SQL: Recursion

Introduction to Databases
CompSci 316 Fall 2022
Announcements (Thu., Nov 30)

• Gradiance due Friday 12/2 10 pm
• Work on your projects – check out Ed post
• Check out practice problems and exams on Sakai
http://xkcdsw.com/1105
A motivating example

Example: find Bart’s ancestors

“Ancestor” has a recursive definition

- $X$ is $Y$’s ancestor if
  - $X$ is $Y$’s parent, or
  - $X$ is $Z$’s ancestor and $Z$ is $Y$’s ancestor
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 PostgreSQL (common table expressions)
WITH RECURSIVE
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';
Finding ancestors

- WITH RECURSIVE 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 the definition as Ancestor = q(Ancestor)

- “Fixed point”
- Start with Ancestor₀ = ∅
- Apply the query Q again and again, i.e.,
  Q(Ancestorᵀ₋₁) = Ancestorᵀ
- Until Q(Ancestorᵀ) = Ancestorᵀ
  i.e., no change
- If Q is monotone, unique fixpoint
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 steps, we use the facts deduced in previous steps to get more ancestor-descendent relationships

• We stop when no new facts can be proven
Linear recursion

- With linear recursion, a recursive definition can make only one reference to itself
- Non-linear
  - WITH RECURSIVE 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))

- Linear
  - WITH RECURSIVE Ancestor(anc, desc) AS
    ((SELECT parent, child FROM Parent)
     UNION
     (SELECT anc, child
      FROM Ancestor, Parent
      WHERE desc = parent))

Gives the same answer
Linear vs. non-linear recursion

• Linear recursion is easier to implement
  • For linear recursion, just keep joining newly generated Ancestor rows with Parent
  • For non-linear recursion, need to join newly generated Ancestor rows with all existing Ancestor rows

• Non-linear recursion may take fewer steps to converge, but perform more work
  • Example: $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e$
  • Linear recursion takes 4 steps
  • Non-linear recursion takes 3 steps
    • More work: e.g., $a \rightarrow d$ has two different derivations
Mutual recursion example

• Table $\textit{Natural}(n)$ 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 RECURSIVE $\textit{Even}(n)$ AS
  (SELECT n FROM Natural
   WHERE n = ANY(SELECT n+1 FROM $\textit{Odd}$)),

RECURSIVE $\textit{Odd}(n)$ AS
  ((SELECT n FROM Natural WHERE n = 1)
   UNION
  (SELECT n FROM Natural
   WHERE n = ANY(SELECT n+1 FROM $\textit{Even}$)))

Order does not matter
Always look at the states of all tables from last step

Base case
Step 0: Odd = {}, Even = {}
Step 1: Odd = {1}, Even = {}
Step 2: Odd = {1}, Even = {2}
Step 3: Odd = {1, 3}, Even = {2}
Step 4: Odd = {1, 3}, Even = {2, 4}
….
…. 
Step 100: Odd = {1, 3, …, 99},
          Even = {2, 4, …, 100}
Step 101: = Step 100
Semantics of WITH

- WITH RECURSIVE $R_1$ AS $Q_1$, ..., RECURSIVE $R_n$ AS $Q_n$

- $Q$;
  - $Q$ and $Q_1$, ..., $Q_n$ may refer to $R_1$, ..., $R_n$

- Semantics
  1. $R_1 \leftarrow \emptyset, \ldots, R_n \leftarrow \emptyset$
  2. Evaluate $Q_1$, ..., $Q_n$ using the current contents of $R_1$, ..., $R_n$:
     
     $R_1^{new} \leftarrow Q_1$, ..., $R_n^{new} \leftarrow Q_n$
    
  3. If $R_i^{new} \neq R_i$ for some $i$
     
     3.1. $R_1 \leftarrow R_1^{new}, \ldots, R_n \leftarrow R_n^{new}$
     
     3.2. Go to 2.
  
  4. Compute $Q$ using the current contents of $R_1$, ... $R_n$ and output the result
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—we wouldn’t know which one to pick as answer!

• Example: all users join either Jessica’s Circle or Tommy’s
  • Those not in Jessica’s Circle should be in Tom’s
  • Those not in Tom’s Circle should be in Jessica’s
  • WITH RECURSIVE TommyCircle(uid) AS
    (SELECT uid FROM User WHERE
     uid NOT IN (SELECT uid FROM JessicaCircle)),
  RECURSIVE JessicaCircle(uid) AS
    (SELECT uid FROM User WHERE
     uid NOT IN (SELECT uid FROM TommyCircle))
Fixed-point iter may not converge

WITH RECURSIVE TommyCircle(uid) AS
(SELECT uid FROM User WHERE
uid NOT IN (SELECT uid FROM JessicaCircle)),
RECURSIVE JessicaCircle(uid) AS
(SELECT uid FROM User WHERE
uid NOT IN (SELECT uid FROM TommyCircle))
Multiple minimal fixed points

WITH RECURSIVE TommyCircle(uid) AS
(SELECT uid FROM User WHERE
  uid NOT IN (SELECT uid FROM JessicaCircle)),
RECURSIVE JessicaCircle(uid) AS
(SELECT uid FROM User WHERE
  uid NOT IN (SELECT uid FROM TommyCircle))

<table>
<thead>
<tr>
<th>uid</th>
<th>name</th>
<th>age</th>
<th>pop</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td>10</td>
<td>0.9</td>
</tr>
<tr>
<td>121</td>
<td>Allison</td>
<td>8</td>
<td>0.85</td>
</tr>
</tbody>
</table>

Bad query!
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 with a “$-$” edge**
  • Called **stratified negation**

• Bad mix: a cycle with at least one edge labeled “$-$”
  
  ![Diagram](attachment:diagram.png)
Stratified negation example

- Find pairs of persons with no common ancestors
- Input: Parent(parent, child)

WITH RECURSIVE 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)),

Person(person) AS
((SELECT parent FROM Parent) UNION
(SELECT child FROM Parent)),

NoCommonAnc(person1, person2) AS
((SELECT p1.person, p2.person
   FROM Person p1, Person p2
   WHERE p1.person <> p2.person)
EXCEPT
(SELECT a1.desc, a2.desc
   FROM Ancestor a1, Ancestor a2
   WHERE a1.anc = a2.anc))

SELECT * FROM NoCommonAnc;
Evaluating stratified negation

• The **stratum** of a node $R$ is the maximum number of “—” edges on any path from $R$ in the dependency graph
  • Ancestor: stratum 0
  • Person: stratum 0
  • 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: Ancestor and Person
    • Stratum 1: NoCommonAnc

☞ Intuitively, there is no negation within each stratum
Practice problem: Recursion

• What does this query compute?
• Input: Edge(start, end) denoting directed edges in a graph from u to v. Assume nodes take integer values.

• WITH RECURSIVE Mystery(x, y) AS
  (SELECT start, end FROM Edge)
  UNION
  (SELECT a1.x, a3.end FROM Mystery a1, Edge a2, Edge a3
   WHERE a1.y = a2.start and a2.end=a3.start)
SELECT y FROM Mystery m1, Mystery m2
WHERE m1.y = m2.x AND m1.x = 5 AND m2.y = 5
Practice problem: Recursion w/ Negation

• Input: \texttt{Edge(start, end)} denoting directed edges in a graph from \texttt{u} to \texttt{v}. Assume nodes take integer values.

Write a query to compute pairs of nodes \((x, y)\) such that there are no paths from \(x\) to \(y\)
Practice problem: Recursion w/ Negation

• Input: $\text{Edge}(\text{start, end})$ denoting directed edges in a graph from u to v. Assume nodes take integer values.

Write a query to compute pairs of nodes $(x, y)$ such that there are no paths from $x$ to $y$
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)