Announcements (October 3)

- Homework #2 graded
  - Solution was emailed during weekend
- Midterm in class this Thursday
  - Open book, open notes
  - Format similar to the sample midterm
    - Solution was emailed during weekend
  - Optional Gradiance problem set for practice is available
    - Covers everything up to today’s lecture
    - Emphasizes materials exercised in homeworks
- Project milestone #1 due next Thursday

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)

Fixed point of a function

- If $f: T \to T$ 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

A motivating example

- “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

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 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.

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 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.

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))

Linear recursion

- With linear recursion, a recursive definition can make only one reference to itself.
- Non-linear:

Linear:

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

Mutual recursion example

- Table \( Natural \) 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.

\( \text{Even}(n) \) AS
(SELECT n FROM Natural
WHERE n = ANY(SELECT n+1 FROM Odd)).

\( \text{Odd}(n) \) 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. $R_i \leftarrow \emptyset$, ..., $R_n \leftarrow \emptyset$
  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 any $i$
     3.1. $R_1 \leftarrow R_1^{new}$, ..., $R_n \leftarrow R_n^{new}$
     3.2. Go to 2.
  4. Compute $Q$ using the current contents of $R_1$, ..., $R_n$ and output the result

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 = $\emptyset$, 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\}
- ...

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 $g$ 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

Note that the bogus tuple reinforces itself!

Mixing negation with recursion

- If $g$ 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 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

Student

<table>
<thead>
<tr>
<th>SID</th>
<th>name</th>
<th>age</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>857</td>
<td>Lisa</td>
<td>8</td>
<td>4.3</td>
</tr>
<tr>
<td>999</td>
<td>Jessica</td>
<td>10</td>
<td>4.2</td>
</tr>
</tbody>
</table>

Scholarship | DeansList | Scholarship | DeansList

- Scholarship = \{857\} | DeansList = \{\}
- Scholarship = \{999\} | DeansList = \{\}
- Scholarship = \{857, 999\} | DeansList = \{\}
- Scholarship = \{\} | DeansList = \{857, 999\}

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))

- Student

<table>
<thead>
<tr>
<th>SID</th>
<th>name</th>
<th>age</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>857</td>
<td>Lisa</td>
<td>8</td>
<td>4.3</td>
</tr>
<tr>
<td>999</td>
<td>Jessica</td>
<td>10</td>
<td>4.2</td>
</tr>
</tbody>
</table>

Scholarship | DeansList | Scholarship | DeansList

- Scholarship = \{857\} | DeansList = \{\}
- Scholarship = \{999\} | DeansList = \{\}
- Scholarship = \{857, 999\} | DeansList = \{\}
- Scholarship = \{\} | DeansList = \{857, 999\}
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 "−"

Legal SQL3 recursion:

- `Legal.sql`
- Illegal!

Stratified negation example

- Find pairs of persons with no common ancestors

```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)),
   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`

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)