Section 4 Worksheet
Part 1. Datalog Practice
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 color.

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

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

WRONG ANSWER (UNSAFE)

```
LonelyV(x) :- !Edge(x, _)
```

WRONG ANSWER (UNSAFE)

```
LonelyV(x) :- Vertex(x, _), !Edge(x, _)
```

CORRECT ANSWER (SAFE)

```
OnlyX(x) :- Edge(x, _)
LonelyV(x) :- Vertex(x, _), !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,_), !BlueV(x), !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), !SameColor(y, z)
OnlySameNeigh(x, a) :- Vertex(x, a), !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), !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)
```

2. (Midterm 16WI)
Consider the following database about a picture tagging website:

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

1. Find how many times the most tagged member of the year, younger than 20, was tagged, if for that year there were more than 100 tags for all members younger than 20.
a) (optional) Write a SQL query for the problem.
b) (optional) Write an RA tree for the problem.

a) Again, like how we should consider most SQL query writing problems, we should look both sub sub-problems and universal quantifiers/negated existential. The latter is not obvious. However, you might be able to observe that we are considering only members that are younger than 20 in this entire problem. We can make a FROM or WITH subquery to compute only the tuples we care about. We should join all tables because that is the only way we can associate members with picture years. We should also group our joined tables into categories of name and year and count in each category. This is an important concept to generate meta-data in a FROM/WITH subquery. Taking advantage of our meta-data (cnt), we can use aggregates the rest of the way.

```
SELECT W.year, max(W.cnt) as max_count
  FROM (SELECT M.name, P.year, count(*) AS cnt
          FROM Member AS M, Tagged AS T, Picture AS P
         WHERE M.mid = T.mid AND
               T.pid = P.pid AND
               M.age < 20
         GROUP BY M.name, P.year) AS W
 GROUP BY W.year
HAVING sum(W.cnt) > 100;
```

b) When it comes to RA, modeling a FROM/WITH subquery is relatively easy as no unnesting or conceptual perspective needs to be taken. We simply feed in the subquery as if it was a table in its own right. The tricky part of this is when you glue together the subquery and the parent query, we cannot combine the grouping functions because count(*) acts on the groups in the subquery, while sum(…) and max(…) act on the tuples output by the subquery.

2. Write a datalog query with negation that returns the mid's and names of all members that were tagged *only* in pictures where an 'Alice' was also tagged.

WRONG ANSWER (this computes anyone who was was *ever* tagged in same picture as Alice)

```
Answer(mid, name) :- Member(mid, name, _), Tagged(mid,
pid),
                     Member(amid, 'Alice', _),
Tagged(amid, pid)
```

We wish to check that a member never appears in a picture where an 'Alice' is not tagged

CORRECT ANSWER

```
NonAns(mid) :- Tagged(mid, pid), Member(amid, 'Alice',
_),
               !Tagged(amid, pid)
Answer(mid, name) :- Member(mid, name,
_), !NonAnswer(mid)
```

ALSO CORRECT ANSWER

```
AliceTagged(pid) :- Member(mid, 'Alice', _),
Tagged(mid, pid)
NonAns(mid) :- Tagged(mid, pid), !AliceTagged(pid)
Answer(mid, name) :- Member(mid, name,
_), !NonAnswer(mid)
```