INFO2820 – Database Systems I (Adv)

Week 5: Recursive SQL and DATALOG

(Kifer/Bernstein/Lewis – Chapter 13.6; Ramakrishnan/Gehrke – Chapter 24; Ullman/Widom – Chapter 10.2)

Dr. Uwe Röhm
School of Information Technologies

Outline

- Recursive Queries in SQL:1999
- Hierarchical Queries in Oracle

Based on material from Kifer/Bernstein/Lewis (2006) “Database Systems”

Limitations of SQL

- We have seen the previous weeks that standard SQL-92 cannot expressive recursion
- So traversals of hierarchies or graphs is not possible in plain SQL
- Example:
  Flights (flightcode, frm, to, departs, arrives)
  - On this database we can ask:
    Are cities X and Y connected by a direct flight? When is it leaving?
  - But we cannot answer (in a general way):
    Between which pairs of cities (X,Y) can we travel by taking one or more flights?

How do we do this in Datalog?

- **EDB:**
  flights (ua450, syd, lax, 0630, 1845)
  flights (qf006, syd, sin, 0645, 1420)
  flights (ua451, lax, nyc, 2155, 0607)
  …
- **IDB:**
  reaches (X, Y) :- flights (_, X, Y, _, _).
  reaches (X, Z) :- reaches (X,Y), flights(_,Y,Z,_,_).

Note: Arrival and departure times are not considered here – how would you do this?

How translate this into SQL?

- Given a relation ***Flights*** with attributes *code* and *frm* and *to*, list the set of all cities that are directly or indirectly connected by flight connections
  - The direct connections are represented by the ***Flights*** relation itself

$$\text{Reaches}_1 = \pi_{frm,\, to}\,(\text{Flights})$$

  - The set Reaches_2, computed by the following expression, contains the immediate and once removed connections for all cities:

$$\text{Reaches}_2 = \pi_{frm,\, to}\,(\text{Flights} \bowtie_{to=frm} \text{Reaches}_1) \cup \text{Reaches}_1$$

  - In general, $Reaches_i$ contains all flight-connections up to those that are *i*-1 removed for all pairs of cities:

$$\text{Reaches}_i = \pi_{frm,\, to}\,(\text{Flights} \bowtie_{to=frm} \text{Reaches}_{i-i}) \cup \text{Reaches}_{i-1}$$

Limitations of SQL (con't)

- **Question**: We can compute $\sigma_{from='SYD'}(\text{Reaches}_i)$ to get all cities reachable from Sydney up to those that are *i*-1 flights away, but how can we be sure that there are not additional connections that are *i* removed?
- **Answer**: When you reach a value of *i* such that $\text{Reaches}_i = \text{Reaches}_{i+1}$ you've got them all. This is referred to as a *stable state*
- **Problem**: There's no way of doing this within relational algebra, or SQL (this is *not* obvious and *not* easy to prove)

Recursion in SQL:1999

- Since SQL:1999, recursion was added to SQL
  - But only gradually available in implementations; PostgreSQL supports the following only since two years (v 8.4)
- Recursive queries can be formulated using a recursive view:

```
(a) CREATE RECURSIVE VIEW Reaches (frm, to) AS
    SELECT frm, to FROM Flights
(b) UNION
    SELECT R.frm, F.to
    FROM Reaches R, Flights F
    WHERE R.to = F.frm
```

- (a) is a *non*-recursive *seed query* – it cannot refer to the view being defined
  - Starts recursion off by introducing the *base case* – the set of direct prerequisites

Recursion in SQL:1999 (cont'd)

```
CREATE RECURSIVE VIEW Reaches (frm, to) AS
SELECT frm, to FROM Flights
UNION
(b) SELECT R.frm, F.to
    FROM Reaches R, Flights F
    WHERE R.to = F.frm
```

- (b) contains *recursion* – this subquery refers to the view being defined.
  - This is a declarative way of specifying the iterative process of calculating successive levels of indirect prerequisites until a stable point is reached

Recursion using *Common Table Expressions* (CTE)

- SQL:1999 introduced *common table expressions* for queries via the WITH construct to not require creating views
- Can also be used to define recursive queries:

```
WITH RECURSIVE  Reaches(frm, to)  AS
            (SELECT  frm, to  FROM  Flights)
        UNION
            (SELECT   R.frm,  F.to
             FROM     Reaches R, Flights F
             WHERE    R.to=F.frm )
SELECT  *
  FROM  Reaches
 WHERE frm = 'SYD'
```


Example: How to compute #Hops?

```
WITH RECURSIVE Connection (frm, to, hops)  AS
   (
        (SELECT  F.frm, F.to, 1
           FROM   Flights F )
        UNION
        (SELECT  C.frm, F.to, C.hops+1
           FROM  Connection C JOIN Flights F ON (C.to= F.frm) )
   )
 SELECT *
  FROM Connection
 WHERE …
```

Restrictions on the Use of Aggregation

- Assume, you want to determine the shortest flight connection between two given cities

```
CREATE RECURSIVE VIEW ShortestConnection (frm, to, hops)  AS
(       (SELECT  F.frm, F.to, 1
           FROM   Flights F )
        UNION
        (SELECT  S.frm, F.to, MIN(S.hops)+1
           FROM  ShortestConnection S, Flights F
        WHERE   S.to= F.frm
        GROUP BY F.frm, S.to) )
```

Doesn't work… For MIN(…) in the second SELECT clause, need whole relation
SQL:1999 does not support this kind of recursive aggregation

So how to do it?

Restrictions on the Use of Aggregation

- Assume, you want to determine the longest flight connection between two given cities.
- Solution:

```
WITH RECURSIVE Connection (frm, to, hops)  AS
(
        (SELECT  F.frm, F.to, 1
           FROM   Flights F )
    UNION
        (SELECT  C.frm, F.to, C.hops+1
           FROM  Connection C JOIN Flights F ON (C.to= F.frm) )
)
SELECT frm, to, MIN(hops)
  FROM Connection
GROUP BY frm, to
```

Another Recursive SQL Example

- Example: find all employee-manager pairs, where the employee reports to the manager directly or indirectly (that is manager's manager, manager's manager's manager, etc.)

```
with recursive empl (employee_name, manager_name ) as(
        select employee_name, manager_name
          from manager
union
        select manager.employee_name, empl.manager_name
          from manager, empl
         where manager.manager_name= empl.employee_name)
select* from empl
```

- This example view, *empl,* is called the *transitive closure* of the *manager* relation

DBMS Comparison

DBMS	Recursive SQL	Syntax
PostgreSQL	Yes, since v8.4 *(both recursive views and recursive CTE)*	`CREATE RECURSIVE VIEW …` or `WITH RECURSIVE … SELECT …`
MySQL	No	
SQLite	No	
IBM DB2	Yes (since ca. v7.2) *(both recursive views and recursive CTE)*	`CREATE RECURSIVE VIEW …` or `WITH RECURSIVE … SELECT …`
SQL Server	since SQL Server 2005 *(recursive CTE)*	`WITH RECURSIVE … SELECT …`
Sybase	since SQL Anywhere v9 *(recursive CTE)*	`WITH RECURSIVE … SELECT …`
Oracle	proprietary solution	`SELECT … CONNECT BY [PRIOR] …`

Recursive Queries in Oracle

- Oracle is traditionally following a different approach, supporting a form of 'hierarchical queries' already quite some time via its CONNECT BY clause
  - It does not support the SQL:1999 recursive views or a WITH clause

- Hierarchical Query Clause:

  SELECT …
  FROM …
  WHERE …
  START WITH **root condition**
  CONNECT BY PRIOR *childattr* = *parentattr*

- Example: (next page)

CONNECT-BY Example

SELECT *
FROM Flights
START WITH *frm* = 'SYD'
CONNECT BY PRIOR *to=frm*

Further Oracle Specialties

- There are some additional utility operators and functions for hierarchical queries in Oracle
- Pseudo-Columns(SELECT clause)
  - LEVEL — iteration counter
  - CONNECT_BY_ISLEAF — returns 1 if leaf, else 0
  - CONNECT_BY_ISCYCLE — returns 1 if current row has a child which is also its ancestor, else 0
- SELECT Functions
  - CONNECT_BY_ROOT *col* — column *col* shall show tree root, not immediate parent
  - SYS_CONNECT_BY_PATH(*col*, *delimiter)*
- Ordering
  - ORDER SIBLINGS BY — determine order of sibling rows
- cf. Oracle SQL reference manual, sections 3 and 9-3

Oracle Example (from Manual)

```
SELECT last_name "Employee",
          CONNECT_BY_ROOT last_name "Manager",
          LEVEL-1 "Pathlen",
          SYS_CONNECT_BY_PATH(last_name, '/') "Path"
   FROM Employees
WHERE LEVEL > 1  AND department_id = 110
CONNECT BY PRIOR employee_id = manager_id;
```

Employee	Manager	Pathlen	Path
Higgins	Kochhar	1	/Kochhar/Higgins
Gietz	Kochhar	2	/Kochhar/Higgins/Gietz
Gietz	Higgins	1	/Higgins/Gietz
Higgins	King	2	/King/Kochhar/Higgins
Gietz	King	3	/King/Kochhar/Higgins/Gietz

You should now be able to:

- Understand the limitations of SQL wrt. recursive queries
- Write a recursive SQL query

References

- Kifer/Bernstein/Lewis (2nd edition – 2005)
  - Chapter 13.6
- Ramakrishnan/Gehrke (3rd edition - the 'Cow' book – 2003)
  - Chapter 24
- Ullman/Widom (3rd edition – 2008)
  - Chapter 10.2
    One nice section about the recursive SQL clause in SQL:1999.
    No oracle specialities
- Oracle SQL Reference Manual