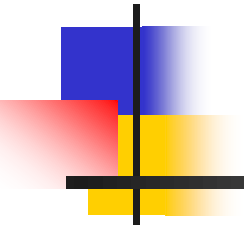
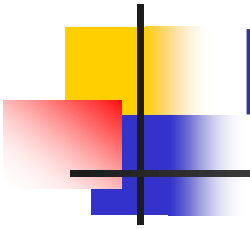
Chapter 10 Advanced topics in relational databases



- Security and user authorization in SQL
- Recursion in SQL
- Object-relational model
1. User-defined types in SQL
2. Operations on object-relational data
- Online analytic processing & data cubes



Examples

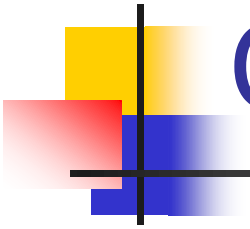
- EDB: Par(c,p) = p is a parent of c.

Query1: Who is the parent of Sally?

Select p from Par where c='Sally';

Query2: find Sally's brothers or sisters ?

```
Select p2.c
From Par P1,Par P2
Where P1.c='Sally' and P1.p=P2.p and P2.c
<>'Sally';
```



Question?

- Query3: We want to find generalized cousins: people with common ancestors one or more generations back.
- Find Sally's generalized cousins?
- Find Sally's ancestors?
- How ?

Solutions: Recursive

Sib(x,y) <- Par(x,p) AND Par(y,p) AND x<>y

Cousin(x,y) <- Sib(x,y)

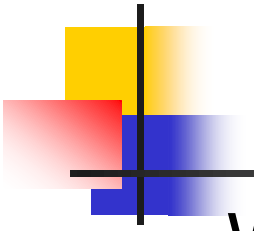
Cousin(x,y) <- Par(x,xp) AND Par(y,yp)
AND Cousin(xp,yp)

- EDB: Par(x,p)
- IDB: Sib(x,y) , Cousin(x,y)

Base query

recursive query

How to Evaluate?



- We'll proceed in rounds to **infer** Sib facts (red) and Cousin facts (green).
- Remember the rules:

Sib(x,y) <- Par(x,p) AND Par(y,p) AND x<>y

Cousin(x,y) <- Sib(x,y)

Cousin(x,y) <- Par(x,xp) AND Par(y,yp)
AND Cousin(xp,yp)

Value used : Round (i) ← round (i-1)

At the beginning, sib(x,y) and cousin(x,y) are empty.

Par Data: Parent Above Child

Sib(x,y) <- Par(x,p) AND Par(y,p) AND x<>y

Cousin(x,y) <- Sib(x,y)

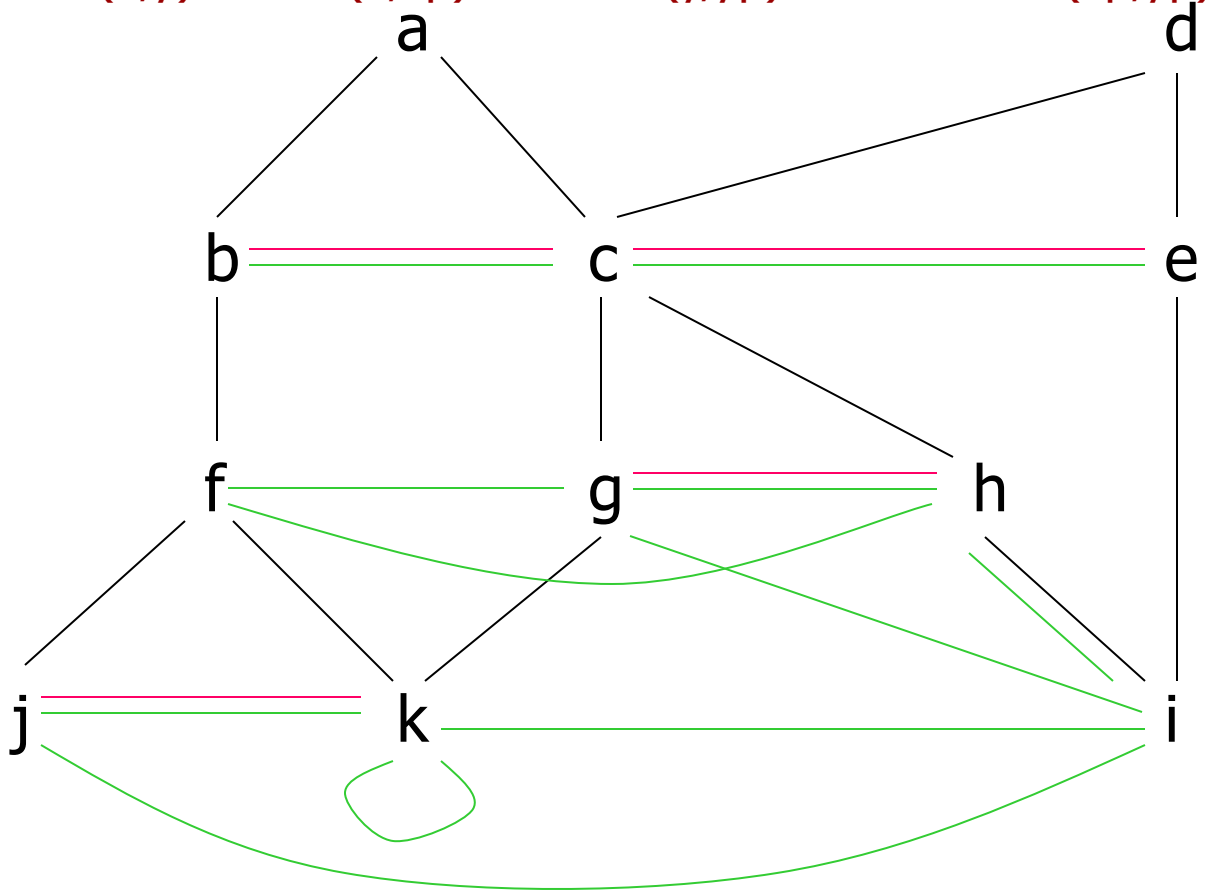
Cousin(x,y) <- Par(x,xp) AND Par(y,yp) AND Cousin(xp,yp)

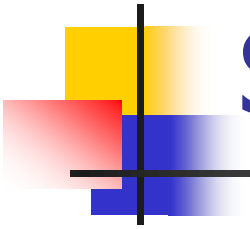
Round 1

Round 2

Round 3

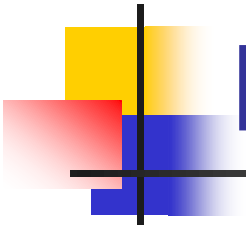
Round 4





SQL-99 Recursion

- Datalog recursion has inspired the addition of recursion to the SQL-99 standard.
- IBM DB2 does implement the SQL-99 proposal.



Form of SQL Recursive Queries

WITH

[RECURSIVE] R1 AS <Definition of R1>

[RECURSIVE] R2 AS <Definition of R2>

<a SQL query about EDB,R1,R2,…>

R1,R2: temporary relations, they are not available outside the query

Example: SQL Recursion – (1)

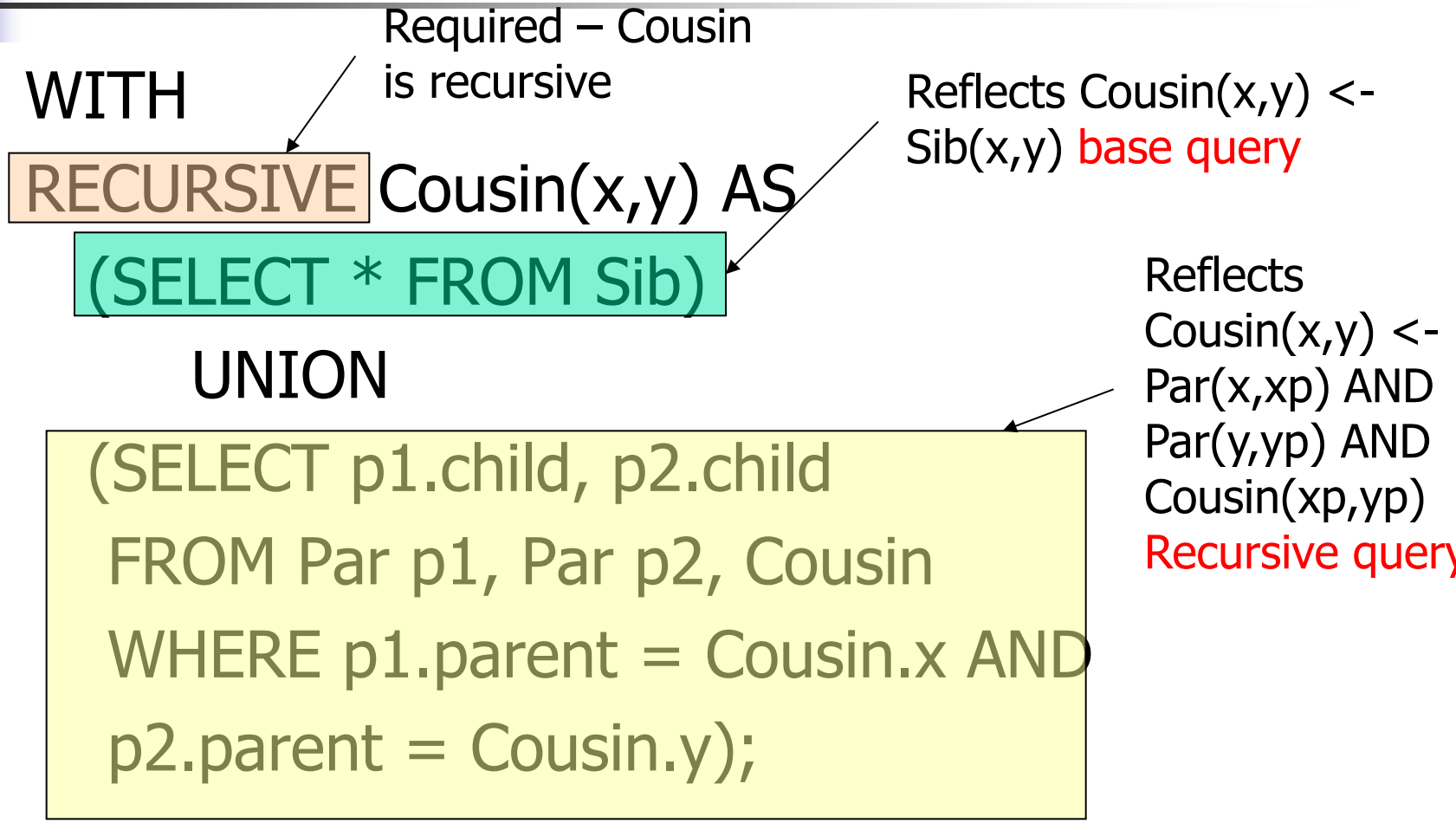
- Find Sally's cousins, using SQL like the recursive Datalog example.
- Par(child,parent) is the EDB.

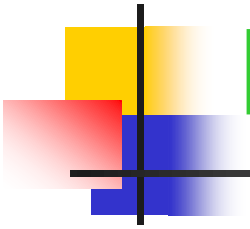
WITH Sib(x,y) AS

```
SELECT p1.child, p2.child
FROM Par p1, Par p2
WHERE p1.parent = p2.parent AND
      p1.child <> p2.child;
```

Like Sib(x,y) <-
Par(x,p) AND
Par(y,p) AND
x <> y

Example: SQL Recursion – (2)

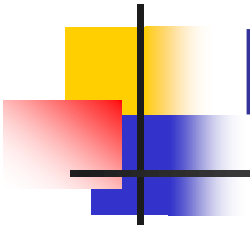




Example: SQL Recursion – (3)

- With those definitions, the query to Cousin(x,y):

```
SELECT y
FROM Cousin
WHERE x = 'Sally';
```



Legal SQL Recursion

- It is possible to define SQL recursions that do not have a meaning.
- The SQL standard restricts recursion so there is a meaning.

Non-linear Recursive

Include more R1 instead of once

WITH

[RECURSIVE] R1 AS <Definition of R1>

[RECURSIVE] R2 AS <Definition of R2>

<a SQL query about EDB,R1,R2,…>

Non-linear Recursive (example)

- ParentOf(parent,child)
- With recursive

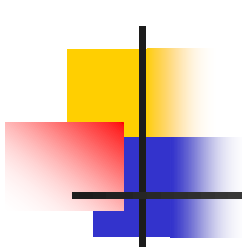
Ancestor(a,d) as (select parent as a,child as d from ParentOf

Union

Select A1.a,A2.d from Ancestor a1,Ancestor a2 where a1.d=a2.a)

Select a from Ancestor where d='Sally' ;

Mutual Recursive



Include R2

WITH

[RECURSIVE] R1 AS <Definition of R1>

[RECURSIVE] R2 AS <Definition of R2>

<a SQL query about EDB,R1,R2,…>

Include R1

Recursive with *aggregation*

With Recursive P(x) As
(select * from R) union
(select * from **Q**),
With Recursive Q(x) As
select sum(x) from **P**

Select * from P;

R is an EDB, consists of tuple 12 and 34

P(x),Q(x) are empty.

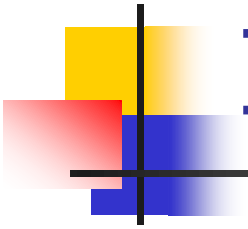
Recursive with aggregation (cont.)

Round	P	Q
1	{(12),(34) }	{null}
2	{(12),(34),(null) }	{(46)}
3	{(12),(34),(46)}	{(46)}
4	{(12),(34),(46)}	{(92)}
5	{(12),(34),(92)}	{(138)}
…		

Problem: Iterative calculation for aggregation, no meaningful solution.

Legal SQL recursion

- **Linear recursion**
- **Mutual recursion with monotone**.
  - ✓ A use of P is monotone if adding an arbitrary tuple to P might add one or more tuples to Q, or it might leave Q unchanged, but it can never cause any tuple to be deleted from Q.



Illegal SQL: non-monotone

With Recursive P(x) As
(select * from R) union
(select * from **Q**),
With Recursive Q(x) As
select sum(x) from **P**

- Add tuples to P may delete tuples in Q

Classroom Exercises (1) write a linear recursive SQL to find the ancestor of Sally

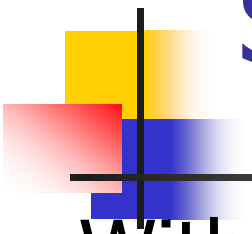
- ParentOf(parent,child)
- With recursive

Ancestor(a,d) as (select parent as a,child as d from ParentOf

Union

Select A1.a,A2.d from Ancestor a1,Ancestor a2 where a1.d=a2.a)

Select a from Ancestor where d='Sally' ;



Solution:

With recursive

Ancestor(a,d) as (select parent as a, child as d from Parentof

Union

Select ancestor.a, parentOf.child as d

From Ancestor, parentOf

Where ancestor.d=parentOf.parent)

Select a from Ancestor where d='Sally";

Classroom Exercise （2）

```
create table Employee(ID int, salary int);
create table Manager(mID int, eID int);
create table Project(name text, mgrID
  int);
```

- Find total salary cost of project 'X'

Solution 1:

Employee(ID , salary)
Manager(mID, eID)
Project(name,mgrID)

```
with recursive
  Superior as (select * from Manager
                union
                select S.mID, M.eID
                from Superior S, Manager M
                where S.eID = M.mID )

select sum(salary)
from Employee
where ID in
  (select mgrID from Project where name = 'X'
   union
   select eID from Project, Superior
   where Project.name = 'X' AND Project.mgrID = Superior.mID );
```

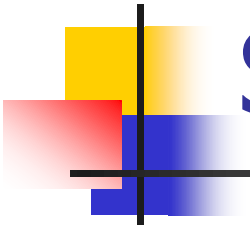
Define one or more recursive rules

Make a query

Solution 2:

Employee(ID , salary)
Manager(mID, eID)
Project(name,mgrID)

```
with recursive
  Xemps(ID) as (select mgrID as ID from
   Project where name = 'X'
                union
                select eID as ID
                from Manager M, Xemps X
                where M.mID = X.ID)
select sum(salary)
from Employee
where ID in (select ID from Xemps);
```



Summary

- SQL recursive query → for some application, it is very useful and powerful.