SQL: Recursion

CPS 196.3
Introduction to Database Systems

Announcement

- Graded Homework #1 in my office D327
  - Sample solution in handout box
- Homework #2 due next Monday, September 29
- Course project milestone 1 due next Wednesday, October 1
  - Come talk to me if you need help deciding

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.
  - 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

```
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';
```

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 $f(x) = x / 2$?

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:
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

\[
\begin{align*}
\text{WITH Ancestor(anc, desc) AS} \\
&\quad (\text{SELECT parent, child FROM Parent}) \\
&\quad \text{UNION} \\
&\quad (\text{SELECT a1.anc, a2.desc FROM Ancestor a1, Ancestor a2 WHERE a1.desc = a2.anc})
\end{align*}
\]

- Think of it as $\text{Ancestor} = q(\text{Ancestor})$

\[\begin{align*}
\text{anc} & \quad \text{desc} \\
\text{Homer Bart} & \quad \text{Homer Lisa} \\
\text{Marge Bart} & \quad \text{Marge Lisa} \\
\text{Abe Homer} & \quad \text{Ape Abe} \\
\text{Abe Bart} & \quad \text{Ape Homer} \\
\text{Abe Lisa} & \quad \text{Ape Lisa} \\
\text{Ape Bart} & \quad \text{Ape Bart} \\
\text{Ape Lisa} & \quad \text{Ape Lisa}
\end{align*}\n
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:
  ```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))
  ```
- Linear:
  ```sql
  WITH Ancestor(anc, desc) AS
  ((SELECT parent, child FROM Parent)
  UNION
  ( ))
  ```

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
  - Example: $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e$
  - Linear recursion takes 4 steps
  - Non-linear recursion takes 3 steps

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
  ```sql
  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)))
  ```
Operational semantics of WITH

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

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

- Operational semantics
  1. $R_1 ← \emptyset$, ..., $R_n ← \emptyset$
  2. Evaluate $Q_1$, ..., $Q_n$ using the current contents of $R_1$, ..., $R_n$:
     - $R_1^{\text{new}} ← Q_1$, ..., $R_n^{\text{new}} ← Q_n$
  3. If $R_i^{\text{new}} ≠ R_i$ for any $i$
     3.1. $R_i ← R_i^{\text{new}}$, ..., $R_n ← R_n^{\text{new}}$
     3.2. Go to 2.
  4. Compute $Q$ using the current contents of $R_1$, ..., $R_n$ and output the result

Computing mutual recursion

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))

- Even = ∅, Odd = ∅

- ...

Fixed points are not unique

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)

- There may be many other fixed points
- But if $q$ is monotone, then all these fixed points must contain the fixed point we computed from fixed-point iteration starting with $\emptyset$
  - Thus the unique minimal fixed point is the "natural" answer to the query
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 no on the Dean’s List should get a scholarship
  - Those without scholarships should be on the Dean’s List
- WITH Scholarship(SID) AS
  (SELECT SID FROM Student WHERE GPA > 3.9 AND SID NOT IN (SELECT SID FROM DeansList)),
  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 Scholarship(SID) AS
(SELECT SID FROM Student WHERE GPA > 3.9 AND SID NOT IN (SELECT SID FROM DeansList)),
DeansList(SID) AS
(SELECT SID FROM Student WHERE GPA > 3.9 AND SID NOT IN (SELECT SID FROM Scholarship))

Multiple minimal fixed points

WITH Scholarship(SID) AS
(SELECT SID FROM Student WHERE GPA > 3.9 AND SID NOT IN (SELECT SID FROM DeansList)),
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 \( R \to S \) 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 SQL-3 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 common ancestors

\[
\begin{align*}
\text{WITH Ancestor(anc, desc) AS} \\
&= ((\text{SELECT parent, child FROM Parent}) \cup \\
&\quad (\text{SELECT a1.anc, a2.desc FROM Ancestor a1, Ancestor a2}
\quad \text{WHERE } a1.desc = a2.anc)), \\
\text{Person(person) AS} \\
&= ((\text{SELECT parent FROM Parent}) \cup \\
&\quad (\text{SELECT child FROM Parent})), \\
\text{NoCommonAnc(person1, person2) AS} \\
&= ((\text{SELECT p1.person, p2.person FROM Person p1, Person p2}
\quad \text{WHERE } p1.person <> p2.person)
\quad \text{EXCEPT} \\
&\quad (\text{SELECT a1.desc, a2.desc FROM Ancestor a1, Ancestor a2}
\quad \text{WHERE } a1.anc = a2.anc))
\end{align*}
\]

\[
\text{SELECT * FROM NoCommonAnc;}
\]

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

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
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)