Recursive query examples

Problem 1 Consider a database for metro and bus lines, consisting of two relations

<table>
<thead>
<tr>
<th>Metro</th>
<th>station</th>
<th>next-station</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus</td>
<td>station</td>
<td>next-station</td>
</tr>
</tbody>
</table>

Write a query to find the pairs of stations \((a, b)\) such that \(b\) can be reached from \(a\) by some combination of metro and bus, but not by metro or bus alone.

The plan is the following: (i) write a view Combination defining the pairs of stations \((a, b)\) such that \(b\) can be reached from \(a\) by some combination of metro and bus, (ii) write views TMetro and TBus defining the transitive closures of Metro and Bus (iii) compute the answer from Combination, TMetro, and TBus (Answer = Combination \(\setminus\) (TMetro \(\cup\) TBus)).

create recursive view Combination as
(select * from Metro union select * from Bus)
union
select x.station, y.next-station
from Combination x, Combination y
where x.next-station = y.station

create recursive view TMetro as
select * from Metro
union
select x.station, y.next-station
from TMetro x, TMetro y
where x.next-station = y.station

create recursive view TBus as
select * from Bus
union
select x.station, y.next-station
from TBus x, TBus y
where x.next-station = y.station

select * from Combination
except
(select * from TMetro) union (select * from TBus)
**Problem 2** Consider a database consisting of the following relations:

<table>
<thead>
<tr>
<th>Node</th>
<th>id</th>
<th>Flow</th>
<th>from</th>
<th>to</th>
</tr>
</thead>
</table>

*Node* provides a set of ids representing data sources in a network. *Flow* consists of pairs of \((a, b)\) of nodes such that \(b\) used data from \(a\). A data source is an *authority* if it does not use data from any other source (i.e., it has indegree zero in Flow). We say that a data source is *trusted* if it is an authority or it only used data from other trusted nodes.

For example, on the instance

<table>
<thead>
<tr>
<th>Node</th>
<th>id</th>
<th>Flow</th>
<th>from</th>
<th>to</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>1 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>2 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>3 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>4 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>3 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>5 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>5 7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

the authorities are nodes 1 and 2, and the set of trusted nodes is

<table>
<thead>
<tr>
<th>Trusted</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>7</td>
</tr>
</tbody>
</table>

Write a recursive SQL query that computes the trusted nodes.
Solution

create recursive view Trusted as
select n.id from Node n
where not exists
    (select * from Flow
    where Flow.to = n.id and
    Flow.from not in (select * from Trusted ))

Observe that the first iteration initializes Trusted to the set of authorities (nodes with in-degree zero in Flow). In this example there is no need for a separate initialization component of the query.

Is the following also a correct solution (explain):

create recursive view Authority as
select n.id from Node n
where n.id not in (select to from Flow)

create recursive view Trusted as
select * from Authority
union
select f.to as id from Flow f, Trusted t
where f.from = t.id

Problem 3 Consider a database consisting of two relations

<table>
<thead>
<tr>
<th>Left</th>
<th>parent</th>
<th>child</th>
</tr>
</thead>
<tbody>
<tr>
<td>Right</td>
<td>parent</td>
<td>child</td>
</tr>
</tbody>
</table>

Each instance of the database represents a binary tree with a single root, in which each node that is not a leaf has a left and right child. Write a recursive query defining a relation

| Before | id1 | id2 |

containing all pairs of nodes \((a, b)\) such that \(a\) appears before \(b\) in the depth-first traversal of the tree (where left children are visited before right children).
Solution

create view Edge as
select parent as id1, child as id2 from Left
union
select parent as id1, child as id2 from Right

create recursive view Descendant as
select * from Edge
union
select e.id1, d.id2 from Edge e, Descendant d
where e.id2 = d.id1

select * from Descendant
union
select l.child as id1, r.child as id2
from Left l, Right r
where l.parent = r.parent
union
select x.id2 as id1, y.id2
from Left l, Right r, Descendant x, Descendant y
where l.parent = r.parent and x.id1 = l.child and y.id1 = r.child