Question ID: 565

Question Text: Consider a relation Parent(par,ch), where a tuple (p,c) in Parent specifies that person p is the parent of person c. The only key for Parent consists of both attributes together. We are interested in writing a recursive query to find all descendants of the person named "Eve." Below you see six WITH-statements, each defining a recursive relation Ancestor(X,Y).

```
1) WITH RECURSIVE Ancestor(X,Y) AS
     ( (SELECT par,ch FROM Parent)
        UNION
       (SELECT Ancestor.X, Parent.ch
        FROM Ancestor, Parent
        WHERE Ancestor.Y=Parent.par) )

2) WITH RECURSIVE Ancestor(X,Y) AS
     ( (SELECT par,ch FROM Parent WHERE par='Eve')
        UNION
       (SELECT Ancestor.X, Parent.ch
        FROM Ancestor, Parent
        WHERE Ancestor.Y=Parent.par) )

3) WITH RECURSIVE Ancestor(X,Y) AS
     ( (SELECT par,ch FROM Parent)
        UNION
       (SELECT Parent.par, Ancestor.Y
        FROM Parent, Ancestor
        WHERE Parent.ch = Ancestor.X) )

4) WITH RECURSIVE Ancestor(X,Y) AS
     ( (SELECT par,ch FROM Parent)
        UNION
       (SELECT Parent.par, Ancestor.Y
        FROM Parent, Ancestor
        WHERE Parent.ch = Ancestor.X and
Parent.par='Eve') )

5) WITH RECURSIVE Ancestor(X,Y) AS
     ( (SELECT par,ch FROM Parent WHERE par='Eve')
        UNION
       (SELECT A1.X, A2.Y
        FROM Ancestor A1, Ancestor A2
        WHERE A1.Y = A2.X) )

6) WITH RECURSIVE Ancestor(X,Y) AS
     ( (SELECT par,ch FROM Parent)
        UNION
       (SELECT A1.X, A2.Y
        FROM Ancestor A1, Ancestor A2
        WHERE A1.Y = A2.X and A1.X='Eve') )
```

Any one of these WITH-statements could be followed by any query about the relation Ancestor. We shall consider two possible queries, with which to complete one of the WITH-statements (1)

through (6):

```
A: SELECT Y FROM Ancestor;

B: SELECT Y FROM Ancestor WHERE X='Eve';
```

You should determine which of the six WITH-statements above, when followed by A or B, or both, will produce the correct answer: exactly those people who are descendants of Eve. Then, select the true statement from the list below. Note: do not concern yourself with the fact that SQL recursion is only required to be implemented for linear recursions, while several of the WITH-statements use nonlinear recursion.

Correct Choice 1:	(B) works but (A) does not for exactly (1), (3), and (6).
Choice Explanation:	**NONE**
Correct Choice 2:	(A) works only with (2).
Choice Explanation:	**NONE**
Correct Choice 3:	Neither (A) nor (B) works on exactly (4) and (5).
Choice Explanation:	**NONE**

Question Explanation: Statement (1) defines Ancestor to be all the true ancestor facts. If we follow it by query (A), we get all the people who are descendants of anybody, not just Eve. However, (B) gives us the correct answer, since it restricts to those Ancestor tuples where the ancestor is Eve.

Statement (2) is almost like (1), but the basis case, where we initialize Ancestor, restricts so the first component is Eve. Notice that the recursion never allows the first component of Ancestor to be anything but Eve, while the second component can take on any descentant of Eve, as we recursively add tuples following the Parent information from generation to generation. As a result, either query (A) or (B) is acceptable.

Statement (3) is like (1), but the recursion uses Parent on the left instead of the right. The same conclusion holds for (3) as for (1).

Statement (4) looks similar to (3) but makes a serious mistake. Although the basis allows all Parent facts to be Ancestor facts, every other use of Parent requires the parent to be Eve. As a result, assuming Eve has no parents, we'll get at most the

grandchildren of Eve. Thus, neither (A) nor (B) works.

Statement (5) makes a mistake similar to (4). When we try to apply the recursion for the first time, we cannot infer any new facts unless Eve is both a parent and a child. Thus, neither (A) nor (B) will work.

Statement (6) fixes the problem of (5) in a subtle way. It is true that every recursively inferred Ancestor fact will have first component Eve. However, the basis at least allows all Parent facts into Ancestor, so the recursion preceeds one generation at a time, like Query (1). As a result, Ancestor winds up holding all parent facts, and all multigeneration ancestor facts where the first component is Eve. Thus, (B) will surely work. (A) does not work, because some of the Parent facts may well have first components other than Eve.

Question Text: Consider relation R(a) containing a set of positive integers with no duplicates, and the following recursive SQL query (do not be concerned that in the SQL definition, nonlinear recursions such as this one are not required to be implemented):

```
WITH RECURSIVE Mystery(X,Y) AS
   (SELECT A AS X, A AS Y FROM R)
    UNION
   (SELECT m1.X, m2.Y
    FROM Mystery m1, Mystery m2
    WHERE m2.X = m1.Y + 1)
SELECT MAX(Y-X) + 1 FROM Mystery;
```

While the definition looks complicated, Mystery in fact computes a property of R that can be stated very succinctly. Try to determine what Mystery is computing about R.

Question Explanation: Mystery contains all those tuples (x,y) such that x <= y, and R contains all the integers from x to y, inclusive. To see why, there are two easy inductions to prove:

1. If Mystery contains (x,y), then R contains x through y, inclusive.
2. If x, x+1,..., y are all in R, then (x,y) will eventually be placed in Mystery.

Then, the query selects one more than the largest difference y-x for any (x,y) in Mystery. As a result, what is returned is the length of the longest consecutive sequence of integers in R.