Datalog

Regole logiche Ricorsione
Ricorsione in SQL-99

Leggere capitolo 11 di Riguzzi et al. Sistemi Informativi

Lucidi derivati da quelli messi di Jeffrey D. Ullman

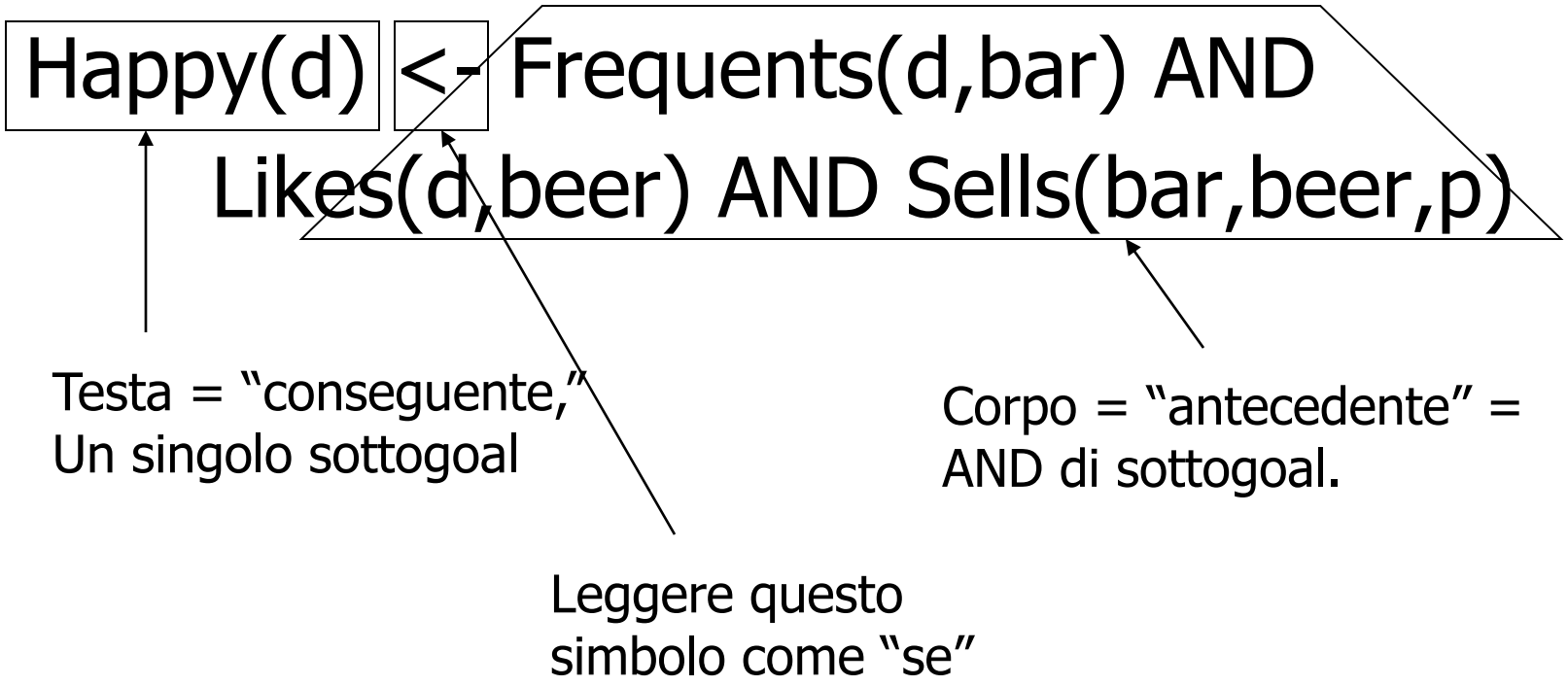
La logica come linguaggio di query

- Regole del tipo If-then sono state usate in molti sistemi.
- Regole non ricorsive sono equivalenti all'algebra relazionale di base.
- Le regole ricorsive estendono l'algebra relazionale --- sono usate per aggiungere la ricorsione a SQL-99.

Una regola logica

- Il nostro primo esempio di una regola usa le relazioni Frequents(drinker,bar), Likes(drinker,beer), e Sells(bar,beer,price).
- La regola e' una query che chiede i bevitori "felici" --- quelli che frequentano un bar che serve una birra che gli piace.

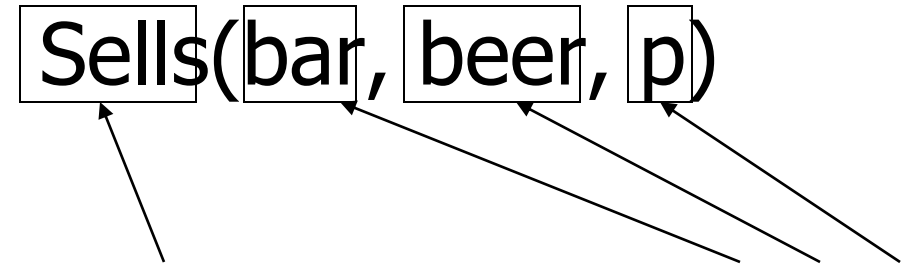
Anatomia di una regola



I sottogoal sono atomi

- Un *atomo* e' un *predicato*, o un nome di relazione, con variabili o costanti come argomenti.
- La testa e' un atomo; il corpo e' un AND di uno o piu' atomi.
- Convenzione: predicati cominciano con una maiuscola, variabili con una minuscola.

Esempio: Atomo



Il predicato
= nome di una relazione

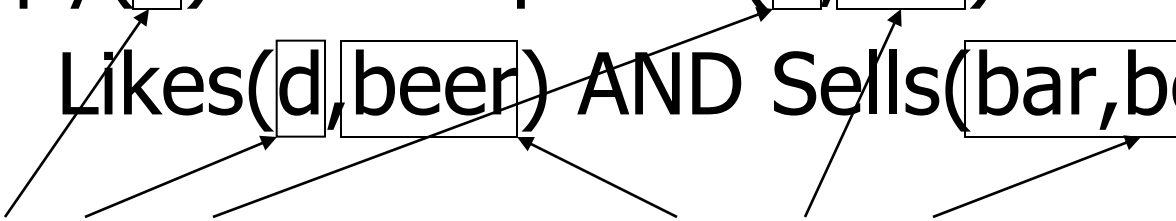
Gli argomenti sono variabili

Interpretazione delle regole

- Una variabile che appare nella testa si chiama *distinguished* ; altrimenti si chiama *nondistinguished*.
- Significato della regola: la testa e' vera per un assegnamento delle variabili distinguished se esistono valori per le variabili nondistinguished che rendono veri tutti i sottogoal del body.

Esempio: interpretazione

Happy(d) <- Frequents(d,bar) AND
Likes(d,beer) AND Sells(bar,beer,p)



variabile distinguished

variabili nondistinguished

Interpretazione: il drinker d e' felice se esiste un bar, una birra, e un prezzo p tali che d frequenta il bar, apprezza la birra, e il bar vende la birra al prezzo p

Sottogoal aritmetici

- Oltre alle relazioni come predicati, un predicato per un sottogoal del body puo' essere un confronto aritmetico.
  - Si scrivono tali sottogoal nella maniera solita, ad es.: $x < y$.

Esempio: sottogoal aritmetici

- Una birra e' economica (cheap) se esistono almento due bar che la vendono a meno di $2.

Cheap(beer) <- Sells(bar1,beer,p1) AND
Sells(bar2,beer,p2) AND p1 < 2.00
AND p2 < 2.00 AND bar1 <> bar2

Sottogoal negati

- Possiamo mettere NOT davanti a un sottogoal per negare il suo significato.
- Esempio: si consideri Arc(a,b) come archi in un grafo.
  - S(x,y) dice che il grafico non e' transitivo da x a y, cioe' c'e' un cammino di lunghezza 2 da x a y ma non c'e' un arco da x a y.

S(x,y) <- Arc(x,z) AND Arc(z,y)
AND NOT Arc(x,y)

Regole safe

- Una regola e' *safe* se:
  1. Ogni variabile distinguished,
  2. Ogni variabile in un sottogoal aritmetico,
  3. Ogni variabile in un sottogoal negato,

  appare anche in un sottogoal non negato e relazionale.
- Si ammettono solo regole safe.

Esempio: regole non safe

- Ciascuna delle seguenti e' non safe e quindi non ammessa:
  1. S(x) <- R(y)
  2. S(x) <- R(y) AND NOT R(x)
  3. S(x) <- R(y) AND x < y
- In ogni caso, un'infinita' di x puo' soddisfare la regola, anche se R e' una relazione finita.

Algoritmi per applicare le regole

- Due approcci:
  1. *Basato sulle variabili*: si considerino tutti i possibili assegnamenti delle variabili nel body. Se un assegnamento rende il corpo vero, si aggiunga la tupla per la testa al risultato.
  2. *Basato sulle tuple* : si considerino tutti gli assegnamenti delle tuple per i sottogoal non negati relazionali. Se il corpo diventa vero si aggiunga la tupla della testa al risultato.

Esempio: basato sulle variabili--- 1

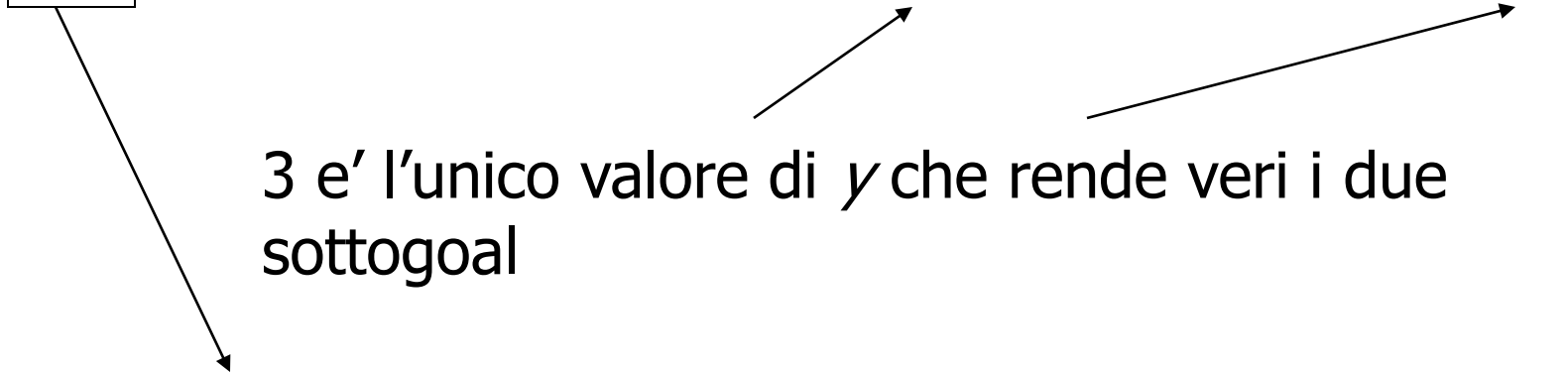
S(x,y) <- Arc(x,z) AND Arc(z,y)
AND NOT Arc(x,y)

- Arc(1,2) e Arc(2,3) sono le uniche tuple nella relazione Arc.
- I soli assegnamenti che rendono vero il primo sottogoal Arc(x,z) sono:
  1. x = 1; z = 2
  2. x = 2; z = 3

Esempio: basato sulle variabili; x=1, z=2

S(x,y) <- Arc(x,z) AND Arc(z,y) AND NOT Arc(x,y)

1 3 1 2 2 3 1 3



3 e' l'unico valore di *y* che rende veri i due sottogoal

rende (1,3) una tupla della risposta

Esempio: basato sulle variabili; x=2, z=3

S(x,y) <- Arc(x,z) AND Arc(z,y) AND NOT Arc(x,y)

2 2 3 3 2

Quindi, nessun contributo alle tuple della testa;
S = {(1,3)}

Nessun valore di y rende Arc(3,y) vero

Algoritmo basato sulle tuple

- Cominciamo con i sottogoal relazionali non negati.
- Consideriamo tutti gli assegnamenti di tuple a questi sottogoal.
  - Scegliamo le tuple solo dalle relazioni corrispondenti.
- Se le tuple assegnate hanno un valore consistente per tutte le variabili e rendono gli altri sottogoal veri, allora aggiungiamo la tuple per la testa al risultato.

Esempio: basato sulle tuple

S(x,y) <- Arc(x,z) AND Arc(z,y) AND NOT Arc(x,y)

Arc(1,2), Arc(2,3)

- 4 possibili assegnamenti ai primi due sottogoal:

Arc(x,z)	Arc(z,y)
(1,2)	(1,2)
(1,2)	(2,3)
(2,3)	(1,2)
(2,3)	(2,3)

Unico assegnamento con valore consistente per z. Dato che rende vero anche NOT Arc(x,y) si aggiunge (1,3) al risultato.

Programmi Datalog

- Un programma Datalog e' una collezione di regole
- In un programma, i predicati possono essere
  1. EDB = *Extensional Database* = tabella memorizzata.
  2. IDB = *Intensional Database* = relazione definita da una regola.
- Mai insieme! Nessun EDB nella testa.

Valutare programmi Datalog

- Se non c'e' ricorsione, possiamo selezionare un ordine per valutare i predicati IDB, affinche' tutti i predicati nel corpo delle sue regole siano gia' stati valutati
- Se un predicato IDB ha piu' di una regola, ogni regola contribuisce tuple alla sua relazione.

Esempio: programma Datalog

- Usando EDB Sells(bar, beer, price) e Beers(name, manf), trovare il produttore delle birre che Joe non vende.

JoeSells(b) <- Sells('Joe"s Bar', b, p)

Answer(m) <- Beers(b,m)

AND NOT JoeSells(b)

Dall'algebra relazionale al Datalog

- Siano R(A,B) e S(A,B) due relazioni
- Intersezione I=R $\cap$ S
  - I(a,b)<-R(a,b) AND S(a,b)
- Unione U=R $\cup$ S
  - U(a,b)<-R(a,b)
  - U(a,b)<-S(a,b)

Dall'algebra relazionale al Datalog

- Differenza D=R-S
  - D(a,b)<-R(a,b) AND NOT S(a,b)
- Proiezione P=π_A R
  - P(a)<-R(a,b) oppure
  - P(a)<-R(a,_) (_ variabile anonima)

Dall'algebra relazionale al Datalog

- Selezione
  - $S=\sigma_{A\geq 1 \text{ AND } B=2}R$:
  - S(a,b)<-R(a,b) AND a≥1 AND b=2
  - $S=\sigma_{A\geq 1 \text{ OR } B=2}R$:
  - S(a,b)<-R(a,b) AND a≥1
S(a,b)<-R(a,b) AND b=2

Dall'algebra relazionale al Datalog

- Selezioni piu' complesse possono essere portate nella "forma normale disgiuntiva" che e' una disgiunzione di congiunzioni
- Ogni congiunto si trasforma in una regola Datalog

Dall'algebra relazionale al Datalog

- Prodotto PR = R $\times$ S
  - PR(a,b,c,d)<-R(a,b) AND S(c,d)
- Join naturale: R(A,B) S(B,C,D), J=R$\triangleright\triangleleft$S
  - J(a,b,c,d)<-R(a,b) AND S(b,c,d)

Dall'algebra relazionale al Datalog

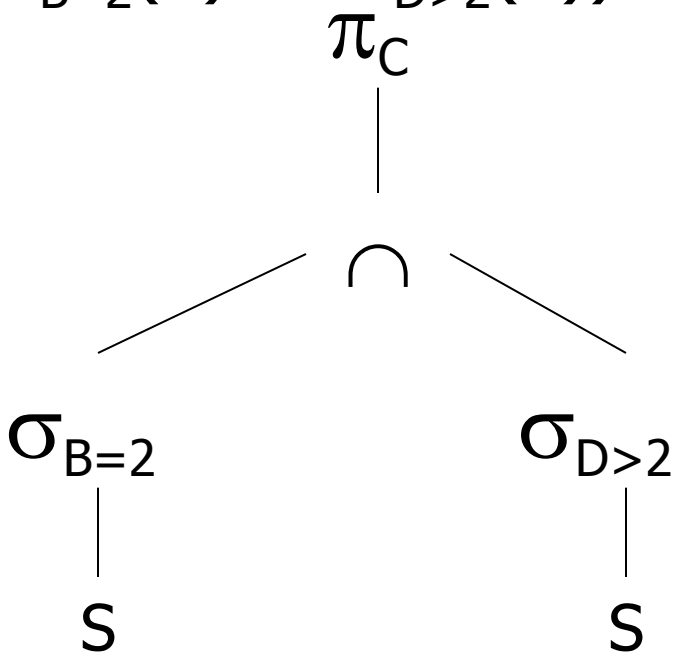
- Theta join: U(A,B,C) V(B,C,D),
  - $U \triangleright\!\triangleleft_{A<D \text{ AND } U.B \neq V.B} V$:
  - J(a,ub,uc,vb,vc,d)<-U(a,ub,uc) AND V(vb,vc,d) AND a<d AND ub $\neq$ vb
  - $U \triangleright\!\triangleleft_{A<D \text{ OR } U.B \neq V.B} V$:
  - J(a,ub,uc,vb,vc,d)<-U(a,ub,uc) AND V(vb,vc,d) AND a<d
  - J(a,ub,uc,vb,vc,d)<-U(a,ub,uc) AND V(vb,vc,d) AND ub $\neq$ vb

Conversione di espressioni relazionali in Datalog

- Si guarda all'albero che rappresenta l'espressione relazionale
- Si crea un predicato IDB per ogni nodo interno dell'albero
- Le regole per gli IDB sono quelle che servono per applicare l'operatore
- Gli operandi sono rappresentati dai corrispondenti predicati EDB o IDB

Esempio

- S(B,C,D)
- $P=\pi_C(\sigma_{B=2}(S) \cap \sigma_{D>2}(S))$



Esempio

- W(b,c,d)<-S(b,c,d) AND B=2
- X(b,c,d)<-S(b,c,d) AND D>2
- Y(b,c,d)<-W(b,c,d) AND X(b,c,d)
- Z(c)<-Y(_,c,_)

Potere espressivo del Datalog

- Senza ricorsione, il Datalog puo' esprimere tutte e sole le query che si possono esprimere con l'algebra relazionale di base.
  - Lo stesso dell' SQL select-from-where, senza aggregazioni e raggruppamenti.
- Ma con la ricorsione, Datalog puo' esprimere piu' di questi linguaggi
- Ma non e' Turing-completo.

Esempio ricorsivo

- EDB: SequelOf(x,y): il film y e' un sequel del film x
- IDB: FollowOn(x,y): il film y e' successivo al film x

FollowOn(x,y)<-SequelOf(x,y)

FollowOn(x,y)<-SequelOf(x,z) AND FollowOn(z,y)

Esempio ricorsivo

- EDB: Par(c,p) = *p* e' un genitore di *c*.
- Cugino generalizzato: due persone con un comune progenitore una o piu' generazioni indietro:

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

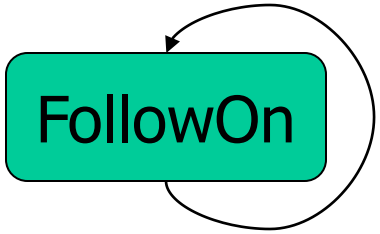
Cousin(x,y) <- Sib(x,y)

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

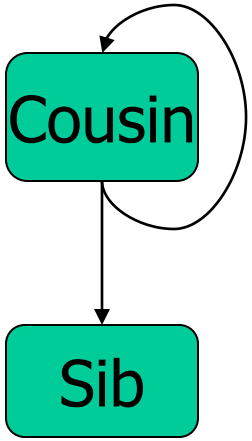
Definizione di ricorsione

- Si crei un grafo di dipendenza i cui nodi sono i predicati IDB
- C'e' un arco *X* -> *Y* se e solo se c'e' una regola con *X* nella testa e *Y* nel corpo.
- Ciclo = ricorsione; nessun ciclo = nessuna ricorsione.

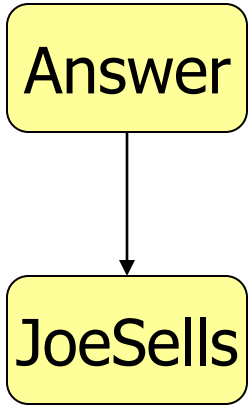
Esempio: grafi di dipendenza



Ricorsivo



Ricorsivo

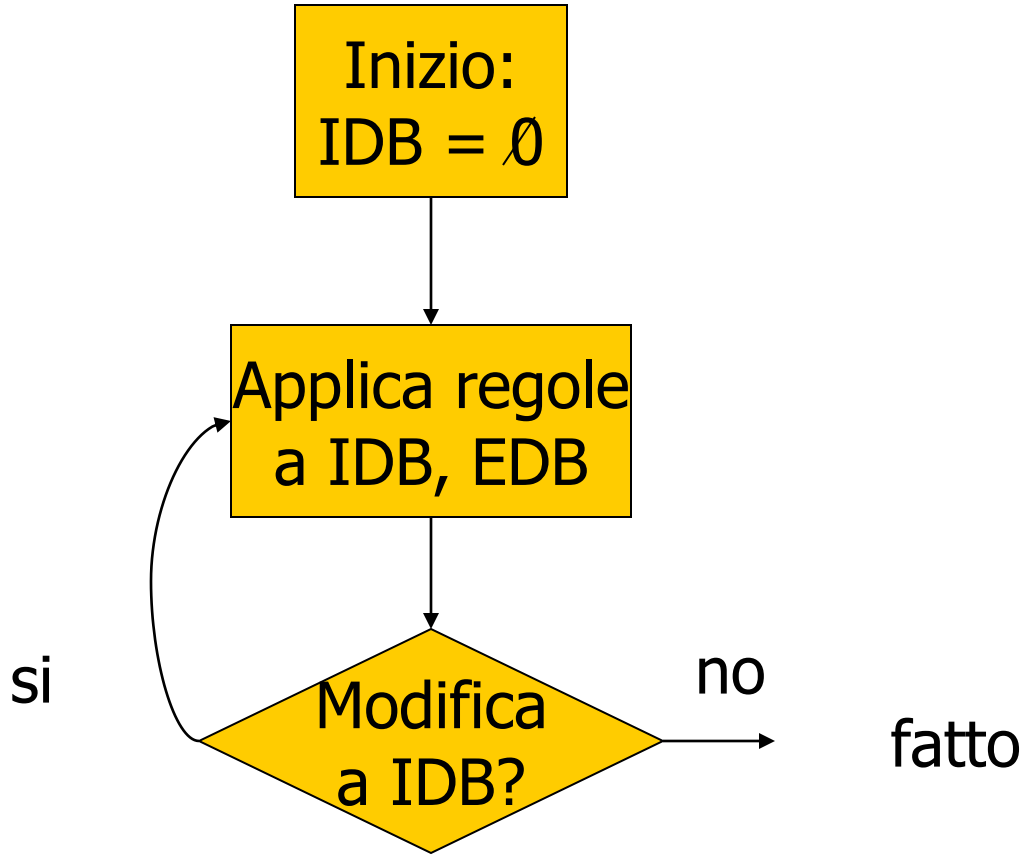


Non ricorsivo

Valutare regole ricorsive

- Quando non c'e' negazione:
  1. Si comincia assumendo che tutte le relazioni IDB siano vuote.
  2. Si valutano le regole ripetutamente usando gli EDB e i precedenti IDB per ottenere un nuovo IDB.
  3. Fine quando non ci sono piu' cambiamenti ad un IDB.
- Si trova un "punto fisso"

Algoritmo di valutazione ingenuo



Valutazione semi-ingenua

- Dato che l'EDB non cambia mai, ad ogni turno otteniamo nuove tuple IDB solo se usiamo almeno una tupla IDB ottenuto al turno precedente.
- Risparmiamo lavoro; evitiamo di riscoprire la maggior parte dei fatti noti
  - Un fatto potrebbe comunque essere riscoperto in un secondo modo

Esempio: valutazione di FollowOn

SequelOf

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

Dopo il I round

x	y
Rocky	Rocky II
Rocky II	Rocky III
Rocky III	Rocky IV

Esempio: valutazione di FollowOn

Dopo il II round

x	y
Rocky	Rocky II
Rocky II	Rocky III
Rocky III	Rocky IV
Rocky	Rocky III
Rocky II	Rocky IV

Dopo il III round

x	y
Rocky	Rocky II
Rocky II	Rocky III
Rocky III	Rocky IV
Rocky	Rocky III
Rocky II	Rocky IV
Rocky	Rocky IV

Esempio: valutazione di cousin

- Procediamo in turni per inferire fatti per Sib e per Cousin
- Le regole sono:

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)

Dati per Par: genitore sopra il figlio

Sib

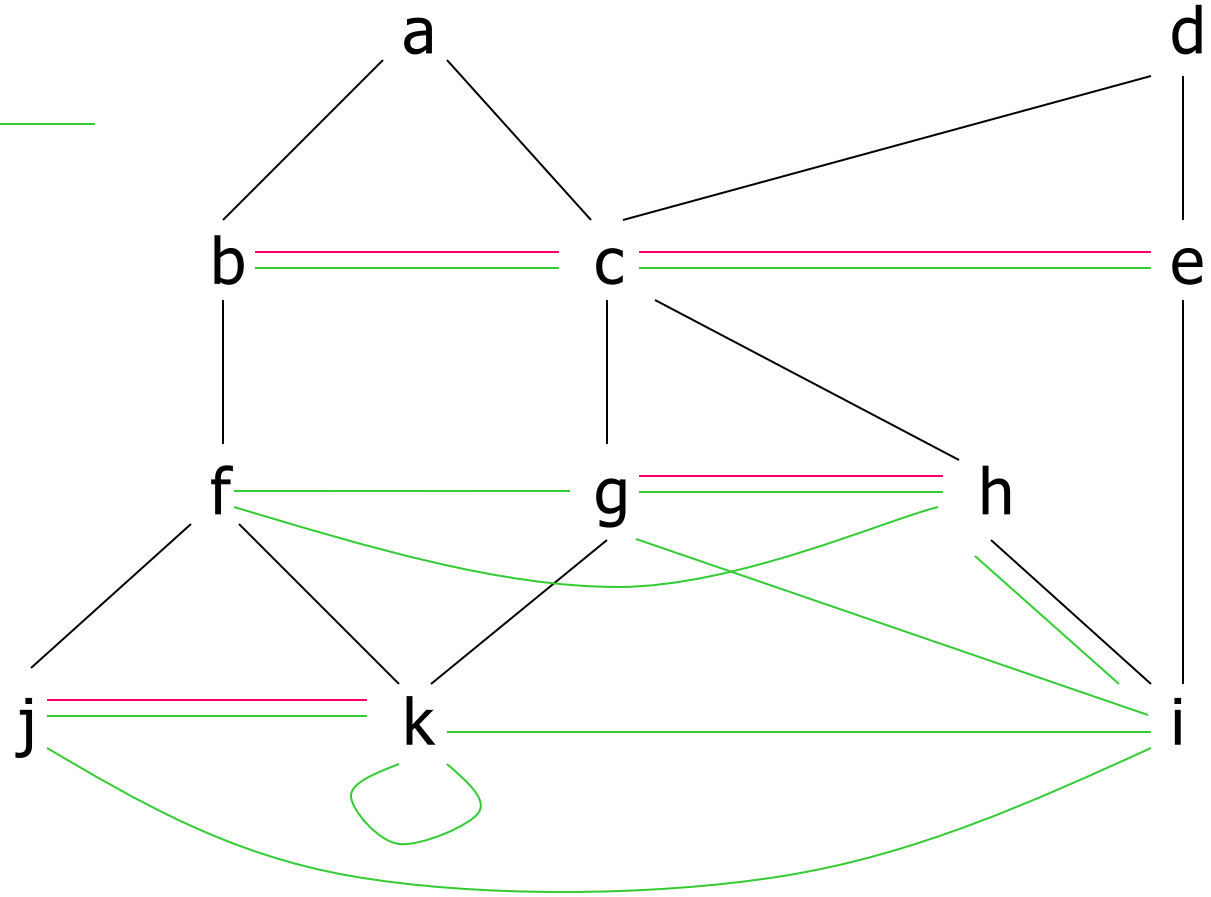
Cousin

Turno 1

Turno 2

Turno 3

Turno 4



Altro esempio: Path

- IDB:
  - Path(x,y) <- Edge(x,y)
  - Path(x,y) <- Path(x,z) AND Edge(z,y)
- EDB:
  - Edge(A, B)
  - Edge(B, A)

Altro esempio

- Turno 1: {Path(A,B),Path(B,A)}
- Turno 2: {Path(A,B),Path(B,A),Path(A,A), Path(B,B)}
- Turno 3: {Path(A,B),Path(B,A),Path(A,A), Path(B,B)}
- Stop

Rispondere a query

- Possiamo stabilire se una query è vera o falsa usando la valutazione semi-ingenua:
  - Calcoliamo tutti gli atomi veri
  - Verifichiamo se la query è tra questi
- Inefficiente: rischiamo di generare molti atomi irrilevanti per la query
- Prolog è efficiente in questo caso perché usa una computazione orientata alla query

Combinazione di valutazione semi-ingenua e orientata alla query

- Trasformazione “magica”
- Trasformare un programma P e una query <-Q in magic(P U {<-Q})
- Inserimento di un filtro (condizione extra) nel body di ciascuna regola A0 <-A1 AND…AND An in modo che A0 sia una conseguenza del programma solo se è necessaria per rispondere alla query

Combinazione di valutazione semi-ingenua e orientata alla query

- magic({A0<-A1 AND…AND An})=
  A0<-Call(A0) AND A1 AND … AND An
  Call(Ai) <- Call(A0) AND A1 AND… Ai-1
  1<=i<=n
- A0<-Call(A0) AND A1 AND … AND An: A0 è vero se A0 è necessario per rispondere alla query e A1 AND … AND An è vero
- Call(Ai) <- Call(A0) AND A1 AND… Ai-1:
  Ai è chiamato se A0 è chiamato e A1 AND… Ai-1 è vero

magic(Path)

- Path(x,y)<-Call_Path(x,y) AND Edge(x,y)
- Call_Edge(x,y)<- Call_Path(x,y)
- Path(x,y)<-Call_Path(x,y) AND Path(x,z) AND Edge(z,y)
- Call_Path(x,z)<-Call_Path(x,y)
- Call_Edge(z,y)<-Call_Path(x,y) AND Path(x,z)
- Edge(A,B)<-Call_Edge(A,B)
- Edge(B,A) <-Call_Edge(B,A)

magic({<-Q})

- Call(Q)
- Q=Path(A,B), magic({<-Path(A,B)}= Call_Path(A,B)

Esempio di valutazione semi-ingenua

- Turno 0, Δ={Call_Path(A,B)}
- Turno 1, Δ={Call_Edge(A,B)}
- Turno 2, Δ={Edge(A,B)}
- Turno 3, Δ={Path(A,B)}
- Turno 4, Δ={Call_Edge(B,B)}
- Turno 5, Δ={}

Ricorsione piu’ negazione

- La valutazione “ingenua“ non funziona quando ci sono sottogoal negati.
- Infatti, in presenza di negazione, si puo’ non raggiungere un punto fisso

Ricorsione negativa problematica

P(x) <- Q(x) AND NOT P(x)
EDB: Q(1), Q(2)

Inizio: P = { }
Turno 1: P = {(1), (2)}
Turno 2: P = { }
Turno 3: P = {(1), (2)}, etc., etc. …
Non si raggiunge un punto fisso

Ricorsione negativa problematica

- EBD R(0)

P(x) <- R(x) AND NOT Q(x)

Q(x) <- R(x) AND NOT P(x)

Inizio: P={}, Q={}

Turno 1: P={(0)} Q={(0)}

Turno 2: P={} Q={}

Turno 1: P={(0)} Q={(0)}

Non si raggiunge un punto fisso

Ricorsione negativa problematica

- Pero' ci sono due punti fissi, ovvero due situazioni in cui applicando le regole si ottengono le stesse relazioni di partenza:
- P={} Q={(0)}
- P={(0)} Q={}
- Entrambi sono minimi, quale scegliamo?
- Nessuno, non possiamo decidere

Negazione stratificata

- La stratificazione e' un vincolo normalmente posto al Datalog con ricorsione e negazione.
- Elimina la negazione coinvolta nella ricorsione.

Strati

- Intuitivamente, lo *strato* di un predicato IDB *P* e' il numero massimo di negazioni che possono essere applicate ad un predicato IDB usato nella valutazione di *P*.
- Negazione stratificata= "strati finiti".
- Si noti che in P(x) <- Q(x) AND NOT P(x), possiamo negare *P* un numero infinito di volte nel derivare P(x).

Grafo degli strati

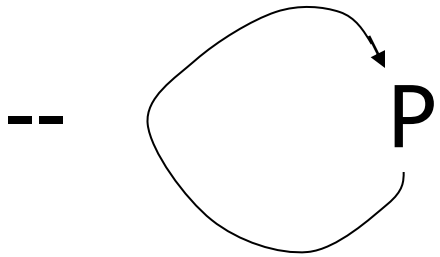
- Per formalizzare gli strati usiamo il grafo degli strati :
  - Nodi = predicati IDB.
  - Arco *A* ->*B* se il predicato *A* dipende da *B*.
  - Etichetta questo arco "–" se il sottogoal *B* e' negato.

Definizione di negazione stratificata

- Lo *strato* di un nodo (predicato) e' il massimo numero di archi – su un percorso che parte da quel nodo.
- Un programma Datalog e' *stratificato* se tutti i suoi predicati IDB hanno strato finito, cioe' se il grafo non ha un ciclo con uno o piu' archi -

Esempio

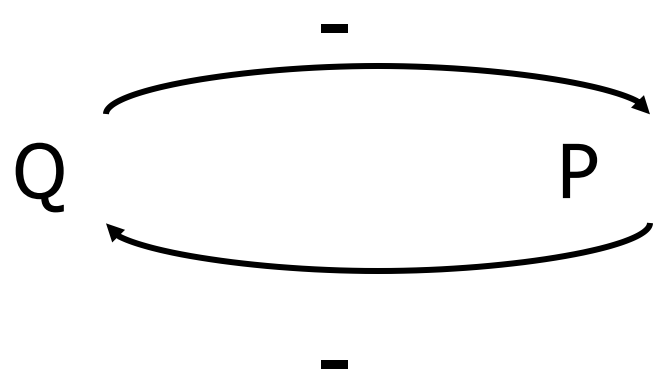
P(x) <- Q(x) AND NOT P(x)



Esempio

P(x) <- R(x) AND NOT Q(x)

Q(x) <- R(x) AND NOT P(x)



Valutazione dei predicati IDB

- Se la negazione e' stratificata, possiamo valutare i predicati IDB nell'ordine degli strati, partendo dallo strato piu' piccolo
- Una volta valutati, sono trattati come predicati EDB per gli strati piu' alti
- Si ottiene cosi' un punto fisso

Valutazione dei predicati IDB

- Anche un programma stratificato puo' avere piu' di un punto fisso ma c'e' un solo punto fisso che ha senso, e' quello che si ottiene nel modo precedente

Un altro esempio

- EDB = Source(x), Target(x), Arc(x,y).
- Regole per "targets non raggiunto da nessuna sorgente":

Reach(x) <- Source(x)

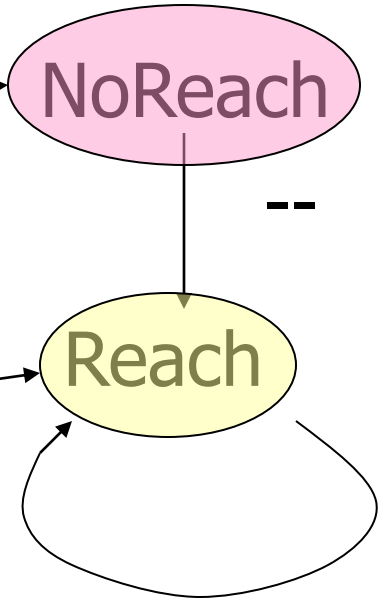
Reach(x) <- Reach(y) AND Arc(y,x)

NoReach(x) <- Target(x)
AND NOT Reach(x)

Il grafo degli strati

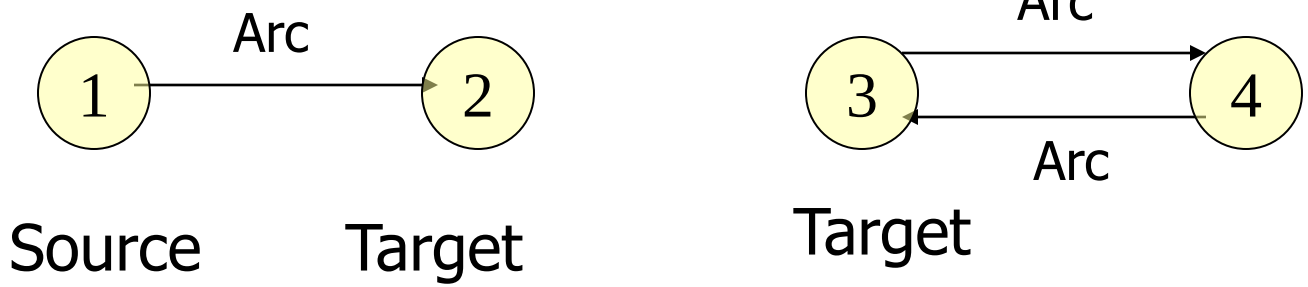
Strato 1:
= 1 arco - su ogni cammino da qui.

Strato 0:
nessun arco – su nessun cammino da qui.



Esempio

- Source(1) Target(2) Target(3)
- Arc(1,2) Arc(3,4) Arc(4,3)



Esempio

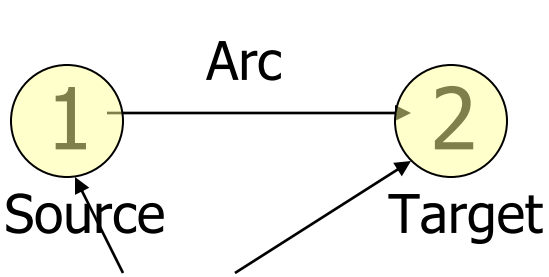
- Si comincia dallo strato 0: Reach
- Reach(1) Reach(2)
- Poi si passa allo strato 1: NoReach
- NoReach(3)

Esempio: punti fissi multipli-1

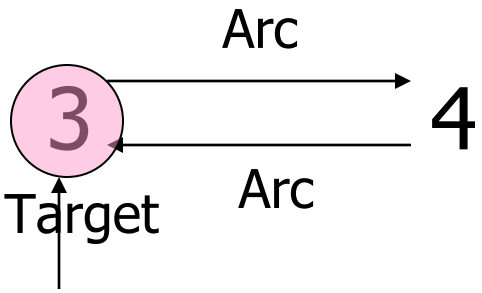
Reach(x) <- Source(x)

Reach(x) <- Reach(y) AND Arc(y,x)

NoReach(x) <- Target(x) AND NOT Reach(x)



Strato 0:
Reach(1), Reach(2)



Strato 1:
NoReach(3)

Esempio: punti fissi multipli-2

Reach(x) <- Source(x)

Reach(x) <- Reach(y) AND Arc(y,x)

NoReach(x) <- Target(x) AND NOT Reach(x)

Un altro punto fisso: Reach(1), Reach(2), Reach(3), Reach(4); NoReach e' vuoto.

Ricorsione in SQL-99

- La ricorsione Datalog ha ispirato l'aggiunta della ricorsione allo standard SQL-99.
- Difficoltoso, perche' SQL consente raggruppamenti e aggregazioni, che si comportano come la negazione e richiedono una nozione di stratificazione piu' complessa.

Forma di query SQL ricorsive

WITH

<regole>

<una query SQL riguardo EDB, IDB>

<regole> =

[RECURSIVE] <nome>(<argomenti>)

AS <query> [,…n]

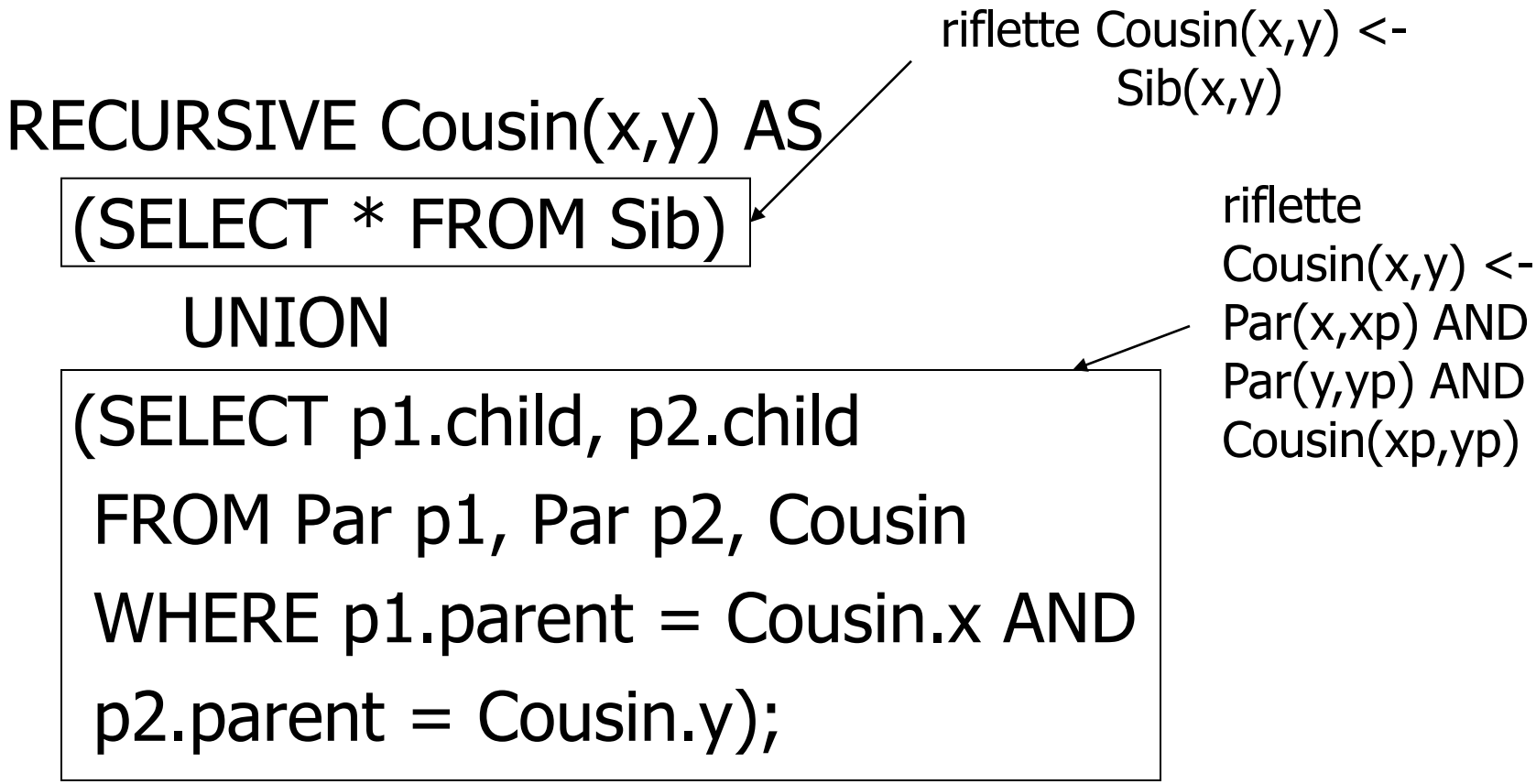
Esempio: ricorsione in SQL --- 1

- Trova i cugini di Sally, usando SQL come nell'esempio Datalog ricorsivo.
- Par(child,parent) e' l'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,
```

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

Esempio: ricorsione in SQL--2



Esempio: ricorsione in SQL---3

- Con queste definizioni, possiamo aggiungere la query, che riguarda la "vista temporanea" Cousin(x,y):

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

Ricorsione legale in SQL

- La negazione deve essere stratificata, esempio di negazione non stratificta

```
WITH
   RECURSIVE P(x) AS
   (SELECT * FROM R) EXCEPT (SELECT *
   FROM Q),
   RECURSIVE Q(x) AS
   (SELECT * FROM R) EXCEPT (SELECT *
   FROM P)
SELECT * FROM P;
```

Ricorsione legale in SQL

- EXCEPT introduce problemi in SQL
- Li introduce anche NOT IN
  - RECURSIVE P(x) AS SELECT x FROM R WHERE x NOT IN Q
- Li introducono anche le query di raggruppamento e aggregazione

Piano per spiegare la ricorsione legale in SQL

1. Si definisca "monotono"=una generalizzazione di stratificato
2. Si generalizzi il grafo degli strati per essere applicato all'SQL
3. Si definisca la ricorsione legale in SQL in termini del grafo degli strati

Monotonicita'

- Se la relazione P e' una funzione di una relazione Q (e forse di altre relazioni), diciamo che P e' *monotono* in Q se inserire tuple in Q non puo' causare la cancellazione di tuple da P
- Esempi:
  - P = Q UNION R.
  - P = SELECT$_{a=10}(Q)$.

Esempio: nonmonotonicita’

- Se Sells(bar,beer,price) e’ la nostra relazione usuale, allora il risultato della query:

```
SELECT AVG(price)
FROM Sells
WHERE bar = 'Joe''s Bar';
```

e’ non monotono in Sells.

- Inserire una tuple per Joe’s Bar puo’ cambiare il prezzo medio e quindi cancellare il vecchio prezzo medio

Grafo degli strati in SQL - 1

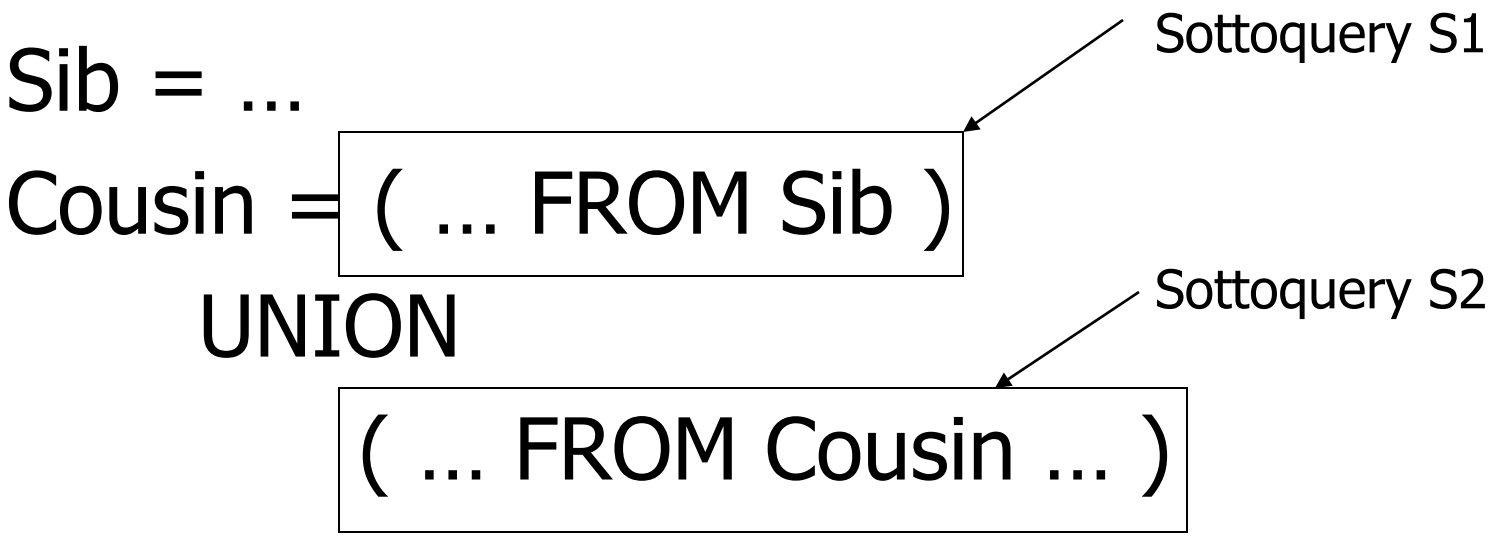
- Nodi =
  1. Relazioni IDB dichiarate in una clausola WITH.
  2. Sottoquery nel corpo delle "query"
     - Include sottoquery a ogni livello di innestamento

Grafo degli strati in SQL - 2

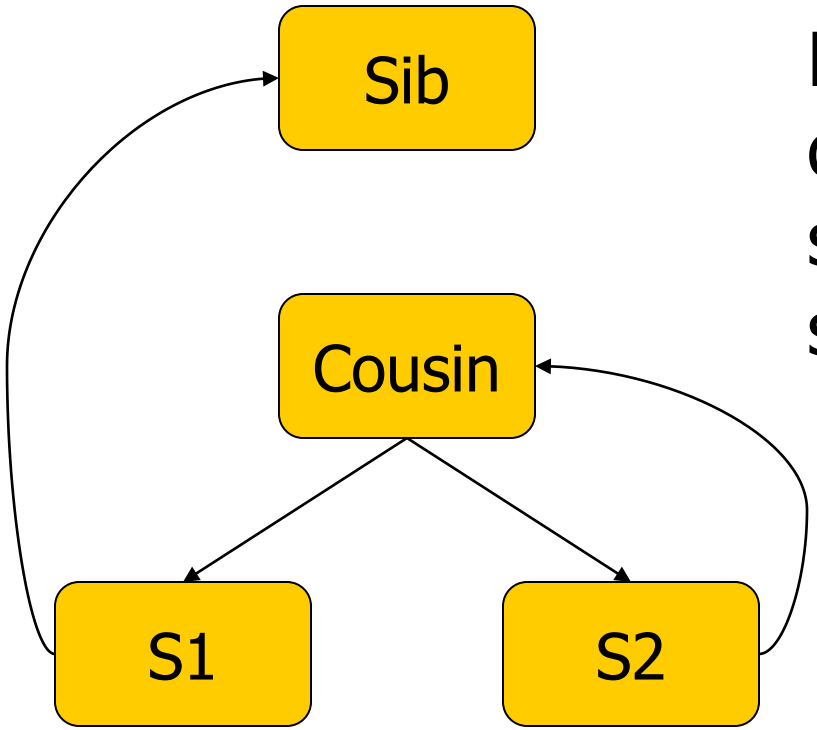
- Arco *P* -> *Q* :
  1. *P* e' il risultato di una query e *Q* e' una relazione nella lista FROM (non una subquery)
  2. *P* e' il risultato di una query e *Q* e' una sottoquery immediata
  3. *P* e' una sottoquery e Q e' una relazione nella sua lista FROM o una sua immediata sottoquery (come 1 e 2).
- Si metta "–" su un arco se P non e' monotona in Q.
- SQL stratificato = # finito di – sui cammini

Esempio: grafo degli strati

- Nell'esempio Cousin, la struttura della regola era:



Il grafo



Nessun “–”, quindi sicuramente stratificato.

Esempio non monotono

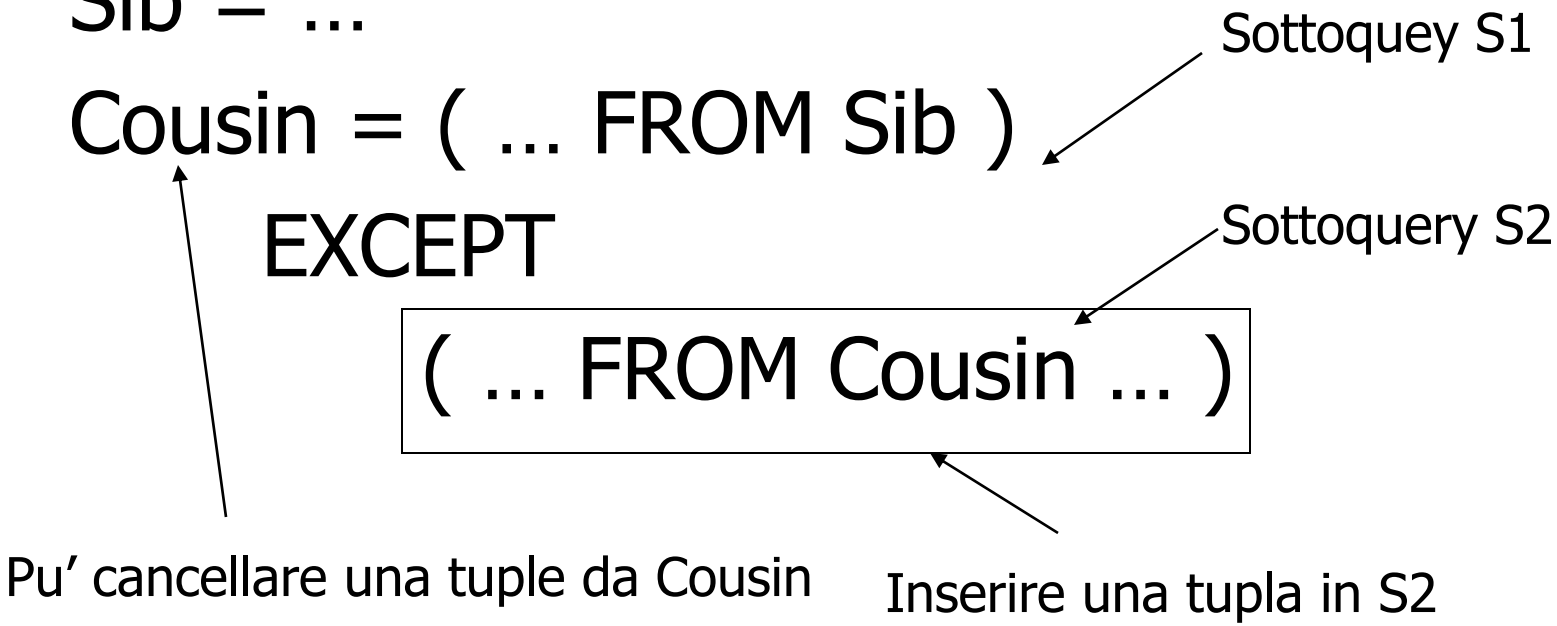
- Cambiare l'UNION in Cousin in EXCEPT:

Sib = …

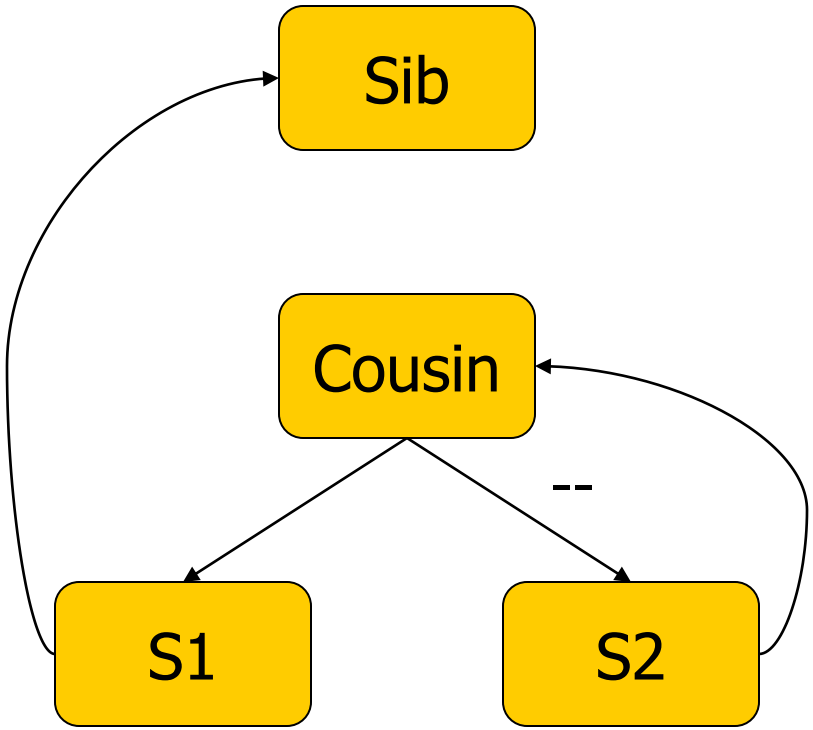
Cousin = (… FROM Sib)

EXCEPT

(… FROM Cousin …)



Il grafo



Nei cicli che coinvolgono Cousin e S2 ci sono un numero infinito di -

NOT non significa nonmonotono

- Non tutti i NOT significano che la query e' non monotona
  - Bisogna considerare ciascun caso separatamente.
- Esempio: negare una condizione in una clausola WHERE cambia solo la condizione di selezione
  - Ma tutte le selezioni sono monotone

Esempio: Cousin rivisto

```
RECURSIVE Cousin AS
   (SELECT * FROM Sib)
        UNION
 (SELECT p1.child, p2.child
  FROM Par p1, Par p2, Cousin
  WHERE p1.parent = Cousin.x AND
      NOT (p2.parent = Cousin.y)
 );
```

Sottoquery S2 rivista

Unica differenza

S2 e' ancora monotono in Cousin

- Intuitivamente, aggiungere una tupla a Cousin non puo' cancellarne da S2
- Tutte le tuple precedenti in Cousin possono fare join con tuple di Par per formare tuple di S2
- In piu', la nuova tupla in Cousin puo' fare join con tuple di Par per generare nuove tuple in S2