SQL: Recursion

CPS 116
Introduction to Database Systems

Announcements (Thu. Sep. 17)

- Homework #2 due in 1½ weeks
  - Start now, if you haven’t already
- Homework #1 sample solution available
- Project milestone #1 due in 3 weeks
  - Come to my office hours and chat
- Midterm in 2 weeks in class

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 DB2 (called common table expressions)

Ancestor query in SQL3

```sql
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';
```

How do we compute such a recursive query?

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
  - \( q \) starting from \( \emptyset \) produces the unique minimal fixed point (assuming \( q \) is monotone)

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 it as \( \text{Ancestor} = q(\text{Ancestor}) \)

Intuition behind fixed-point iteration

- Initially, we know nothing about ancestor-descendant relationships
- In the first step, we deduce that parents and children form ancestor-descendant relationships
- In each subsequent steps, we use the facts deduced in previous steps to get more ancestor-descendant 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
  UNION
  (SELECT a1.anc, a2.desc
  FROM Ancestor a1, Ancestor a2
  WHERE a1.desc = a2.anc))
- Linear:
  WITH RECURSIVE Ancestor(anc, desc) AS

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: \(a \to b \to c \to d \to e\)
  - Linear recursion takes 4 steps
  - Non-linear recursion takes 3 steps
  - More work: e.g., \(a \to d\) has two different derivations

Mutual recursion example

- Table Natural \((n)\) contains 1, 2, \ldots, 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 Even(n) AS
    (SELECT n FROM Natural
     WHERE n = ANY(SELECT n+1 FROM Odd)),
  RECURSIVE Odd(n) AS
    ((SELECT n FROM Natural WHERE n = 1)
    UNION
    (SELECT n FROM Natural
     WHERE n = ANY(SELECT n+1 FROM Even)))
Operational semantics of \textsc{WITH}

\begin{itemize}
  \item \textsc{WITH RECURSIVE} $R_1$ \textsc{AS} $Q_1$, \ldots, \textsc{RECURSIVE} $R_n$ \textsc{AS} $Q_n$;
  \item $Q_1$, \ldots, $Q_n$ may refer to $R_1$, \ldots, $R_n$.
\end{itemize}

Operational semantics

1. $R_i \leftarrow \emptyset$, \ldots, $R_n \leftarrow \emptyset$
2. Evaluate $Q_1$, \ldots, $Q_n$ using the current contents of $R_1$, \ldots, $R_n$;
   \hspace{1em} $R_{i, \text{new}} \leftarrow Q_i$, \ldots, $R_{n, \text{new}} \leftarrow Q_n$.
3. If $R_{i, \text{new}} \neq R_i$ for any $i$
   \hspace{1em} 3.1. $R_i \leftarrow R_{i, \text{new}}$, \ldots, $R_n \leftarrow R_{n, \text{new}}$
   \hspace{1em} 3.2. Go to 2.
4. Compute $Q$ using the current contents of $R_1$, \ldots, $R_n$ and output the result.

Computing mutual recursion

\begin{verbatim}
WITH RECURSIVE Even(n) AS
  (SELECT n FROM Natural WHERE n = ANY(SELECT n+1 FROM Odd))
  UNION
  ((SELECT n FROM Natural WHERE n = 1)
    UNION
    (SELECT n FROM Natural WHERE n = ANY(SELECT n+1 FROM Even)))
  \textsc{RECURSIVE} Odd(n) AS
  ((SELECT n FROM Natural WHERE n = 1)
    UNION
    (SELECT n FROM Natural WHERE n = ANY(SELECT n+1 FROM Even)))
\end{verbatim}

Even = $\emptyset$, Odd = $\emptyset$
Even = $\emptyset$, Odd = \{1\}
Even = \{2\}, Odd = \{1\}
Even = \{2\}, Odd = \{1, 3\}
Even = \{2, 4\}, Odd = \{1, 3\}
Even = \{2, 4\}, Odd = \{1, 3, 5\}
\ldots

Fixed points are not unique

\begin{verbatim}
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))
\end{verbatim}

\begin{itemize}
  \item There may be many other fixed points
  \item But if $q$ is monotone, then all these fixed points must contain the fixed point we computed from fixed-point iteration starting with $\emptyset$
  \hspace{1em} Thus the unique minimal fixed point is the "natural" answer to the query
\end{itemize}

Note that the bogus tuple reinforces itself!
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
  - WITH RECURSIVE Scholarship(SID) AS
    (SELECT SID FROM Student WHERE GPA > 3.9 AND SID NOT IN (SELECT SID FROM DeansList)),
    RECURSIVE DeansList(SID) AS
    (SELECT SID FROM Student WHERE GPA > 3.9 AND SID NOT IN (SELECT SID FROM Scholarship))

Fixed-point iteration does not converge

WITH RECURSIVE Scholarship(SID) AS
(SELECT SID FROM Student WHERE GPA > 3.9 AND SID NOT IN (SELECT SID FROM DeansList)),
RECURSIVE DeansList(SID) AS
(SELECT SID FROM Student WHERE GPA > 3.9 AND SID NOT IN (SELECT SID FROM Scholarship))

Multiple minimal fixed points

WITH RECURSIVE Scholarship(SID) AS
(SELECT SID FROM Student WHERE GPA > 3.9 AND SID NOT IN (SELECT SID FROM DeansList)),
RECURSIVE DeansList(SID) AS
(SELECT SID FROM Student WHERE GPA > 3.9 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 “−”

Stratified negation example

- Find pairs of persons with no common ancestors

```sql
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 t FROM P t) UNION
   Ancestor
   ((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
  - \( \text{Ancestor} \): 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)