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:

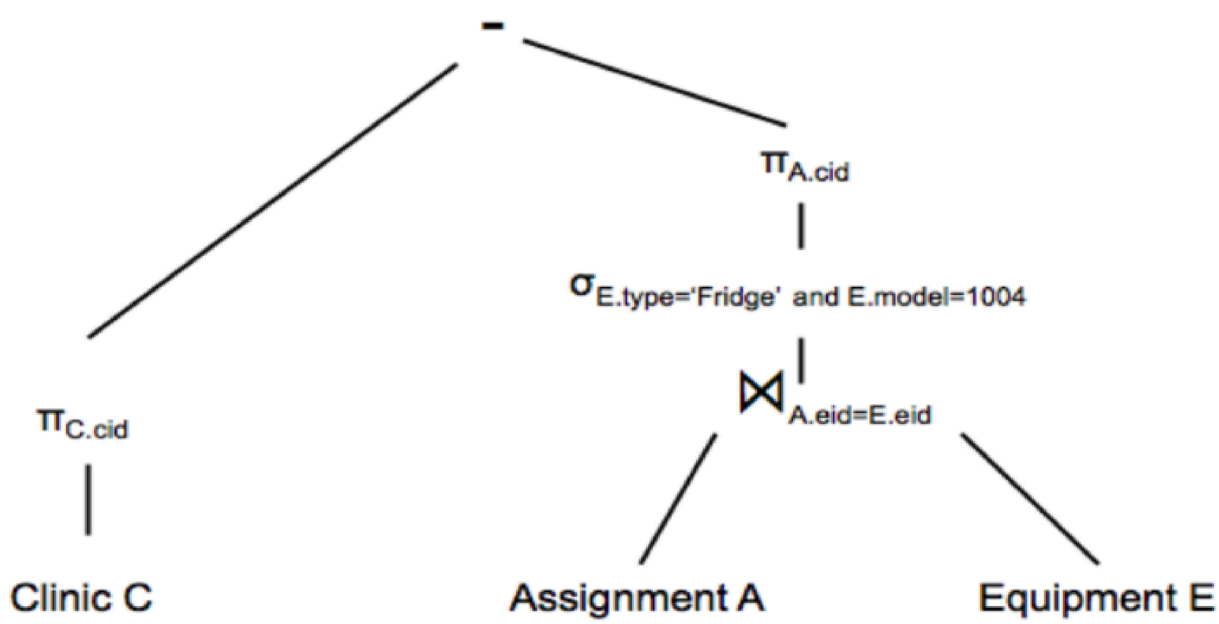
A. Select all clinics that do not have an assignment to a Model 1004 'Fridge'.

Schema: **Clinic(<u>cid</u>, name, street, state)** // cid is the Clinic ID

Equipment(<u>eid</u>, type, model) // eid is the Equipment ID

Assignment(cid, eid)

```
SELECT C.cid
  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);
```



The selection could be pushed down into the join above E, as a query optimization.

B. Select the greatest difference in price between items exchanged between the same two people within the same category, for each category among all categories that have more than 5 such exchanges.

Schema: **Item(<u>oid</u>, category, price)**

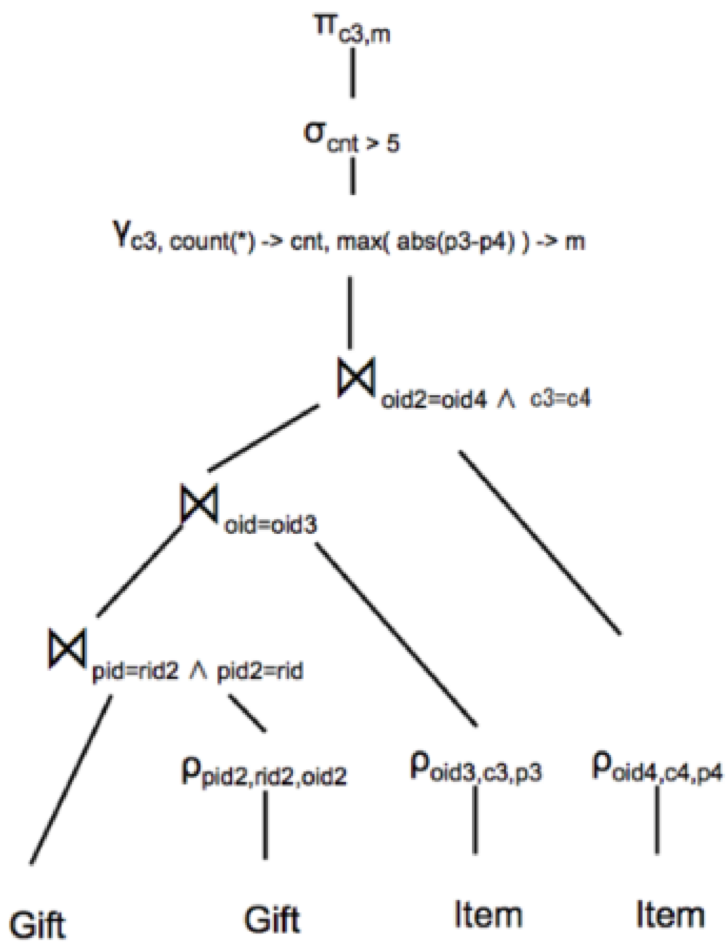
Gift(<u>pid</u>, rid, oid)

Gift.pid: presenter ID

Gift.pid: recipient ID

Gift.oid is a foreign key to Item.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;
```



Solutions that use different join orders are possible.

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:

A. Find all green vertices.

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

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

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

C. 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)
```

D. 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)
```

E. 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)
```

OR here's another solution:

```
NotRedNeighbor(x) :- Vertex(x,_), Edge(x,y), V(y,c), c != 'red'
OnlyRedNeighbor(x) :- Vertex(x,_), !NotRedNeighbor(x)
```

F. 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)
DifferentNeighbor(x) :- Neigh(x, y, a), Neigh(x, z, b), a != b
OnlySameNeigh(x, a) :- Vertex(x, a), not DifferentNeighbor(x)
```

G. For some vertex v, find all vertices connected to v by *all* blue vertices (i.e., those blue vertices connected to v by a chain of blue vertices). [This one requires recursion.]

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