CSE 344 Section 4 Worksheet Solutions

Relational Algebra & Datalog

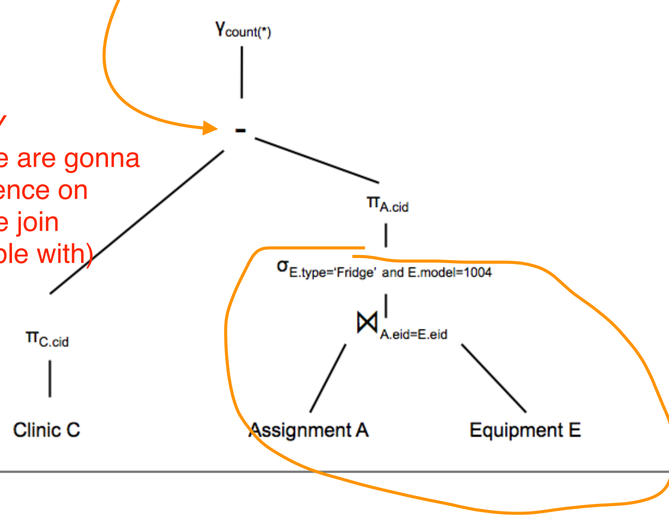
1. **SQL to Relational Algebra.** Write an expression in the form of a logical query plan (i.e., draw a tree) that is equivalent to each of the SQL query below:

a.
```
Clinic(cid, name, street, state)
Equipment(eid, type, model)
Assignment(cid, eid)

SELECT COUNT(*)
FROM Clinic C
WHERE NOT EXISTS (
      SELECT * FROM Assignment A, Equipment E
      WHERE C.cid = A.cid AND A.eid = E.eid
            AND E.type = 'Fridge' AND E.model = 1004
);
```

correlated subquery —> Think like you are pulling this out of our exists statement.
Then, to add in the exists we select these two specific attributes and do a set difference!

have to select ONLY elements we are gonna check existence on (i.e. what we join our main table with)



subquery. Elements of subquery should be separable from main query (i.e. a subtree). So even tho join occurs in subquery, we sort of fold this into the exists statement, because the Clinic C table is outside the subquery.

Solution:

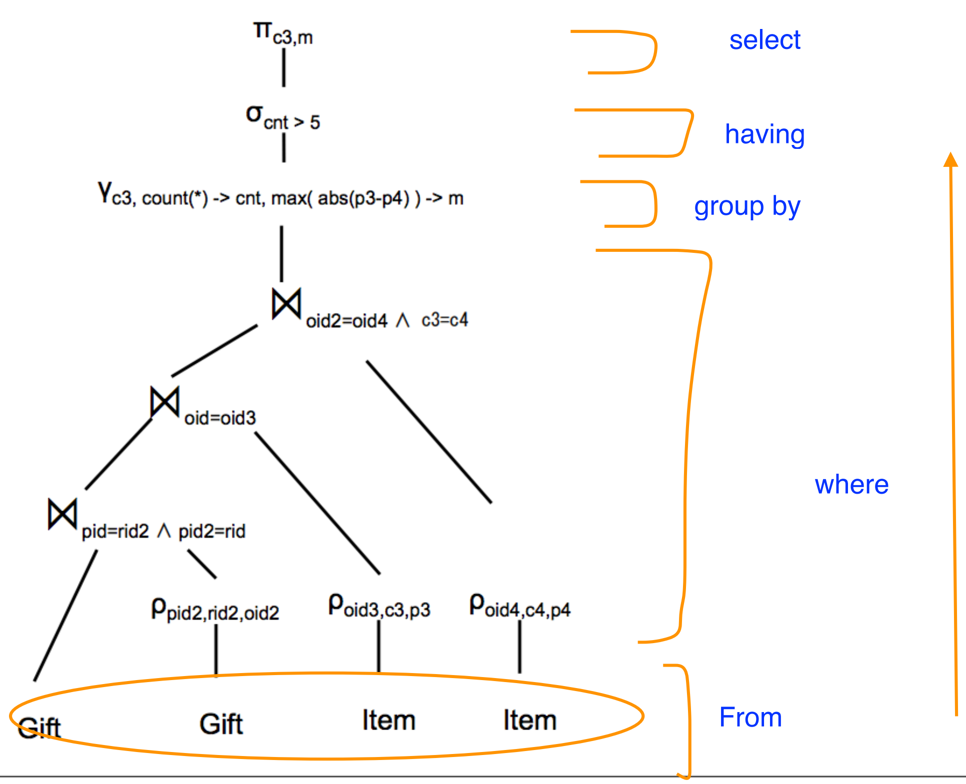
b. `Item(oid, category, price)`
`Gift(pid, rid, oid)`

```
SELECT O1.category, max(abs(O1.price - O2.price))
FROM Gift G1, Gift G2, Item O1, Item O2
WHERE G1.pid = G2.rid
      AND G2.pid = G1.rid
      AND O1.oid = G1.oid AND O2.oid = G2.oid
      AND O1.category = O2.category
GROUP BY O1.category
HAVING count(*) > 5;
```

repeated joins

FWGHOS

Solution:



2. **Datalog**

Consider a graph of colored vertices and undirected edges where the vertices can be red, green, blue. In particular, you have the relations

```
Vertex(x, color)
Edge(x, y)
```

The Edge relation is symmetric in that if (x, y) is in Edge, then (y, x) is in Edge. Your goal is to write a datalog program to answer each of the following questions

1. Find all green vertices.

```
GreenV(x) :- Vertex(x, ‘green’)
```

2. Find all pairs of blue vertices connected by one edge.

```
BluePairs(x, y) :- Vertex(x, ‘blue’), Vertex(y , ‘blue’), Edge(x, y)
```

3. Find all triangles where all the vertices are the same color. Output the three vertices and their shared color.

```
Triangle(x, y, z, a) :- Vertex(x, a), Vertex(y, a), Vertex(z, a), Edge(x, y),
Edge(y, z), Edge(z, x)
```

4. Find all vertices that don’t have any neighbors.

WRONG ANSWER (UNSAFE)

```
LonelyV(x) :- not Edge(x, _)
```

WRONG ANSWER (UNSAFE)

```
LonelyV(x) :- Vertex(x, _), not Edge(x, _)
```

RIGHT ANSWER (SAFE)

```
OnlyX(x) :- Edge(x, _)
LonelyV(x) :- Vertex(x, _), not OnlyX(x)
```

5. Find all vertices such that they only have red neighbors.

```
BlueV(x) :- Vertex(x, _), Edge(x, y), Vertex(y, ‘blue’)
GreenV(x) :- Vertex(x,_), Edge(x, y), Vertex(y, ‘green’)
RedV(x) :- Vertex(x,_), not BlueV(x), not GreenV(x)
```

6. Find all vertices such that they only have neighbors with the same color. Return the vertex and color.

```
SameColor(x, y, a) :- Vertex(x, a), Vertex(y, a)
NotSameNeigh(x) :- Vertex(x, _), Edge(x, y), Edge(x, z), not SameColor (y, z)
OnlySameNeigh(x, a) :- Vertex(x, a), not NotSameNeigh(x)
```

OR

```
Neigh(x, y, a) :- Edge(x, y), Vertex(y, a)
DifferentNeigh(x) :- Neigh(x, y, a), Neigh(x, z, b), a != b
OnlySameNeigh(x, a) :- Vertex(x, a), not DifferentNeigh(x)
```

7. For some vertex v, find all vertexes connected to v by blue vertexes (this one requires recursion).

```
ConnectedTo(x) :- Vertex(x, ‘blue’), Edge(x, v)
ConnectedTo(x) :- Vertex(x, ‘blue’), Edge(x, y), ConnectedTo(y)
```

3. **RA & Datalog**

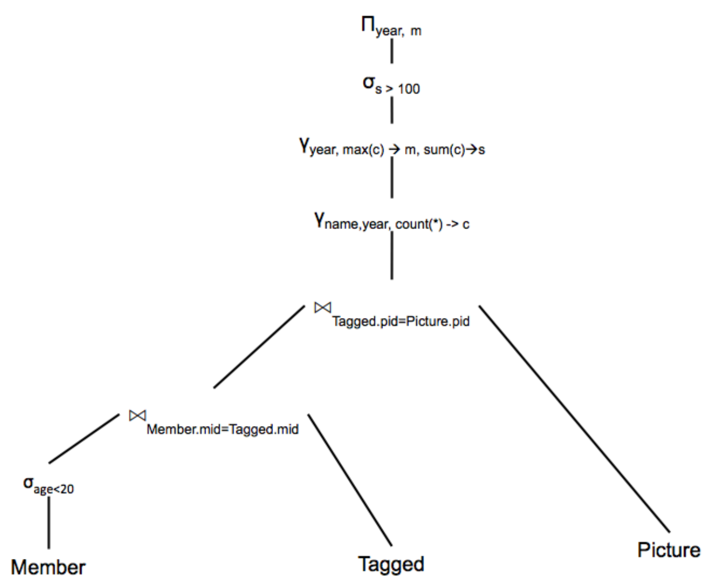
Winter 2016 #2a, b

Consider the following database about a picture tagging website:

```
Member(mid, name, age)
Picture(pid, year)
Tagged(mid, pid)
```

(a) Write a Relational Algebra expression in the form of a logical query plan (i.e., draw a tree) that is equivalent to the SQL query below. Your query plan does not have to be necessarily “optimal”: however, points will be taken off for overly complex solutions.

```
select w.year, max(w.c) as m
from (select x.name, z.year, count(*) as c
      from Member x, Tagged y, Picture z
      where x.mid = y.mid
        and y.pid = z.pid
        and age < 20 group by x.name, z.year) w
group by w.year
having sum(w.c) > 100;
```



Solution:

(b) Write a query in datalog with negation that returns the mid's and names of all members that were tagged only in pictures were Alice was also tagged.

Solution:

```
aliceTagged(pid) :- Member(mid, 'Alice',-),Tagged(mid, pid)
nonAnswer(mid) :- Tagged(mid,pid) not aliceTagged(pid)
answer(mid,name) :- Member(mid,name,-), not nonAnswer(mid)
```

4. **More RA & Datalog**

Autumn 2013 #3a, b, d

Consider the following two relations:

```
Person(id, name)
Trusts(id1, id2)
```

(a) Write a program in non-recursive datalog-with-negation that returns the id's and names of all persons who don't trust Alice.

Solution: Solution 1:

```
Ans(x,y) :- Person(x,y), Person(z,'Alice'), x!=z
```

Solution2:

```
NonAns(x) :- Trusts(x,z), Person(z,'Alice')
Ans(x,y) :- Person(x,y), not NonAns(x)
```

For this question and the next, 2 Points where taken off for adding `Person(x,y)` to the `NonAns` rule. 3 points where taken off for writing `Trusts(x,'Alice')`.

(b) Write a program in non-recursive datalog-with-negation that returns the id's and names of all persons who trust only Alice. (In particular, your query should return all persons who don't trust anyone, e.g. persons who do not appear in Trust.)

Solution: Solution 1:

```
NonAnsw(x) :- Trusts(x,y), not Person(y,'Alice')
Ans(x,y) :- Person(x,y), not NonAns(x)
```

Solution 2:

```
NonAnsw(x) :- Trusts(x,y), Person(z,'Alice'), y != z
Ans(x,y) :- Person(x,y), not NonAns(x)
```

Solution 3:

```
NonAnsw(x) :- Trusts(x,y), Person(y,n), n != 'Alice'
Ans(x,y) :- Person(x,y), not NonAns(x)
```

(d) Consider the following SQL query:

```
select distinct t1.id1
from Trusts t1, Person p1
where t1.id2 = p1.id
  and p1.name = ‘Alice’
  and not exists (select *
                  from Trusts t2, Person p2
                  where p1.id = t2.id1 and t2.id2 = p2.id
                  and p2.name = ‘Bob’)
```

Write this query in the Relational Algebra. Turn in a Relational Algebra plan:

Solution: Write it first in datalog (it's much easier to read this way):

```
V(y) :- Trusts(y,z), Person(z,'Bob')
Q(x) :- Trusts(x,y), Person(y,'Alice'), not V(y)
```

Next, expose the difference clearly:

```
V(y) :- Trusts(y,z), Person(z,'Bob')
U(y) :- Person(y,'Alice'), not V(y) -- here's the set difference
Q(x) :- Trusts(x,y), U(y)
```

The query plan is: