of grocers that sell at least one very inexpensive ingredient:

```
SELECT gname
FROM Grocer G, Sells S
WHERE G.gid = S.gid AND S.price < 0.26
```

Assume the following statistics about the two relations:

Table	# tuples	# blocks
Grocer	1000	25
Sells	10,000	100

Assume the following statistics about the `price` column of `S`:

# distinct	Low	High
1000	0.01	2.50

At the bottom of this page, we provide some formulas from lecture that may be useful if you do not already have them in your own notes. The problems start **on the next page**.

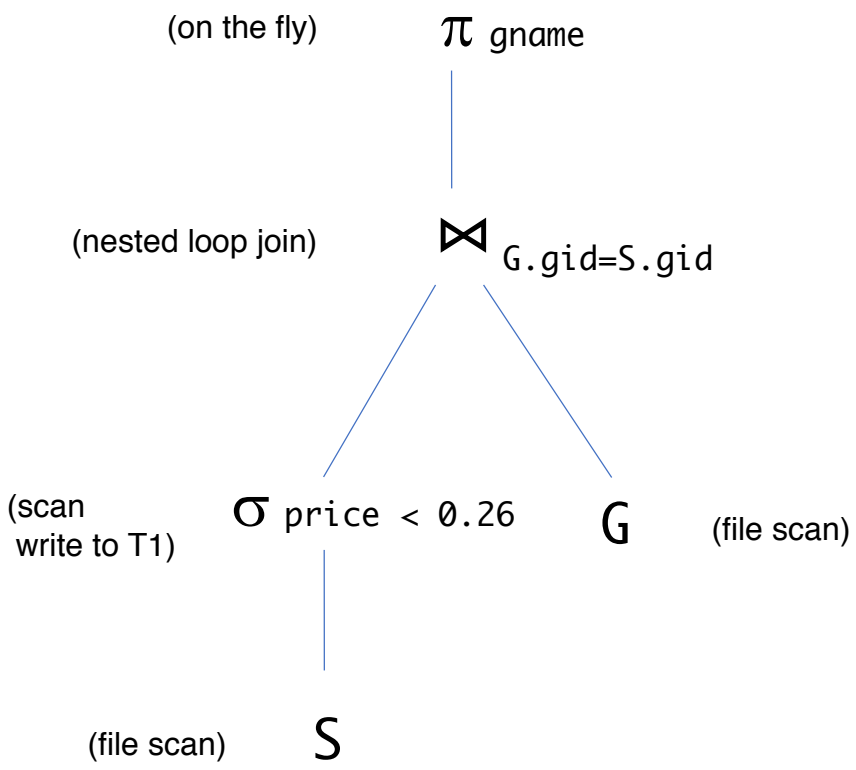
Estimated cost of `X JOIN Y`:

- Using a nested loop, the cost is B(X) + B(X) B(Y), where B(X) is # blocks in X.
- Using a clustered index on Y(A), the cost is B(X) + T(X) B(Y) * E, where T(X) is # tuples in X, and E is the selectivity of the condition `A = c`.
- Using an unclustered index on Y(A), the cost is B(X) + T(X) T(Y) * E.

Estimated selectivity of conditions:

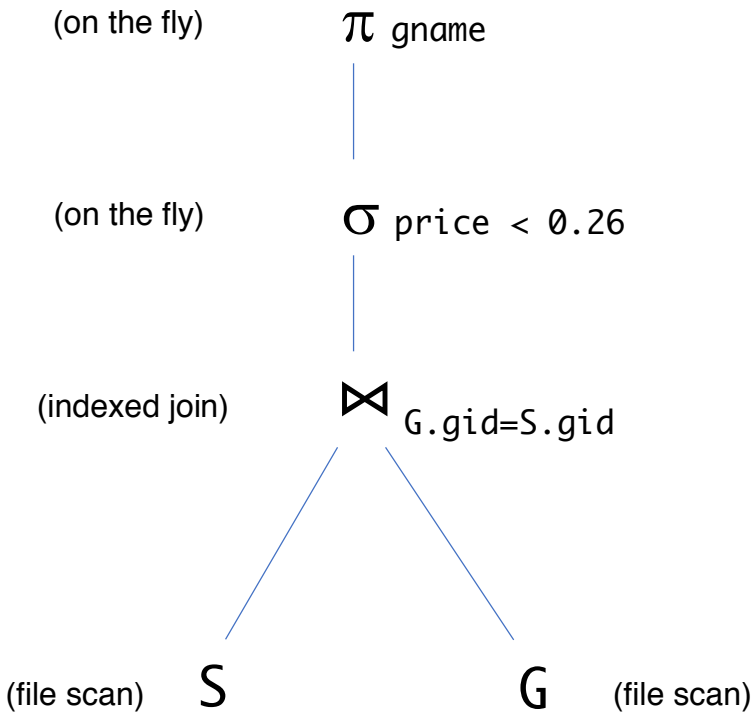
- For `A = c`, the selectivity is 1 / (# distinct values of A)
- For `A < c`, the selectivity is (c - lowest value of A) / (highest - lowest value of A)

1. The total cost of the plan below is _____370_____ disk block I/Os.



Notes:
- The cost to scan S is B(S) = 100. The resulting table has 10 blocks since the selectivity of `price < 0.26` is 1/10. It costs 10 to write those to disk.
- The cost of the join is then 10 + 10 * B(G) = 260.

2. The total cost of the plan below is _____10,100_____ disk block I/Os.



Notes:
- The cost of the indexed join is B(S) + T(S) B(G) E, but E = 1/B(G) because a primary key lookup will always read just a single block (to find one tuple).
- Thus, the total cost is B(S) + T(S) = 100 + 10,000 = 10,100.

3. Circle the plan that is probably faster in this case:

nested loop join (1) indexed join (2)

Problem 3 (RA & Datalog)

1. Consider the following query, which finds the IDs of grocers that sell extremely expensive items (more than $2 per ounce):

```
SELECT DISTINCT gid
FROM Grocer NATURAL JOIN Sells
WHERE price > 2.00
```

   a. Write an extended relational algebra expression that computes the result. (You can write the expression with the usual math notation or draw a tree.)

$\delta(\pi_{\text{gid}}(\sigma_{\text{price} > 2.00}(\text{Grocer} \bowtie \text{Sells})))$

   b. Write a datalog rule that computes the same result or explain why it is impossible to do so.

Q1(x) :- Grocer(x, y, z), Sells(x, u, v), v > 2.00

2. Consider the following query, which finds the IDs of grocers that only sell cheap items (i.e., grocers that do not sell any extremely expensive items):

```
SELECT DISTINCT gid
FROM Grocer
WHERE gid NOT IN
   (SELECT DISTINCT gid
    FROM Grocer NATURAL JOIN Sells
    WHERE price > 2.00)
```

   a. Write an extended relational algebra expression that computes the result.

$$\delta(\pi_{\text{gid}}(\text{Grocer})) - \delta(\pi_{\text{gid}}(\sigma_{\text{price} > 2.00}(\text{Grocer} \bowtie \text{Sells})))$$

   b. Write a datalog rule that computes the same result or explain why it is impossible to do so.

Q2(x) :- Grocer(x, y, z), not Q1(x)

3. Consider the following query, which shows each recipe along with the total amount of ingredients (in ounces) used in that recipe:

```
SELECT rname, sum(amount)
FROM Recipe NATURAL JOIN Ingredient
GROUP BY rid, rname
```

   a. Write an extended algebra expression that computes the result.

π rname, s (γ rid, rname, sum(amount) $\rightarrow$ s (Recipe $\bowtie$ Ingredient))

   b. Write a datalog rule that computes the same result or explain why it is impossible to do so.

This is impossible as datalog does not support aggregation.

Problem 4 (Multiple-Choice / Short Answer)

1. Consider the following SQL query using relations S(A,B,C) and T(C,D):

```
SELECT *
FROM S, T
WHERE ________
```

Circle those columns that can be legally used in the WHERE clause:

A B C D

The name C is ambiguous, so it cannot be used.

2. Consider the following SQL query using relation T(A,B):

```
SELECT ________
FROM T
GROUP BY A
```

Circle those expressions that can be legally used in the SELECT clause:

A B sum(A) sum(B)

We cannot use B since it was not listed after GROUP BY.

3. Give an example of an error (illegal SQL) that will not be caught by SQLite. (Limit your answer to one sentence.)

Writing B in the previous example would be illegal, yet it would not receive an error in SQLite.

4. Suppose that S(A,…) is a relation containing m rows and T(F,…) is a relation containing n rows. Assume that A is the primary key of S and that F is a foreign key reference to S(A). Suppose that we perform an inner join between S and T on S.A = T.F. Circle the maximum number of rows that can be output:

m n m+n m*n

5. Circle the one that best describes how many logical and physical query plans will be considered by a cost-based query optimizer:

   a. one logical plan, one physical plan

   b. one logical plan, multiple physical plan

   c. multiple logical plans, one physical plan

   d. multiple logical plans, multiple physical plans

6. Order the following four query languages by expressive power: standard RA, extended RA, SELECT-FROM-WHERE (SFW), and SQL. Between each two in the ordering, you may write < or = to indicate strictly more powerful or equal power.

   SFW < Standard RA < Extended RA < (or =) SQL

7. Suppose we have a relation T(A,B,C) and we create an index on (A,B). For each type of index listed below, circle those columns that can searched for an exact match using an index of that type on (A,B).

   Hash index on (A,B): A B C (A,B) (B,C)

   B+ tree index on (A,B): A B C (A,B) (B,C)