SQL: Recursion
Introduction to Databases
CompSci 316 Fall 2021

Announcements (Tue., Sep. 28)
• Homework 1 graded
• We are still working on Project Milestone 1 submissions
• Homework 2, Gradiance SQL constraints due today
  • Tomorrow morning:
    • Homework 2 sample solution will be posted on Sakai
    • Old Gradiance exercises will reopen
    • We will take a snapshot of the scores before that
• My office hours today extended to 5pm, but I need to step out 3:30-4pm
• Gradiance SQL recursion assigned

Announcements cont’d
• Thursday: Midterm
  • In-class, open-book, open-notes
  • Questions on paper, but you will need a computer to submit—make sure it has enough battery!
  • Sample midterm/solution posted on Sakai
  • NEW: It will NOT cover materials discussed today
  • Triggers/views are fair game, but coverage will be light (if any)
• Thursday after fall break
  • Project Milestone 2 due
  • Gradiance SQL Triggers/Views due (extended)
  • Gradiance SQL recursion due
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 pl.parent AS grandparent
    FROM Parent pl, Parent p2
    WHERE pl.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)
Ancestor query in SQL3

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';

Fixed point of a function

• If $f : D \rightarrow D$ is a function from a type $D$ 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 a fixed point of $f$
    • Otherwise, $x \leftarrow f(x)$; repeat
• Example: compute the fixed point of $f(x) = x/2$
  • With seed: 1, 1/2, 1/4, 1/8, 1/16, … → 0
• Does not always work, but happens to work for us!

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 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)

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))
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 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 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)))
Semantics of \textsc{WITH}

- \textsc{WITH RECURSIVE }\mathcal{R}_1 \text{ AS } \mathcal{Q}_1, \ldots, \\
  \text{RECURSIVE } \mathcal{R}_n \text{ AS } \mathcal{Q}_n \\
  \mathcal{Q};

  - \mathcal{Q} and \mathcal{Q}_1, \ldots, \mathcal{Q}_n may refer to \mathcal{R}_1, \ldots, \mathcal{R}_n

- Semantics

  1. \( \mathcal{R}_1 \leftarrow \emptyset, \ldots, \mathcal{R}_n \leftarrow \emptyset \)
  2. Evaluate \( \mathcal{Q}_1, \ldots, \mathcal{Q}_n \) using the current contents of \( \mathcal{R}_1, \ldots, \mathcal{R}_n \):
     
     - \( \mathcal{R}_{\text{new}} \leftarrow \mathcal{Q}_1, \ldots, \mathcal{R}_{\text{new}} \leftarrow \mathcal{Q}_n \)
  3. If \( \mathcal{R}_{\text{new}} \neq \mathcal{R}_i \) for some \( i \)
     
     - 3.1. \( \mathcal{R}_i \leftarrow \mathcal{R}_{\text{new}}, \ldots, \mathcal{R}_n \leftarrow \mathcal{R}_{\text{new}} \)
     
     - 3.2. Go to 2.
  4. Compute \( \mathcal{Q} \) using the current contents of \( \mathcal{R}_1, \ldots, \mathcal{R}_n \)
  and output the result

Computing mutual recursion

\textsc{WITH RECURSIVE} \textsc{Even}(n) \text{ AS } \\
(\text{SELECT } n \text{ FROM } \textsc{Natural} \\
\text{WHERE } n = \text{ANY(}\text{SELECT } n+1 \text{ FROM } \textsc{Odd})), \\
\text{RECURSIVE} \textsc{Odd}(n) \text{ AS } \\
(\text{SELECT } n \text{ FROM } \textsc{Natural} \text{WHERE } n = 1) \\
\text{UNION} \\
(\text{SELECT } n \text{ FROM } \textsc{Natural} \text{WHERE } n = \text{ANY(}\text{SELECT } n+1 \text{ FROM } \textsc{Even}));

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

Fixed points are not unique

\textsc{WITH RECURSIVE} \\
\textsc{Ancestor}(\text{anc, desc}) \text{ AS } \\
(\text{SELECT } \text{parent, \text{child} FROM } \textsc{Parent}) \\
\text{UNION} \\
(\text{SELECT } a1.\text{anc, a2.desc} \\
\text{FROM } \textsc{Ancestor} a1, \text{Ancestor} a2 \\
\text{WHERE } a1.\text{desc} = a2.\text{anc})

- 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 \textbf{minimal} fixed point is the “natural” answer

\textbf{Note how 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—we wouldn’t know which one to pick as answer!
- Example: popular users (pop \( \geq 0.8 \)) 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 pop >= 0.8
   AND uid NOT IN (SELECT uid FROM JessicaCircle)),
RECURSIVE JessicaCircle(uid) AS
  (SELECT uid FROM User WHERE pop >= 0.8
   AND uid NOT IN (SELECT uid FROM TommyCircle))

Fixed-point iter may not converge

WITH RECURSIVE TommyCircle(uid) AS
  (SELECT uid FROM User WHERE pop >= 0.8
   AND uid NOT IN (SELECT uid FROM JessicaCircle)),
RECURSIVE JessicaCircle(uid) AS
  (SELECT uid FROM User WHERE pop >= 0.8
   AND uid NOT IN (SELECT uid FROM TommyCircle))

Multiple minimal fixed points

WITH RECURSIVE TommyCircle(uid) AS
  (SELECT uid FROM User WHERE pop >= 0.8
   AND uid NOT IN (SELECT uid FROM JessicaCircle)),
RECURSIVE JessicaCircle(uid) AS
  (SELECT uid FROM User WHERE pop >= 0.8
   AND uid NOT IN (SELECT uid FROM TommyCircle))
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 “−”

Stratified negation example

• Find pairs of persons with no common 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)),
    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
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 ∅
• 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)