Assignment 7 (Sample problems and solutions)

Object-Relational Database Programming

This is an assignment to test your understanding of Lecture 19 on Object-Relational Database programming.

You should submit a .sql file with the solutions and a separate .txt file with appropriate outputs.

1. **For this problem you can not use arrays.**

   Consider the relational schema `Tree(parent int, child int)` representing the schema for storing a rooted tree T.[1] A pair of nodes (m, n) is in `Tree` if m is the parent of n in T. Notice that a node m can be the parent of multiple children but a node n can have at most one parent node.

   It should be clear that for each pair of different nodes m and n in T, there is a unique shortest path of nodes $(n_1, \ldots, n_k)$ in T from m to n provided we interpret the edges in T as undirected. A good way to think about this path from a node m to a node n is to first consider the *lowest common ancestor node* of m of n in T. Then the unique path from m to n is the path that is comprised of the path up the tree from m to this common ancestor and then, from this common ancestor, the path down the tree to the node n. (Note that in this path $n_1 = m$ and $n_k = n$.)

   Define the *distance* from m to n to be $k - 1$ if $(n_1, \ldots, n_k)$ is the unique shortest path from m to n in T.

   Write a PostgreSQL function `distance(m,n)` that computes the distance in T for any possible pair m and n in T.[2]

   For example, if m is the parent of n in T then $\texttt{distance}(m, n) = 1$ because the shortest path from m to n is (m, n) which has length 1. If m is the grandparent of n in T then $\texttt{distance}(m, n) = 2$ since (m, p, n) is the path from m to n where p is the parent of m and p is a child of n. And if m and n have a common grandparent k then $distance(m, n) = distance(m, k) + distance(k, n) = 4$, etc.

[1] We assume that a tree is a connected graph with a finite number of nodes.

[2] Incidentally, if $m = n$ then $\texttt{distance}(m, n) = 0$ since, in this case, the unique path from m to n is (m) which is a path of length 0.

2. **In this problem, you can not use arrays**.

Consider the following relational schemas. (You can assume that the domain of each of the attributes in these relations is `int`.)

```
partSubpart(pid,sid,quantity)
basicPart(pid,weight)
```

A tuple (p, s, q) is in `partSubPart` if part s occurs q times as a **direct** subpart of part p. For example, think of a car c that has 4 wheels w and 1 radio r. Then $(c, w, 4)$ and $(c, r, 1)$ would be in `partSubpart`. Next think of a wheel w that has 5 bolts b. Then $(w, b, 5)$ would be in `partSubpart`.

A tuple (p, w) is in `basicPart` if basic part p has weight w. A basic part is defined as a part that does not have subparts. In other words, the pid of a basic part does not occur in the pid column of `partSubpart`.

(In the above example, a bolt and a radio would be basic parts, but car and wheel would not be basic parts.)

We define the *aggregated weight* of a part inductively as follows:

(a) If p is a basic part then its aggregated weight is its weight as given in the `basicPart` relation

(b) If p is not a basic part, then its aggregated weight is the sum of the aggregated weights of its subparts, each multiplied by the quantity with which these subparts occur in the `partSubpart` relation:

Example: The following example is based on a desk lamp with `pid` 1. Suppose a desk lamp consists of 4 bulbs (with `pid` 2) and a frame (with `pid` 3), and a frame consists of a post (with `pid` 4) and 2 switches (with `pid` 5). Furthermore, we will assume that the weight of a bulb is 5, that of a post is 50, and that of a switch is 3.

Then the `partSubpart` and `basicPart` relation would be as follows:

partSubPart

pid	sid	quantity
1	2	4
1	3	1
3	4	1
3	5	2

basicPart

pid	weight
2	5
4	50
5	3

Then the aggregated weight of a lamp is $4 \times 5 + 1 \times (1 \times 50 + 2 \times 3) = 76$.

Write a PostgreSQL function `aggregatedWeight(p integer)` that returns the aggregated weight of a part `p`.

(a) A problem very similar to the aggregated weight problem is described in the PostgreSQL manual page

`https://www.postgresql.org/docs/8.4/queries-with.html`

Adapt the recursive program in that manual page to the aggregrated weight problem.

(b) Write a non-recursive version for aggregate weight problem. Obviously, you program will require looping statements.

3. **In this problem, you can use arrays, but only as a mechanism to represents subsets of `A(x)`.**

   Consider the relation schema `A(x)` representing a schema for storing a set A. (You can assume that the domain of attribute `x` is integer.)

   Using arrays to represent sets, write a PostgreSQL program

   `superSetsOfSet(X int[])`

   that returns a relation that contains each subset of A that is a superset of X, i.e., each set Y such that $X \subseteq Y \subseteq A$.

   For example, if $X = \{\}$, then `superSetsofSets(X)` should return the relation that consist of each element in the powerset of A.

4. **In this problem, you can use arrays, but only as a mechanism to represents sets of words**.

   Consider the relation schema `document(doc int, words text[])` representing a relation of pairs (d, W) where d is a unique document id and W denotes the set of words that occur in d.

   Let $\mathbf{W}$ denote the set of all words that occur in the documents and let t be a positive integer denoting a *threshold*.

   Let $X \subseteq \mathbf{W}$. We say that X is t-frequent if

   $$\texttt{count}(\{d|(d, W) \in \texttt{document} \text{ and } X \subseteq W\}) \geq t$$

   In other words, X is *t-frequent* if there are at least t documents that contain all the words in X.

   Write a PostgreSQL program `frequentSets(t int)` that returns the relation of all t-frequent sets.

   In a good solution for this problem, you should use the following rule: if X is not t-frequent then any set Y such that $X \subseteq Y$ is not t-frequent either. In the literature, this is called the *Apriori* rule of the frequent itemset mining problem. This rule allows you to avoid examing supersets of sets that are not frequent. This can drastically reduce the search space.