CS245: Databases

SQL

Vijaya saradhi

Department of Computer Science and Engineering
Indian Institute of Technology Guwahati

Recursion

Use case

- A common kind of operation on data is that operators cannot be applied infinitely and recursively which are defined using sequence of similar expressions

Use case

M1	
movie	sequel
Rocky	Rocky II
Rocky II	Rocky III
Rocky III	Rocky IV

Use case - Sequel of Sequel

Select M1C1.movie, M1C2. sequel FROM M1 as M1C1 JOIN M1 as M1C2 ON M1C1.sequel = M1C2.movie

M1C1		M1C2		M1C1.sequel = M1C2.movie
movie	sequel	movie	sequal	Not in M1
Rocky	Rocky II	Rocky	Rocky II	
Rocky	Rocky II	Rocky II	Rocky III	(Rocky, Rocky II)
Rocky	Rocky II	Rocky III	Rocky IV	
Rocky II	Rocky III	Rocky	Rocky II	
Rocky II	Rocky III	Rocky II	Rocky III	
Rocky II	Rocky III	Rocky III	Rocky IV	(Rocky II, Rocky IV)
Rocky III	Rocky IV	Rocky	Rocky II	
Rocky III	Rocky IV	Rocky II	Rocky III	
Rocky III	Rocky IV	Rocky III	Rocky IV	

Recursion

Perform Join

obtain records which are not in $M1$

M1C1-M1C2

movie	sequel
Rocky	Rocky III
Rocky II	Rocky IV

Perform Union

Add the new records to $M1$ to make it $M2$

M2

movie	sequel
Rocky	Rocky II
Rocky II	Rocky III
Rocky III	Rocky IV
Rocky	Rocky III
Rocky II	Rocky IV

Use case - Sequel of Sequel of Sequel

Perform (M21 ← M2 Join M1) followed by (M2 Union M21)

M2	
movie	sequel
Rocky	Rocky II
Rocky II	Rocky III
Rocky III	Rocky IV
Rocky	Rocky III
Rocky II	Rocky IV

M1	
movie	sequel
Rocky	Rocky II
Rocky II	Rocky III
Rocky III	Rocky IV

Use case - Sequel of Sequel of Sequel

M1-M2				
movie	sequel	movie	sequal	Not in M2
Rocky	Rocky II	Rocky	Rocky II	
Rocky	Rocky II	Rocky II	Rocky III	(Rocky, Rocky III)
Rocky	Rocky II	Rocky III	Rocky IV	
Rocky II	Rocky III	Rocky	Rocky II	
Rocky II	Rocky III	Rocky II	Rocky III	
Rocky II	Rocky III	Rocky III	Rocky IV	(Rocky II, Rocky IV)
Rocky III	Rocky IV	Rocky	Rocky II	
Rocky III	Rocky IV	Rocky II	Rocky III	
Rocky III	Rocky IV	Rocky III	Rocky IV	(Rocky, Rocky IV)
Rocky	Rocky III	Rocky	Rocky II	
Rocky	Rocky III	Rocky II	Rocky III	
Rocky	Rocky III	Rocky III	Rocky IV	
Rocky II	Rocky IV	Rocky	Rocky II	
Rocky II	Rocky IV	Rocky II	Rocky III	
Rocky II	Rocky IV	Rocky III	Rocky IV	

Use case - Sequel of Sequel of Sequel

Perform (M21 ← M2 Join M1) followed by (M2 Union M21)

M2	
movie	sequel
Rocky	Rocky II
Rocky II	Rocky III
Rocky III	Rocky IV
Rocky	Rocky III
Rocky II	Rocky IV

M1	
movie	sequel
Rocky	Rocky II
Rocky II	Rocky III
Rocky III	Rocky IV

Use case - Sequel of Sequel of Sequel

M1-M2				
movie	sequel	movie	sequal	Not in M2
Rocky	Rocky II	Rocky	Rocky II	
Rocky	Rocky II	Rocky II	Rocky III	(Rocky, Rocky III)
Rocky	Rocky II	Rocky III	Rocky IV	
Rocky	Rocky II	Rocky	Rocky III	
Rocky	Rocky II	Rocky II	Rocky IV	(Rocky, Rocky IV)
Rocky II	Rocky III	Rocky	Rocky II	
Rocky II	Rocky III	Rocky II	Rocky III	
Rocky II	Rocky III	Rocky III	Rocky IV	(Rocky II, Rocky IV)
Rocky II	Rocky III	Rocky	Rocky III	
Rocky II	Rocky III	Rocky II	Rocky IV	
Rocky III	Rocky IV	Rocky	Rocky II	
Rocky III	Rocky IV	Rocky II	Rocky III	
Rocky III	Rocky IV	Rocky III	Rocky IV	(Rocky III, Rocky IV
Rocky III	Rocky IV	Rocky	Rocky IV	
Rocky III	Rocky IV	Rocky II	Rocky IV	

Recursion: Sequel of Sequel of Sequel

Perform Join

obtain records which are not in $M2$

M1-M2

movie	sequel
Rocky	Rocky IV

Perform Union

Add the new records to $M2$ to make it $M3$

M3

movie	sequel
Rocky	Rocky II
Rocky II	Rocky III
Rocky III	Rocky III
Rocky	Rocky III
Rocky II	Rocky IV
Rocky	Rocky IV

Recursion: Sequel of Sequel of Sequel of Sequel?

Trilogy and more?

- Repeating above steps: Perform M3 Join M1
- Perform M3 Union (records (M3 Join M1) not in M3) to obtain M4
- These two steps yields no new records. The recursion terminates

Datalog

Formulation

- FollowOn(x, y) $\leftarrow$ M1(x, y)
- FollowOn(x, y) $\leftarrow$ M1(x, z) AND FollowOn(z, y)

Airlines Database

Graph & Table

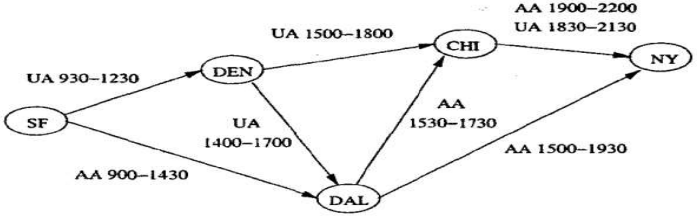

Figure 10.5: A map of some airline flights

airline	*from*	*to*	*departs*	*arrives*
UA	SF	DEN	930	1230
AA	SF	DAL	900	1430
UA	DEN	CHI	1500	1800
UA	DEN	DAL	1400	1700
AA	DAL	CHI	1530	1730
AA	DAL	NY	1500	1930
AA	CHI	NY	1900	2200
UA	CHI	NY	1830	2130

Datalog: Recursive Programming

Recursive Rules

1. `Reaches(x, y) ← Flights(a, x, y, d, r)`
2. `Reaches(x, y) ← Reaches(x, z) AND Reaches(z, y)`

Flights				
airline	from	to	departs	arrives
UA	SF	DEN	0930	1230
AA	SF	DAL	0900	1430
UA	DEN	CHI	1500	1800
UA	DEN	DAL	1400	1700
AA	DAL	CHI	1530	1730
AA	DAL	NY	1500	1930
AA	CHI	NY	1900	2200
UA	CHI	NY	1830	2130

Round # 1

Reaches	
x	y
SF	DEN
SF	DAL
DEN	CHI
DEN	DAL
DAL	CHI
DAL	NY
CHI	NY

Datalog: Recursive Programming

Recursive Result

1. `Reaches(x, y) ← Reaches(x, z) AND Reaches(z, y)`

Reaches R1		Reaches R2	
R1.x	R1.z	R2.z	R2.y
CHI	NY	SF	DEN
CHI	NY	SF	DAL
CHI	NY	DEN	CHI
CHI	NY	DEN	DAL
CHI	NY	DAL	CHI
CHI	NY	DAL	NY
CHI	NY	CHI	NY

Round # 1	
Reaches	
x	y
SF	DEN
SF	DAL
DEN	CHI
DEN	DAL
DAL	CHI
DAL	NY
CHI	NY
Round #2	
SF	CHI
SF	NY
DEN	NY

Recursive Result

1. `Reaches(x, y) ← Reaches(x, z) AND Reaches(z, y)`

Reaches R1		Reaches R2	
R1.x	R1.z	R2.z	R2.y
DEN	NY	SF	DEN
DEN	NY	SF	DAL
DEN	NY	DEN	CHI
DEN	NY	DEN	DAL
DEN	NY	DAL	CHI
DEN	NY	DAL	NY
DEN	NY	CHI	NY
DEN	NY	DAL	CHI
DEN	NY	DAL	NY
DEN	NY	CHI	NY

Round # 2

Reaches	
x	y
SF	DEN
SF	DAL
DEN	CHI
DEN	DAL
DAL	CHI
DAL	NY
CHI	NY
SF	CHI
SF	NY
DEN	NY

Datalog: Recursive Programming

Recursive Rules

1. Difference between Departure time at next hop and arrival time should be at least 100 minutes
2. `Connects(x, y, d, r)` ← `Flights(a, x, y, d, r)`
3. `Connects(x, y, d, r)` ← `Connects(a, x, z, d, t1)` AND `Connects(a, z, y, t2, r)` AND `t1` $\leq$ `t2 - 100`

Flights				
airline	from	to	departs	arrives
UA	SF	DEN	0930	1230
AA	SF	DAL	0900	1430
UA	DEN	CHI	1500	1800
UA	DEN	DAL	1400	1700
AA	DAL	CHI	1530	1730
AA	DAL	NY	1500	1930
AA	CHI	NY	1900	2200
UA	CHI	NY	1830	2130

Connects			
x	y	d	r
Round #1			
SF	DEN	0930	1230
SF	DAL	0900	1430
DEN	CHI	1500	1800
DEN	DAL	1400	1700
DAL	CHI	1530	1730
DAL	NY	1500	1930
CHI	NY	1900	2200
Round #2			
SF	CHI	0900	1730
SF	CHI	0930	1800
SF	DAL	0930	1700
DEN	NY	1500	2200
DAL	NY	1530	2130
DAL	NY	1530	2200
Round #3			
SF	NY	0900	2130
SF	NY	0900	2200
SF	NY	0930	2200

Datalog: Recursive Programming

Recursive Rules

1. Find those pairs of cities (x, y) in which only UA operates but not AA
2. `UAReaches(x, y) ← Flights(a, x, y, d, r)`
3. `UAReaches(x, y) ← UAReaches(x, z) AND UAReaches(z, y)`
4. `AAReaches(x, y) ← Flights(a, x, y, d, r)`
5. `AAReaches(x, y) ← AAReaches(x, z) AND AAReaches(z, y)`
6. `UAOnly(x, y) ← UAReaches(x, y) AND NOT AAReaches(x, y)`

Flights				
airline	from	to	departs	arrives
UA	SF	DEN	0930	1230
AA	SF	DAL	0900	1430
UA	DEN	CHI	1500	1800
UA	DEN	DAL	1400	1700
AA	DAL	CHI	1530	1730
AA	DAL	NY	1500	1930
AA	CHI	NY	1900	2200
UA	CHI	NY	1830	2130

UAReaches	
x	y
SF	DEN
SF	DAL
SF	CHI
SF	NY
DEN	DAL
DEN	CHI
DEN	NY
CHI	NY

AAReaches	
SF	DAL
SF	CHI
SF	NY
DAL	CHI
DAL	NY
CHI	NY

UAOnly	
x	y
SF	DEN
DEN	DAL
DEN	CHI
DEN	NY

SQL: Recursive Programming

Introduction

- SQL has grown to be an expressive data-oriented language
- Intentionally it has not been designed as a general-purpose programming language
- SQL does not loop forever.
- Any SQL query is expected to terminate, regardless of size/contents of the input tables
- SQL queries are evaluated efficiently

SQL: Addition of recursion

Expressive

SQL becomes a Turing-complete language thus a general-purpose programming lanugage

Efficiency

No longer queries are guaranteed to terminate.

SQL: WITH RECURSIVE

Recursive common table expression (CTE)

```
WITH RECURSIVE T(c1, c2, ..., ck)  -- common schema of q0 and q.(.)
AS(
   q0      -- base case query, evaluated once

   UNION [ALL]

   q0(T) -- recursive query refers to T itself
)        -- evaluated repeatedly

q(T)      -- final post processing query
```

SQL: WITH RECURSIVE

Initial query

Forms the base result set of the CTE structure. The initial query part is referred to as an anchor member.

Recursive query

References to the CTE name, therefore, it is called a recursive member. The recursive member is joined with the anchor member by UNION or UNION ALL

Termination

A termination condition that ensures the recursion stops when the recursive member returns no row

SQL: WITH RECURSIVE

Step 1

Separate the members into anchor and recursive members

Step 2

Execute the anchor member to form the base result set R0. Use this base result set for next iteration

Step 3

Execute the recursive member with Ri result set as input and make Ri+1 as an output

Step 4

Repeat step 3 till recursive member returns an empty result set

Step 5

Combine result sets from R0 to Rn using UNION [ALL] operator

Example - 01

Recursive Rules

1. `Reaches(x, y) ← Flights(a, x, y, d, r)`
2. `Reaches(x, y) ← Reaches(x, z) AND Reaches(z, y)`

Recursive SQL Query

```
WITH RECURSIVE Reaches(frm, to)  -- T(c1, c2)
AS(
    (SELECT frm, to FROM Flights) -- q0

      UNION

    ( SELECT R1.frm, R2.to
      FROM   Reaches AS R1, Reaches AS R2
      WHERE  R1.to = R2.frm
    )
)
    SELECT * FROM Reaches;
```