SQL: Recursion

Recursion in SQL

- SQL2 had no recursion
  - You can find Bart’s parents, grandparents, great grandparents, etc.
    
    ```sql
    SELECT p1.parent AS grandparent
    FROM Parent p1, Parent p2
    WHERE p1.child = p2.parent
    AND p2.child = 'Bart';
    ```
  - But you cannot find all his ancestors with a single query
- SQL3 introduces recursion
  - WITH clause
  - Implemented in DB2 (called common table expressions)

Ancestor query in SQL3

```sql
WITH Ancestor(anc, desc) AS
  ((SELECT parent, child FROM Parent)
  UNION
  (SELECT a1.anc, a2.desc
   FROM Ancestor a1, Ancestor a2
   WHERE a1.desc = a2.anc))

SELECT anc
FROM Ancestor
WHERE desc = 'Bart';
```

Fixed point of a function

- If $f : T \rightarrow T$ is a function from a type $T$ to itself, a fixed point of $f$ is a value $x$ such that $f(x) = x$
- Example: What is the fixed point of $f(x) = x / 2$?
  - 0, because $f(0) = 0 / 2 = 0$
- To compute a fixed point of $f$
  - Start with a “seed”: $x \leftarrow x_0$
  - Compute $f(x)$
    - If $f(x) = x$, stop; $x$ is fixed point of $f$
    - Otherwise, $x \leftarrow f(x)$; repeat
  - Example: compute the fixed point of $f(x) = x / 2$
    - With seed 1: 1, 1/2, 1/4, 1/8, 1/16, … → 0

Fixed point of a query

- A query $q$ is just a function that maps an input table to an output table, so a fixed point of $q$ is a table $T$ such that $q(T) = T$
- To compute fixed point of $q$
  - Start with an empty table: $T \leftarrow \emptyset$
  - Evaluate $q$ over $T$
    - If the result is identical to $T$, stop; $T$ is a fixed point
    - Otherwise, let $T$ be the new result; repeat
  - Starting from $\emptyset$ produces the unique minimal fixed point (assuming $q$ is monotone)
Finding ancestors

WITH Ancestor(anc, desc) AS
(SELECT parent, child FROM Parent)
UNION
(SELECT a1.anc, a2.desc
FROM Ancestor a1, Ancestor a2
WHERE a1.desc = a2.anc)

Think of it as
\[
\text{Ancestor} = \text{q(Ancestor)}
\]

<table>
<thead>
<tr>
<th>Parent</th>
<th>Child</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homer</td>
<td>Bart</td>
</tr>
<tr>
<td>Homer</td>
<td>Lisa</td>
</tr>
<tr>
<td>Marge</td>
<td>Bart</td>
</tr>
<tr>
<td>Marge</td>
<td>Lisa</td>
</tr>
<tr>
<td>Abe</td>
<td>Homer</td>
</tr>
<tr>
<td>Ape</td>
<td>Abe</td>
</tr>
</tbody>
</table>

Intuition behind fixed-point iteration

- Initially, we know nothing about ancestor-descendent relationships
- In the first step, we deduce that parents and children form ancestor-descendent relationships
- In each subsequent step, we use the facts deduced in previous steps to get more ancestor-descendent relationships
- We stop when no new facts can be proven

Mutual recursion example

- Table Natural(n) contains 1, 2, …, 100
- Which numbers are even/odd?
  - An odd number plus 1 is an even number
  - An even number plus 1 is an odd number
  - 1 is an odd number

WITH Even(n) AS
(SELECT n FROM Natural
WHERE n = ANY(SELECT n+1 FROM Odd)),
Odd(n) AS
(SELECT n FROM Natural
WHERE n = 1)
UNION
(SELECT n FROM Natural
WHERE n = ANY(SELECT n+1 FROM Even))
Mixing negation with recursion

- If \( q \) is non-monotone
  - The fixed-point iteration may flip-flop and never converge
  - There could be multiple minimal fixed points—so which one is the right answer?
- Example: reward students with GPA higher than 3.9
  - Those not on the Dean’s List should get a scholarship
  - Those without scholarships should be on the Dean’s List

\[
\text{WITH Scholarship(SID) AS} \\
(\text{SELECT SID FROM Student WHERE GPA > 3.9} \\
\text{AND SID NOT IN (SELECT SID FROM DeansList)}), \\
\text{DeansList(SID) AS} \\
(\text{SELECT SID FROM Student WHERE GPA > 3.9} \\
\text{AND SID NOT IN (SELECT SID FROM Scholarship)})
\]

Multiple minimal fixed points

\[
\text{WITH Scholarship(SID) AS} \\
(\text{SELECT SID FROM Student WHERE GPA > 3.9} \\
\text{AND SID NOT IN (SELECT SID FROM DeansList)}), \\
\text{DeansList(SID) AS} \\
(\text{SELECT SID FROM Student WHERE GPA > 3.9} \\
\text{AND SID NOT IN (SELECT SID FROM Scholarship)})
\]

Legal mix of negation and recursion

- Construct a dependency graph
  - One node for each table defined in WITH
  - A directed edge \( R \rightarrow S \) if \( R \) is defined in terms of \( S \)
  - Label the directed edge \("-"\) if the query defining \( R \) is not monotone with respect to \( S \)
- Legal SQL3 recursion: no cycle containing a \("-"\) edge
- Called stratified negation
- Bad mix: a cycle with at least one edge labeled \("-"\)

Evaluating stratified negation

- The stratum of a node \( R \) is the maximum number of \("-"\) edges on any path from \( R \) in the dependency graph
  - \( \text{Ancestor} \): stratum 0
  - \( \text{Person} \): stratum 0
  - \( \text{NoCommonAnc} \): stratum 1
- Evaluation strategy
  - Compute tables lowest-stratum first
  - For each stratum, use fixed-point iteration on all nodes in that stratum
    - Stratum 0: \( \text{Ancestor} \) and \( \text{Person} \)
    - Stratum 1: \( \text{NoCommonAnc} \)
  - Intuitively, there is no negation within each stratum
Summary

- SQL3 WITH recursive queries
- Solution to a recursive query (with no negation): unique minimal fixed point
- Computing unique minimal fixed point: fixed-point iteration starting from \( \emptyset \)
- Mixing negation and recursion is tricky
  - Illegal mix: fixed-point iteration may not converge; there may be multiple minimal fixed points
  - Legal mix: stratified negation (compute by fixed-point iteration stratum by stratum)