OTTIMIZZAZIONE DI INTERROGAZIONI

Nei sistemi di gestione di basi di dati l'utente formula generalmente le proprie interrogazioni utilizzando un linguaggio ad alto livello (come SQL) di tipo dichiarativo: l'utente specifica **cosa** vuole piuttosto che **come** calcolare il risultato dell'interrogazione. Pertanto è lasciato al sistema il compito di individuare una strategia efficiente per calcolare il risultato, cioè di eseguire quel processo che va sotto il nome di **ottimizzazione delle interrogazioni**. Il termine "ottimizzazione" è, in realtà, impreciso in quanto il sistema non individua in generale la strategia ottima, ma una strategia ragionevolmente efficiente per eseguire l'interrogazione. Infatti per trovare la strategia ottima può essere necessario impiegare un tempo eccessivo (se non nel caso di interrogazioni molto semplici) e disporre di informazioni su come i files sono implementati e sul contenuto stesso della base di dati, e queste informazioni potrebbero non essere disponibili nel catalogo del sistema.

Ci sono due tecniche principali di ottimizzazione:

- La prima utilizza regole euristiche, basate su proprietà degli operatori dell'algebra relazionale, per effettuare una trasformazione di una interrogazione in una equivalente (cioè che produce lo stesso risultato) ma più efficiente (che richiede un tempo di esecuzione non superiore a quello richiesto dall'interrogazione iniziale);
- La seconda effettua una stima dei costi di esecuzione delle diverse strategie e sceglie quella con costo minore.

I sistemi utilizzano in genere una combinazione delle due strategie.

Introduciamo la prima tecnica con un esempio. Supponiamo di avere una base di dati relazionale con il seguente schema:

STUDENTE (MATR , NOME , RESIDENZA)
ESAME (MATR , C# , VOTO , DATA)
CORSO (C# , TITOLO , D#)
DOCENTE (D#, NOME , COMUNE)

e di voler conoscere i **nomi** degli **studenti** che hanno preso **30** nel corso di **BasiII**; possiamo esprimere la query in SQL nel modo seguente:

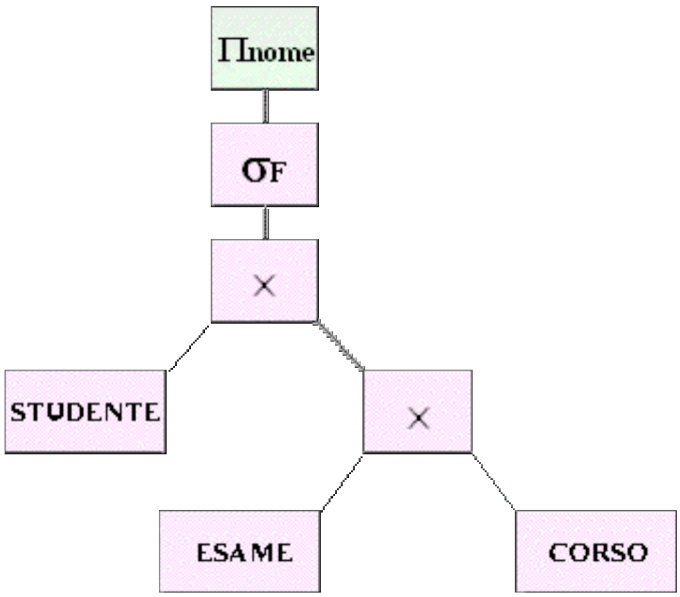
SELECT *NOME*
FROM *STUDENTE S , ESAME E , CORSO C*
WHERE *VOTO=30* ***AND*** *TITOLO="BasiII"* ***AND*** *S.MATR=E.MATR* ***AND*** *E.C#=C.C#*

La stessa query può essere espressa in algebra relazionale nel modo seguente:

$\pi_{NOME}(\sigma_{VOTO=30 \wedge TITOLO='BasiII' \wedge S.MATR=E.MATR \wedge E.C\#=C.C\#}(STUDENTE \times ESAME \times CORSO))$
(dove π è l'operatore di proiezione e σ è l'operatore di selezione).

E' evidente che mentre in SQL (linguaggio dichiarativo) specifichiamo le proprietà che devono essere soddisfatte dalle tuple della relazione "risposta", in algebra relazionale (linguaggio procedurale) specifichiamo le operazioni che devono essere effettuate per produrre tale relazione.

Una volta verificata la correttezza sintattica di un'interrogazione formulata in un linguaggio ad alto livello, il DBMS produce una rappresentazione interna della interrogazione: in genere, l'albero sintattico (**query tree**) di una espressione dell'algebra relazionale, cioè un albero in cui i nodi interni corrispondono a operatori dell'algebra relazionale e le foglie a relazioni della base di dati, ma in alcuni casi può essere un grafo (**query graph**) o un ipergrafo. La query di esempio viene quindi rappresentata, all'interno del DBMS, con il seguente albero:



dove F è la condizione: VOTO=30 ∧ TITOLO='BasiII' ∧ S.MATR=E.MATR ∧ E.C#=C.C#

Questa rappresentazione interna viene utilizzata dall'**ottimizzatore delle interrogazioni** per produrre un **piano di esecuzione** dell'interrogazione. Quest'ultimo viene fornito al **generatore del codice** per produrre il codice che verrà eseguito dal processore del DBMS utilizzando routines di accesso al database che implementano operatori dell'algebra relazionale o combinazioni di tali operatori. L'ottimizzatore può prendere in considerazione solo le strategie che possono essere implementate per mezzo delle routines di accesso disponibili. Pertanto il piano di esecuzione è una sequenza di passi rappresentanti ciascuno:

- l'implementazione di un operatore dell'algebra relazionale oppure
- una combinazione di operatori dell'algebra relazionale.

Ad esempio, una routine di accesso alla base di dati che esprima la combinazione di operatori $\pi_{A1\ldots Ak}(\sigma_C(R \times S))$ può essere implementata nel modo seguente: per ogni coppia di tuple t di R e t' di S, se t e t' soddisfano la condizione C le componenti delle due tuple relative agli attributi $A_1 \ldots A_k$ vengono memorizzate in una relazione temporanea.

Proprietà degli operatori dell'algebra relazionale

I metodi che utilizzano proprietà degli operatori dell'algebra relazionale per trasformare una interrogazione in un'altra la cui valutazione è più efficiente, sono basati su un concetto di equivalenza di espressioni dell'algebra relazionale.

Una espressione relazionale che ha per argomenti gli schemi di relazione $R_1, R_2, \ldots R_k$ definisce una funzione il cui dominio è l'insieme delle k-ple $(r_1, r_2, \ldots r_k)$ dove ciascun r_i è una istanza di relazione sullo schema R_i. Il valore della funzione si ottiene sostituendo r_i a R_i e valutando l'espressione.

Due espressioni sono **equivalenti** se sostituendo la stessa istanza di relazione a ciascun nome di relazione i risultati della valutazione delle due espressioni sono la stessa relazione.

Si ricordi che una relazione è un insieme di tuple e che ogni tupla è un mapping da un insieme di attributi a un insieme di valori nei corrispondenti domini. Per quanto riguarda gli attributi della relazione risultante, questi vengono presi dalle relazioni operando (eventualmente se un attributo è in entrambi gli operandi si usa la notazione R.A) per il prodotto cartesiano e il join; per unione, intersezione e differenza si assume che i nomi degli attributi siano forniti.

Nel seguito indichiamo con E (eventualmente con pedice) un'espressione dell'algebra relazionale e con F (eventualmente con pedice) una condizione sugli attributi di E. Valgono le seguenti equivalenze:

1. Commutatività degli operatori binari ($\times$, $\cup$, $\bowtie$, $\cap$)
   $E_1 \square E_2 \equiv E_2 \square E_1$, dove $\square \in \{\times, \cup, \bowtie, \cap\}$
2. Associatività degli operatori binari ($\times$, $\cup$, $\bowtie$, $\cap$)
   $(E_1 \square E_2) \square E_3 \equiv E_1 \square (E_2 \square E_3)$, dove $\square \in \{\times, \cup, \bowtie, \cap\}$
3. Cascata di proiezioni
   $\pi_{A1,A2,\ldots An}(\pi_{B1,B2,\ldots Bm}(E)) \equiv \pi_{A1,A2,\ldots An}(E)$, se $\{A_1,A_2,..A_n\} \subseteq \{B_1,B_2,..,B_m\}$

Ad esempio:
$\pi_{NOME}(STUDENTE) \equiv \pi_{NOME}(\pi_{NOME,MATR}(STUDENTE))$

4. Cascata di selezioni
   $\sigma_{F1 \textbf{ and } F2}(E) \equiv \sigma_{F1}(\sigma_{F2}(E)) \equiv \sigma_{F2}(\sigma_{F1}(E))$
5. Commutabilità degli operatori unari (σ e π)
   $\pi_{A1,A2,\ldots An}(\sigma_F(E)) \equiv \sigma_F(\pi_{A1,A2,\ldots An}(E))$, se F coinvolge solo attributi in $\{A_1,A_2,..A_n\}$ {e, più in generale
   $\pi_{A1,A2,\ldots An}(\sigma_F(E)) \equiv \pi_{A1,A2,\ldots An}(\sigma_F(\pi_{B1,B2,\ldots Bm}(E)))$, se F coinvolge solo attributi in $\{B_1,B_2,..,B_m\}$ e $\{A_1,A_2,..A_n\} \subseteq \{B_1,B_2,..,B_m\}$

Esempio:
$\pi_{NOME}(\sigma_{RESIDENZA='ROMA'}(STUDENTE)) \equiv \pi_{NOME}(\sigma_{RESIDENZA='ROMA'}(\pi_{NOME,RESIDENZA,MATR}(STUDENTE)))$

6. Commutabilità di selezione e prodotto cartesiano
   $\sigma_F(E_1 \times E_2) \equiv \sigma_F(E_1) \times E_2$, se F coinvolge solo attributi di E_1
   $\sigma_F(E_1 \times E_2) \equiv \sigma_{F1}(E_1) \times \sigma_{F2}(E_2)$, se $F=F_1 \wedge F_2$, F_1 coinvolge solo attributi di E_1 ed F_2 coinvolge solo attributi di E_2
   $\sigma_F(E_1 \times E_2) \equiv \sigma_{F2}(\sigma_{F1}(E_1) \times E_2)$, se F_1 coinvolge solo attributi di E_1 ed F_2 coinvolge attributi di E_1 ed E_2
7. Commutabilità di selezione e unione
   $\sigma_F(E_1 \cup E_2) \equiv \sigma_F(E_1) \cup \sigma_F(E_2)$
8. Commutabilità di selezione e differenza
   $\sigma_F(E_1 - E_2) \equiv \sigma_F(E_1) - \sigma_F(E_2)$

Si osservi che nella 7 e nella 8 le varie occorrenze della condizione *F* possono differire per tener conto dei diversi nomi degli attributi.

Esempio: Supponiamo di volere i dati degli studenti e dei docenti che risiedono a Roma e di aver scelto per la relazione *STUDENTE* $\cup$ *DOCENTE* gli attributi *CODICE,NOME,COMRES.* In tal caso si ha la seguente equivalenza:
$\sigma_{COMRES='ROMA'}(STUDENTE \cup DOCENTE) \equiv \sigma_{RESIDENZA='ROMA'}(STUDENTE) \cup \sigma_{COMUNE='ROMA'}(DOCENTE)$

9. Commutabilità di selezione e join naturale
   $\sigma_F(E_1 \bowtie E_2) \equiv \sigma_F(E_1) \bowtie \sigma_F(E_2)$, se F coinvolge attributi comuni di E_1 ed E_2
10. Commutabilità di proiezione e prodotto cartesiano
    $\pi_{A1,A2,\ldots An}(E_1 \times E_2) \equiv \pi_{B1,B2,\ldots Bm}(E_1) \times \pi_{C1,C2,\ldots Ck}(E_2)$, se $B_1,B_2,..,B_m$ sono attributi di E_1, $C_1,C_2,..,C_k$ sono attributi di E_2 e $\{A_1,A_2,..A_n\} \subseteq \{B_1,B_2,..,B_m, C_1,C_2,..,C_k\}$
11. Commutabilità di proiezione e unione
    $\pi_{A1,A2,\ldots An}(E_1 \cup E_2) \equiv \pi_{A1,A2,\ldots An}(E_1) \cup \pi_{A1,A2,\ldots An}(E_2)$

Si noti che anche nella regola 11 se i nomi degli attributi di E_1 e/o E_2 sono diversi da quelli di $E_1 \cup E_2$, le formule devono essere modificate mettendo i nomi appropriati agli attributi.

ATTENZIONE:
Non esiste una equivalenza analoga alla **8** per la proiezione
(Ovvero π non commutativo con -)

Vediamo perché $\pi_{A1,A2,\ldots An}(R - S) \neq \pi_{A1,A2,\ldots An}(R) - \pi_{A1,A2,\ldots An}(S)$ con un semplice controesempio.

Consideriamo le seguenti relazioni R ed S:

R

A	B
a	b
a	c

S

A	B
a	b

E' facile vedere che

$$\sigma_{\square}(R - S) \neq \sigma_{\square}(R) - \sigma_{\square}(S)$$

Infatti $R - S = \{(a,c)\}$ e quindi $\sigma_{\square}(R - S) = \{(a)\}$; mentre $\sigma_{\square}(R) - \sigma_{\square}(S) = \{(a)\} - \{(a)\} = \varnothing$

Vedremo ora un algoritmo che utilizza le proprietà sopra elencate per trasformare una espressione dell'algebra relazionale in una equivalente più efficiente. I principi alla base dell'algoritmo sono:

1. effettuare le selezioni e le proiezioni prima possibile
2. combinare selezioni e prodotti cartesiani in modo da effettuare join
3. combinare sequenze di operazioni unarie (selezione e proiezione)

Un ulteriore aumento dell'efficienza si può ottenere ricercando sottoespressioni ripetute in una espressione, in modo da calcolare il risultato una volta sola (sottoespressioni ripetute compaiono tipicamente quando una interrogazione coinvolge una vista, in quanto ogni occorrenza di una vista è sostituita dalla stessa sottoespressione).

L'output dell'algoritmo è costituito da un **programma** (il piano di esecuzione) che è una sequenza di passi; ogni passo può consistere di una:

1. applicazione di una singola selezione o proiezione ($\sigma_F(R)$, $\pi_{A1,A2,....An}(R)$)
2. applicazione di una singola selezione seguita da una proiezione ($\pi_{A1,A2,....An}(\sigma_F(R))$)
3. applicazione di prodotto cartesiano, unione o differenza a selezioni o proiezioni, eventualmente seguita da una selezione o da una proiezione (esempio: $\pi_{A1,A2,....An}(\sigma_F(\sigma_{F1}(R_1) \times \sigma_{F2}(R_2)))$)

I passi di tipo 1 e 2 vengono eseguiti esaminando una alla volta le tuple della relazione operando e memorizzando il risultato in una relazione temporanea. I passi di tipo 3 vengono eseguiti applicando la selezione o proiezione a ciascuna tupla delle relazioni operando e la successiva selezione e/o proiezione a ciascuna tupla generata dal prodotto cartesiano (o unione o differenza) e memorizzando il risultato in una relazione temporanea. Quando un prodotto cartesiano non è seguito da una selezione può essere preferibile effettuare le selezioni/proiezioni interne e memorizzarle in una relazione temporanea prima di effettuare il prodotto cartesiano.

Algoritmo Trasformazione di una espressione dell'algebra relazionale in una equivalente.
Input Un'espressione dell'algebra relazionale.
Output Un programma (piano di esecuzione) per valutarla.
Metodo Esegui la seguente sequenza di passi:

1. Usa la regola 4 per ottenere una cascata di selezioni da ciascuna selezione
2. Usa le regole da 4 a 9 per spostare le selezioni il più in basso possibile nell'albero sintattico dell'espressione
3. Usa le regole 3, 10, 11 e 5 per spostare le proiezioni il più in basso possibile nell'albero sintattico dell'espressione
4. Usa le regole 3-5 per combinare cascate di selezioni e proiezioni in una singola proiezione o in una selezione seguita da una proiezione
5. Partiziona i nodi dell'albero sintattico risultante in gruppi nel modo seguente:
   ogni nodo non foglia che rappresenta una operazione binaria (prodotto cartesiano, unione, differenza) è raggruppato insieme ai suoi immediati antenati che rappresentano operazioni unarie (selezione e proiezione, fino ad incontrare il primo operatore che non è unario) e insieme a qualsiasi catena di discendenti, che rappresentano operazioni unarie, e che termini in una foglia
6. Esamina i gruppi ottenuti al passo precedente in un qualsiasi ordine tale che nessun gruppo è esaminato prima dei suoi discendenti, e per ciascun gruppo genera un passo del programma (passo del piano di esecuzione).

Esempio. Consideriamo una base di dati con lo schema seguente:

CLIENTE (C#, NOME, IND)
ORDINE (C#, A#, DATA, QUANT)
ARTICOLO (A#, DENOM, PREZZO)

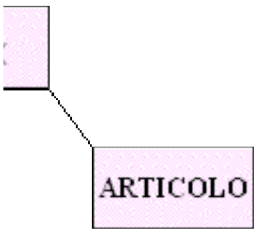
E supponiamo di voler conoscere i nomi dei clienti che hanno ordinato articoli con prezzo maggiore di 1000 in quantità maggiore di 100 in questo anno (2002). A tale scopo possiamo effettuare la seguente query SQL:

```
SELECT   NOME
FROM     CLIENTE C, ORDINE O, ARTICOLO A
WHERE    C.C#=O.C# AND O.A#=A.A# AND A.PREZZO>1000
         AND O.QUANT>100 AND O.DATA>1/1/02
```

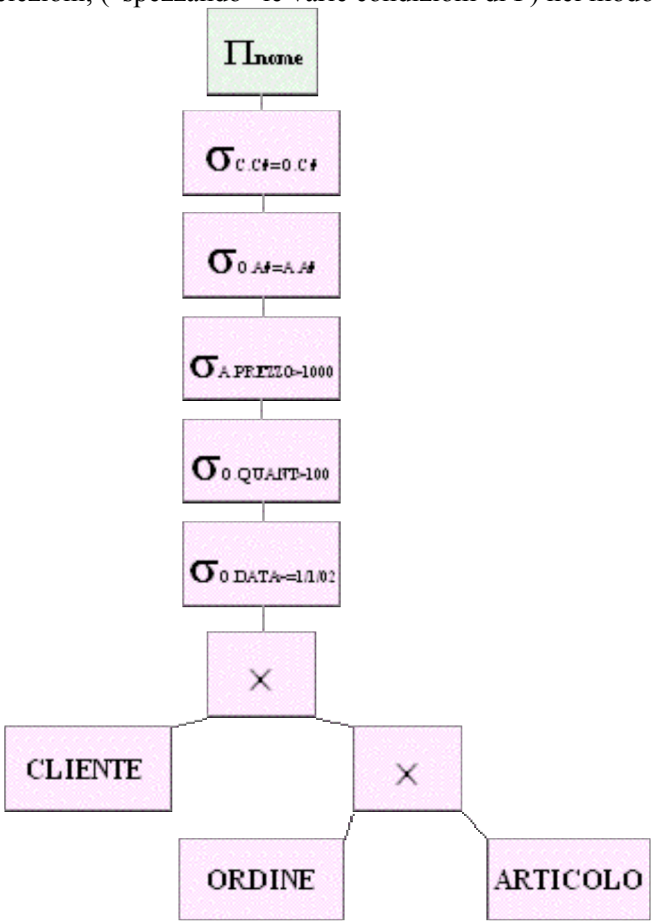
che può essere tradotta dal sistema nella seguente espressione dell'algebra relazionale:

Π_{NOME} (σ_F (CLIENTE × (ORDINE × ARTICOLO)))
dove F = C.C# O.C# **AND** O.A#=A.A# **AND** A.PREZZO>1000 **AND** O.QUANT>100 **AND** O.DATA>=1/1/02

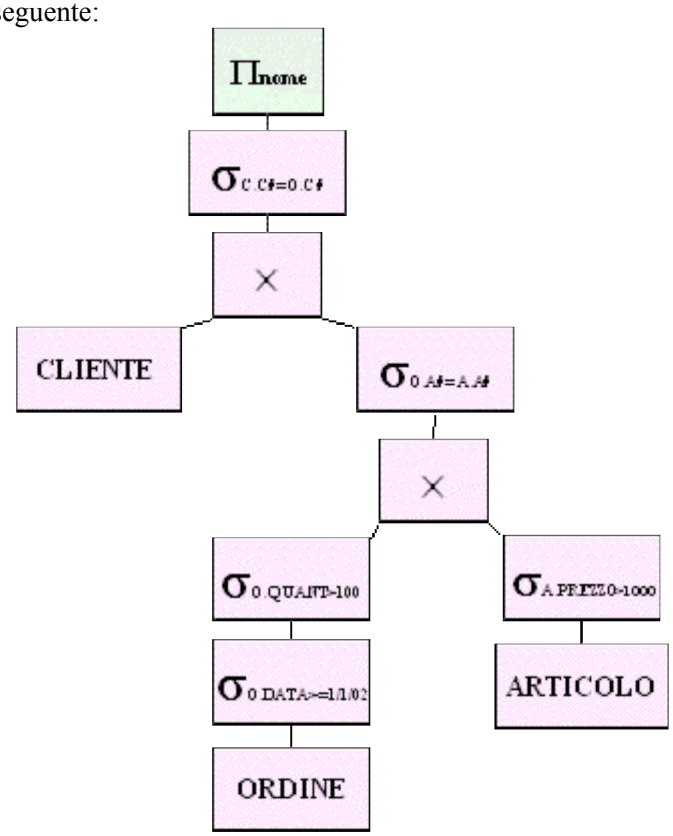
il cui albero sintattico è:



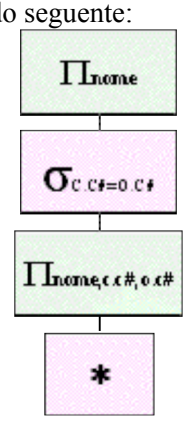
Applicazione del passo 1 dell'algoritmo:
Otteniamo una cascata di selezioni, ("spezzando" le varie condizioni di F) nel modo seguente:



Applicazione del passo 2 dell'algoritmo:
la condizione O.DATA>=1/1/02, coinvolge solo un attributo della relazione ORDINE, quindi posso spostare la selezione $\sigma_{O.DATA>=1/1/02}$ in modo che sia effettuata prima di ORDINE $\times$ ARTICOLO riducendo le dimensioni degli operandi del prodotto cartesiano.
Analoghe considerazioni possono essere fatte per le selezioni $\sigma_{O.QUANT>100}$ e per $\sigma_{A.PREZZO>1000}$.
D'altra parte la selezione $\sigma_{O.A\#=A.A\#}$ non può essere effettuata prima del prodotto cartesiano ORDINE $\times$ ARTICOLO in quanto la condizione di selezione coinvolge attributi di entrambe le relazioni ORDINE e ARTICOLO. Analoghe considerazioni possono essere fatte per la selezione $\sigma_{O.C\#=C.C\#}$.
Si ottiene quindi l'albero seguente:



Applicazione del passo 3 dell'algoritmo:
Applichiamo prima la regola 5 (commutatività di selezione e proiezione)
L'espressione π_{NOME} ($\sigma_{C.C\#=O.C\#}$(…)) (proiezione di una selezione) è equivalente alla seguente:
π_{NOME} ($\sigma_{C.C\#=O.C\#}$($\pi_{NOME,C.C\#,O.C\#}$(…)))
Quindi l'albero può essere trasformato nel modo seguente:

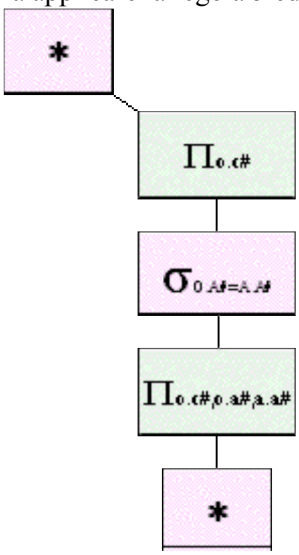


(nel punto * l'albero resta invariato).

Applicando la regola 10 l'albero si trasforma nel modo seguente:

(nel punto * l'albero resta invariato)

In presenza di $\Pi_{O.C\#}$ ($\Pi_{O.A\#=A.A\#}$) posso ancora applicare la regola 5 ed ottenere:



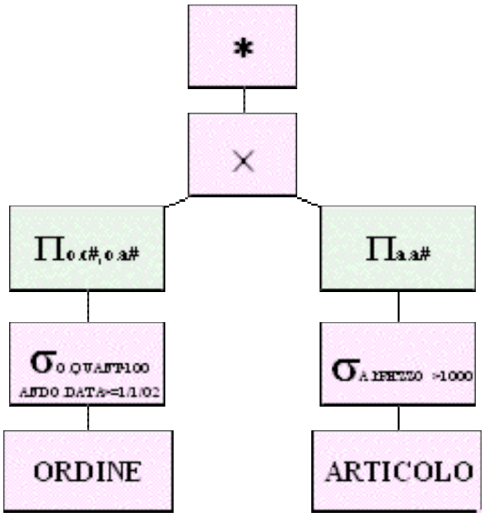
(nel punto * l'albero resta invariato)

Il punto in cui $\square_{O.C\#,O.A\#=A.A\#}$ mi permette di applicare ancora la regola 10 ed ottenere:

(nel punto * l'albero resta invariato)

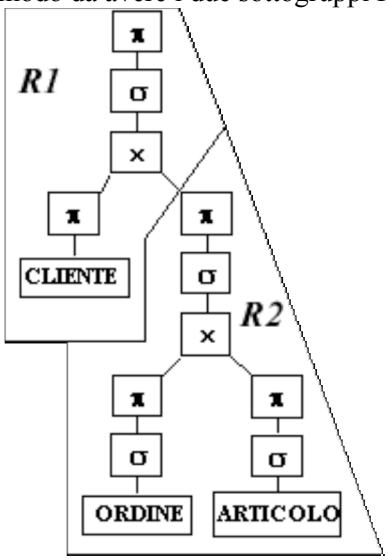
A questo punto potrei far scendere ancora le proiezioni, ma devo tener conto del passo 4 dell'algoritmo che mi permette di avere tutte le proiezioni e tutte le selezioni insieme (tornerei alla situazione precedente)

Applicazione del passo 4 dell'algoritmo:
Ricompattiamo in modo da avere:



(nel punto * l'albero resta invariato)

Applicazione del passo 5 dell'algoritmo:
Evidenziamo i due gruppi risultanti in modo da avere i due sottogruppi R1 e R2:



Applicazione del passo 6 dell'algoritmo
R1 non può essere esaminato prima di R2 (perché R2 è discendente da R1). L'analisi di R2 genera l'espressione dell'Algebra Relazionale seguente:

$R2 = \pi_{O.C\#} (\pi_{O.A\#=A.A\#} (\pi_{O.C\#,O.A\#} (\sigma_{O.QUANT>100 \text{ AND } O.DATA>=1/1/02} (ORDINE)) \times \pi_{A.A\#} (\sigma_{A.PREZZO>1000} (ARTICOLO))))$

A questo punto $R1 = \pi_{NOME} (\pi_{C.C\#=O.C\#} (\pi_{NOME,C\#} (CLIENTE) \times R2))$

Questo permette, in base al discorso già fatto sulle routine che implementano combinazioni di operatori dell'algebra relazionale (quale è R1), di generare una sola relazione intermedia per ottenere R2, anziché generare una relazione intermedia per ogni singolo operatore.

Metodi basati sulla stima dei costi

Un sistema di basi di dati relazionali con un linguaggio di interrogazione ad alto livello deve fornire un insieme di metodi o algoritmi per implementare gli operatori dell'algebra relazionale (o loro combinazioni) che possono comparire in una strategia di esecuzione di una interrogazione. Per ciascun operatore possono essere disponibile più **routines di accesso**. Ciascuna di queste routines è associata a una particolare struttura di organizzazione fisica (file hash, file con indice, ecc.) e può quindi essere utilizzata solo se i files coinvolti nell'operazione hanno quella particolare struttura.

Un sistema quindi effettua una stima dei costi (misurata in numero di accessi alla memoria di massa) delle varie implementazioni degli operatori e sceglie quella più conveniente.

Nell'effettuare una stima dei costi si terrà conto dei seguenti parametri:

- T_R : numero di tuple nella relazione R
- B_R : numero di blocchi su cui può essere memorizzata la relazione R (T_R/B_R rappresenta il numero di tuple di R che possono essere memorizzate in un blocco)
- $I_{R.A}$ (o semplicemente I_A, se non c'è ambiguità): numero di valori distinti per l'attributo A presenti nell'istanza corrente di R (**image size** di A in R).
- dom $_R$ (A): dominio dei valori dell'attributo A nella relazione R, ovvero l'insieme di valori presenti per A nell'istanza corrente di R al momento in cui si applica un operatore. Si noti che |dom $_R$ (A)| = I_A

Operatore di Selezione

Premesse:

In questo paragrafo prenderemo in considerazione una strategia che può essere usata (ed è usata in System R) per rispondere a interrogazioni del tipo:

SELECT	A1, A2..An	
FROM	R	(1)
WHERE	$C_1 \square C_2 \square \square C_m$	

cioè di selezioni in cui la condizione sia l'**and** di condizioni che possono essere composte di sottocondizioni connesse da AND, OR, NOT.

Per effettuare una stima del costo (numero di accessi a memoria secondaria necessari) di una tale operazione occorre considerare la **Selettività** di ciascuna condizione semplice, cioè il rapporto tra il numero di tuple di R che soddisfano la condizione e il numero totale delle tuple di R; se si assume che la distribuzione dei valori sia uniforme la selettività della condizione A='a' è data da $1 / I_A$ ($=(T_R/ I_A) / T_R$)).

Un altro fattore da tenere presente è l'esistenza di indici per la relazione R. Distinguiamo due tipi di indici**:**

- **indici clustering**
- **indici nonclustering.**

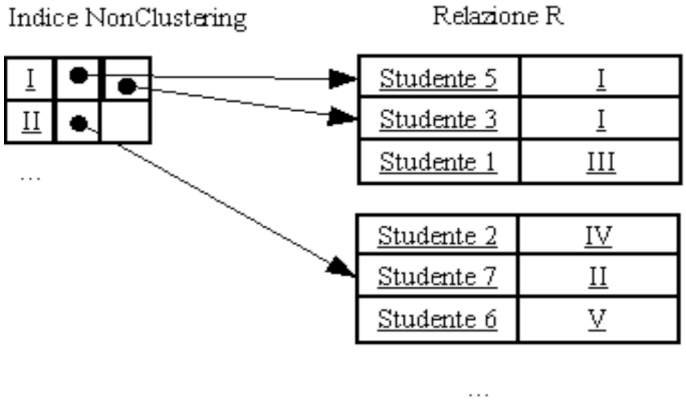
Indice clustering

Se su un attributo A di R è definito un indice clustering (ovvero raggruppante) questi è solitamente di tipo ISAM, nel quale il file principale è ordinato in base al valore di un campo (o di un insieme di campi) del record, come ad esempio nell'immagine seguente:

In questo caso, assumendo che la distribuzione dei valori sia uniforme, il numero di blocchi che contengono le T_R / I_A tuple di R per cui A ='a' è approssimativamente il numero di blocchi in cui tali tuple possono essere memorizzate (B_R / I_A).

Indice nonclustering

Se in un DB contenente, ad esempio, i dati degli studenti abbiamo definito un indice denso sull'attributo AnnodiCorso, si verifica il seguente scenario, in cui abbiamo tanti puntatori quante sono le tuple con un determinato valore nel campo 'AnnodiCorso'. Un indice di questo tipo è detto nonclustering (non raggruppante) in quanto le tuple con uno stesso valore per l'attributo su cui è definito l'indice possono trovarsi ciascuna in un blocco diverso.



Quindi, se su un attributo A di R è definito un indice nonclustering, il numero di blocchi che contengono tuple di R per cui A='a' è stimato uguale al numero di tuple di R per cui A ='a' (T_R / I_A)

Per stimare il costo di un'operazione di selezione del tipo $\sigma_{A='a'}(R)$, dove R è una relazione che ha un indice sull'attributo A, assumiamo che il numero di blocchi che devono essere letti è dato dal numero di blocchi che contengono tuple di R che soddisfano la condizione A='a', senza tener conto del numero di blocchi del file indice che devono essere letti (in generale, tale numero è molto più piccolo del numero di blocchi di R che devono essere letti). Con le assunzioni fatte una stima del costo (numero di accessi necessari) per calcolare $\sigma_{A='a'}(R)$ è data da :

- T_R / I_A, se su A è definito un indice nonclustering (infatti, assumendo che i valori possono essere assunti con uguale probabilità, il numero di tuple di R che soddisfano la condizione A='a' è T_R/ I_A, e, poiché l'indice su A è nonclustering tali tuple potrebbero essere memorizzate in altrettanti blocchi)
- B_R / I_A, se su A è definito un indice clustering.

La strategia utilizzata da System R per valutare interrogazioni del tipo (1) ha come parametri di input:

- gli attributi di R su cui è definito un indice e, per ciascuno di questi il tipo dell'indice (clustering o nonclustering)
- i valori stimati per T_R e B_R
- gli image size stimati per ciascun attributo di R su cui è definito un indice.

La strategia esamina in sequenza una lista di metodi per calcolare la risposta (per i metodi che richiedono la scelta di un indice devono essere considerate tutte le possibili scelte), effettua una stima del costo di ciascun metodo e sceglie quello con costo minore.

1. Se una condizione C_i è della forma A='a' e sull'attributo A è definito un indice clustering, utilizza tale indice per ricercare tutte le tuple di R che soddisfano C_i; per ciascuna di tali tuple verifica che siano soddisfatte le altre condizioni.
   Costo stimato: B_R / I_A.
2. Se una condizione C_i è della forma Aθ'a' ($\theta \in \{<, >, \leq, \geq\}$) e sull'attributo A è definito un indice clustering, utilizza tale indice per ricercare tutte le tuple di R che soddisfano C_i; per ciascuna di tali tuple verifica che siano soddisfatte le altre condizioni.
   Costo medio stimato: B_R / 2.

Infatti, se assumiamo senza perdita di generalità che θ sia l'operatore di confronto $\leq$, cioè C_i è la condizione A$\leq$'a', possiamo osservare che:

- assumendo che la distribuzione dei valori sia uniforme, il numero di valori distinti di A in un blocco è I_A / B_R
- se a è un valore di A che si trova nell'*i*-esimo blocco, *i* accessi sono sufficienti per ricercare tutte le tuple che soddisfano la condizione A$\leq$'a'

Pertanto una stima del costo medio può essere ottenuta come segue:

$$\frac{\frac{I_A}{B_R} * (1 + 2 + .. + B_R)}{I_A} = \frac{I_A}{B_R} * \frac{B_R * (B_R + 1)}{2 * I_A} \approx \frac{B_R}{2}$$

3. Se una condizione C_i è della forma A='a', e sull'attributo A è definito un indice nonclustering, utilizza tale indice per ricercare tutte le tuple di R che soddisfano C_i; per ciascuna di tali tuple verifica che siano soddisfatte le altre condizioni.
   Costo stimato: T_R / I_A.
4. Se R è memorizzata in forma compatta, ricerca tutte le tuple di R e per ogni tupla verifica che siano soddisfatte tutte le condizioni.
   Costo stimato: B_R.
5. Se R ha un indice clustering su un attributo A, che non compare in alcuna condizione, utilizza tale indice per ricercare tutte le tuple di R e, per ogni tupla, verifica che siano soddisfatte tutte le condizioni.
   Costo stimato: B_R.
6. Se una condizione C_i è della forma Aθ'a' ($\theta \in \{<, >, \leq, \geq\}$) e sull'attributo A è definito un indice nonclustering utilizza tale indice per ricercare tutte le tuple di R che soddisfano C_i; per ciascuna di tali tuple verifica che siano soddisfatte le altre condizioni.
   Costo medio stimato: $T_R/ 2$.
   Assumiamo senza perdita di generalità che θ sia l'operatore di confronto $\leq$, cioè C_i è la condizione A$\leq$'a'. Assumiamo inoltre che sul dominio dei valori di A in R ($dom_R(A)$), sia definito l'ordinamento a_1, a_2, …, a_n.
   Si osservi che, assumendo che la distribuzione dei valori sia uniforme, il numero di tuple di R che hanno il valore a_i per l'attributo A è T_R / I_A e poiche l'indice su A è nonclustering queste tuple possono essere memorizzate in T_R / I_A blocchi distinti
   Pertanto una stima del costo medio è data da:

$$\frac{\frac{T_R}{I_A} + 2\frac{T_R}{I_A} + 3\frac{T_R}{I_A} + .. + I_A\frac{T_R}{I_A}}{I_A} = \frac{T_R}{I_A} * (1 + 2 + .. + I_A) * \frac{1}{I_A} = \frac{T_R}{I_A} * \frac{I_A * (I_A + 1)}{2} * \frac{1}{I_A} \approx \frac{T_R}{2}$$

7. Se R ha un indice nonclustering su un attributo A che non compare in alcuna condizione, utilizza tale indice per ricercare tutte le tuple di R, e per ogni tupla, verifica che siano soddisfatte tutte le condizioni.
   Costo stimato: T_R.
8. Se nessuno dei metodi precedenti può essere utilizzato, esamina tutti i blocchi che possono contenere tuple di R e per ogni tupla di R verifica che siano soddisfatte tutte le condizioni.
   Costo stimato: T_R (o più, se non si sa quali blocchi possono contenere tuple di R).

Vediamo ora un esempio di applicazione di questa analisi delle strategie .
Supponiamo di avere la tabella:

ORDINE (F# , O#, NART ,QUANT)

e di voler effettuare la seguente query su singola relazione:

SELECT *O#*
FROM *ORDINE*
WHERE *QUANT>=5 AND NART='bullone'*

Che restituisce l'identificativo degli ordini con cui sono stati ordinati di bulloni in quantità di almeno 5 pezzi.

Ipotesi: La relazione *ORDINE* ha
- un indice clustering su O#
- indici nonclustering su NART e QUANT.

Input:
- il numero di tuple della relazione ORDINE: $T_{ORDINE} = 1000$
- la capacità dei blocchi: 10 tuple (pertanto: $B_{ORDINE} = 100$)
- l'image size di NART: $I_{NART} = 50$

Metodo 1: Non applicabile perché ho NART='bullone' e su NART è definito un indice nonclustering.
Metodo 2: Non applicabile perché ho QUANT ≥ 5 e su QUANT è definito un indice nonclustering.
Metodo 3: Applicabile; costo = $T_{ORDINE} / I_{NART} = 1000 / 50 = 20$ accessi
Metodo 4: Supponendo ORDINE memorizzata in forma compatta, il costo è il numero di blocchi $B_{ORDINE} = 100$
Metodo 5: Applicabile; costo = $B_{ORDINE} = 100$ accessi.
Metodo 6: Applicabile; costo = $T_{ORDINE} / 2 = 1000 / 2 = 500$ accessi
Metodo 7: Non applicabile perché gli attributi su cui ho un indice nonclustering sono tutti nelle condizioni
Metodo 8: Non applicabile in quanto ho supposto già ORDINE memorizzata in forma compatta.

Il costo minore si ha nel caso dell'applicazione del Metodo 3 che verrà scelto come migliore strategia possibile.

Operatore di Prodotto cartesiano
In questo paragrafo vedremo come può essere effettuata una stima del costo di esecuzione del prodotto cartesiano di due relazioni R ed S (R × S). A tale scopo assumeremo che le due relazioni operando siano memorizzate in forma compatta (se così non fosse, dovremmo considerare il costo addizionale per memorizzarle in forma compatta; tale costo è dato, per la relazione R, da T_R+B_R in quanto T_R accessi sono necessari per leggere R e B_R accessi sono necessari per scrivere R in forma compatta). Considereremo il costo dell'implementazione di un operando come la somma del costo dell'operazione di lettura delle relazioni operando (cioè le relazioni sulle quali è applicato l'operatore) e del costo dell'operazione di scrittura della relazione risultante (memorizzazione del risultato).
Inoltre, denoteremo con:
- U il numero di blocchi necessari per memorizzare il risultato di R×S, e con
- M il numero di blocchi disponibili in memoria principale

Costo per la scrittura
Cominciamo con il fornire una stima di U in termini dei seguenti parametri di R ed S:
- l_R che rappresenta il numero di bytes necessari per memorizzare una tupla di R (lunghezza di un record di R)
- l_S che rappresenta il numero di bytes necessari per memorizzare una tupla di S (lunghezza di un record di S)
- b che rappresenta il numero di bytes che costituiscono un blocco.

Il numero di bytes necessari per memorizzare 1 tupla di R×S è dato da l_R+l_S ; poiché le tuple di R×S sono T_R*T_S, allora il numero di bytes necessari a memorizzare tutto il prodotto cartesiano è: $T_R*T_S * (l_R+l_S)$.

Il numero di blocchi necessari a memorizzare tutto il prodotto cartesiano sarà quindi:

$$\frac{T_R * T_S * (l_R + l_S)}{b} = \frac{T_R * l_R}{b} * T_S + T_R * \frac{T_S * l_S}{b}$$

Poiché $(T_R * l_R) / b$ è circa uguale al numero (B_R) di blocchi necessari per memorizzare R e, analogamente $(T_S * l_S) / b$ è circa uguale al numero (B_S) di blocchi necessari per memorizzare S, si ha che:

$$U \approx B_R * T_S + B_S * T_R \qquad (2)$$

Costo per la lettura

Per fornire una stima del numero di accessi necessari per leggere gli operandi, assumiamo (senza perdita di generalità) che $B_S \leq B_R$. Inoltre, distinguiamo due casi:

1. $M > B_S$. In tal caso si ha sufficiente spazio affinché i blocchi di S possano essere mantenuti tutti in memoria principale, e quelli di R uno alla volta. Pertanto il costo per leggere R ed S è dato da
$$B_R + B_S \qquad (3)$$
2. $M \leq B_S$. In tal caso possiamo procedere nel modo seguente. Suddividiamo S in segmenti costituiti da $M-1$ blocchi. Carichiamo un segmento alla volta in memoria principale e ogni volta utilizziamo il rimanente blocco di M per leggere (uno alla volta) tutti i blocchi di R. Poiché il numero di segmenti di S è circa
$B_S / (M-1)$, il costo per leggere tutto l'input è dato da:
$$(B_R * B_S/(M-1)) + B_S \qquad (4),$$

dove $B_R * B_S/(M-1)$ è il costo per leggere R, e B_S è il costo per leggere S.

Se alla (3) e alla (4) aggiungiamo la (2) otteniamo la stima del costo di $R \bowtie S$ nei due casi.
Si osservi che il costo per produrre l'output non può essere migliorato. Per quanto riguarda il costo per leggere l'input, la (4) fornisce un upper bound, mentre il seguente teorema fornisce un lower bound, e ci dice che anche il costo per leggere l'input non può essere migliorato in modo significativo rispetto alla (3).

Teorema. Se il numero di blocchi disponibili in memoria principale è M, per calcolare $R \bowtie S$ il numero di accessi necessari per leggere R ed S è almeno $B_S * B_R/(M-1)$.

Dimostrazione. Ogni volta che viene letto un blocco di R, il massimo numero di tuple di $R \bowtie S$ che possono essere generate ad ogni accesso, avendo a disposizione altri M-1 blocchi, è dato da $(M-1)*(T_S/B_S)*(T_R/B_R)$ (dove T_S / B_S e T_R / B_R rappresentano, rispettivamente, il numero di tuple di S e di R in un blocco); questo valore si ottiene quando i rimanenti $M-1$ blocchi in memoria principale sono tutti blocchi di S. Analoga considerazione può essere fatta nel caso che venga letto un blocco di S. Pertanto ogni volta che viene letto un blocco di uno dei due operandi (cioè ogni volta che viene fatto una accesso per leggere un blocco di uno dei due operandi) il massimo numero di tuple di $R \bowtie S$ che possono essere generate è dato da $(M-1)*(T_S/B_S)*(T_R/B_R)$. D'altra parte il numero complessivo di tuple generate deve essere $T_S * T_R$ (numero di tuple di $R \bowtie S$). Pertanto, se indichiamo con A il numero di accessi in lettura necessari per leggere gli operandi si ha:

$$T_S * T_R \leq A*(M-1)*(T_S / B_S)*(T_R / B_R)$$

da cui si ottiene

$$A \geq B_S B_R / (M-1).$$

Join naturale

Anche nel caso del join naturale per ottenere una stima del costo dobbiamo sommare il costo necessario per leggere l'input (le relazioni operando) e quello necessario per scrivere l'output (la relazione risultante).

Come vedremo il primo costo (lettura) dipende sia dal metodo usato per calcolare il join naturale sia dal tipo di organizzazione fisica degli operandi.

Il secondo costo (scrittura) dipende dal numero di tuple delle relazioni operando che hanno gli stessi valori sugli attributi comuni. Poiché tale numero in generale non è conosciuto, per effettuare una stima del secondo costo occorre fare riferimento ad un ragionevole modello statistico per le relazioni. Faremo pertanto le seguenti due assunzioni:

i. assumiamo che se R ha T_R tuple e il prodotto cartesiano dei domini associati ai suoi attributi ha n tuple ($|dom_R(A_1) \times dom_R(A_2) \times .. \times dom_R(A_i)| = n$), allora ciascuna di queste n tuple ha probabilità T_R/n di essere in R, dove $dom_R(A_i)$ (dominio di A_i in R) è l'insieme di valori presenti per l'attributo A_i in R al momento in cui si effettua il join

ii. assumiamo inoltre che quando si fa il join naturale, per ogni attributo comune delle relazioni operando ci deve essere una relazione di contenimento tra i domini associati all'attributo nelle due relazioni.

Vediamo ora come è possibile stimare il costo (in numero di blocchi) per scrivere il risultato di un join naturale.

Supponiamo di voler calcolare il join naturale $R \triangleright\triangleleft S$ con R=AB ed S=BC;

poniamo inoltre:

$D_A=dom_R(A)$,
$D_C=dom_S(C)$,
$E_B=dom_R(B)$,
$D_B=dom_S(B)$,

e assumiamo che $E_B \subseteq D_B$.

Valutiamo la probabilità che una tupla "abc" di $D_A \times D_B \times D_C$ appartenga a $R \triangleright\triangleleft S$.

$$p(abc \in R \triangleright\triangleleft S) = p(ab \in R) \times p(bc \in S). \quad (5)$$

In base alle assunzioni fatte, si ha:

$$p(bc \in S) = \frac{T_S}{|D_B \times D_C|} = \frac{T_S}{I_{S.B} * I_C} \quad (6)$$

Inoltre:

$$p(ab \in R) = p(b \in D_B\text{-}E_B) \times p(ab \in R \mid a \in D_A, b \in D_B\text{-}E_B) + p(b \in E_B) \times p(ab \in R \mid a \in D_A, b \in E_B)$$

Si osservi che $p(ab \in R \mid a \in D_A, b \in D_B\text{-}E_B) = 0$, in quanto nessun valore in $D_B\text{-}E_B$ compare in R. Inoltre in base alle assunzioni fatte, si ha che:

$p(b \in E_B) = |E_B| / |D_B| = I_{R.B} / I_{S.B}$ e che:

$$p(ab \in R \mid a \in D_A, b \in E_B) = \frac{T_R}{|D_A \times E_B|} = \frac{T_R}{I_A * I_{R.B}}.$$

Pertanto:

$$p(ab \in R) = \frac{I_{R.B}}{I_{S.B}} * \frac{T_R}{I_A * I_{R.B}} = \frac{T_R}{I_A * I_{S.B}} \quad (7)$$

Sostituendo la (6) e la (7) nella (5) si ottiene:

$$p(abc \in R \rhd\lhd S) = 0 + \frac{T_R}{I_A * I_{S.B}} * \frac{T_S}{I_{S.B} * I_C} = \frac{T_R * T_S}{I_A * I_{S.B}^2 * I_C}$$

Moltiplicando tale probabilità per il numero (I_A * $I_{S.B}$ * I_C) di tuple in $D_A \times D_B \times D_C$ si ottiene una stima del numero di tuple di $R \rhd\lhd S$

$$T_R * T_S / I_{S.B}$$

Per effettuare una stima del numero di blocchi necessari per memorizzare $R \rhd\lhd S$ possiamo osservare che il numero stimato di tuple di $R \rhd\lhd S$ è pari a $1/I_{S.B}$ per il numero ($T_R * T_S$) di tuple del prodotto cartesiano $R \times S$; pertanto possiamo stimare che il numero stimato dei blocchi necessari per memorizzare $R \rhd\lhd S$ sarà circa $1/I_{S.B}$ per il numero ($B_R * T_S + B_S * T_R$) di blocchi necessari per memorizzare $R \times S$. Si ottiene quindi:la seguente stima del numero di blocchi necessari per memorizzare $R \rhd\lhd S$:

$$(B_R * T_S + B_S * T_R) / I_{S.B}$$

Tale risultato può essere facilmente generalizzato. Pertanto con le assunzioni fatte si ha che:
se

- $R = A_1 A_2 \ldots A_n B_1 B_2 \ldots B_k$ e
- $S = B_1 B_2 \ldots B_k C_1 C_2 \ldots C_m$ e
- $\forall$ i, i=1,…k, abbiamo $dom_R(B_i) \subseteq dom_S(B_i)$ oppure $dom_R(B_i) \supseteq dom_S(B_i)$

allora una stima del numero di tuple di $R \rhd\lhd S$ è data da:

$$T_R * T_S / I$$

e una stima del numero di blocchi necessari per memorizzare $R \times S$ è data da:

$$(B_R * T_S + B_S * T_R) / I$$

dove:

$$I = I_{B1} I_{B2} \ldots I_{Bk}$$

e I_{Bi}, i=1,…k, è l'image size del dominio più grande per B_i, cioè:

$$I_{Bi} = \max(|dom_R(B_i)|, |dom_S(B_i)|)$$

Esaminiamo ora diversi metodi per calcolare il join naturale di due relazioni e vediamo nei vari casi (a seconda del metodo di lettura degli operandi) quale è una stima del costo.

Join per selezione del prodotto cartesiano.
Il modo più ovvio per calcolare il join naturale di due relazioni è quello basato sulla definizione stessa di join naturale: per ogni tupla $\in$ R e per ogni tupla $\in$ S, se $\in$ e $\in$ hanno gli stessi valori per gli attributi comuni viene generata una tupla di $R \rhd\lhd S$. In tal caso, supponendo ad esempio $B_S \leq B_R$, il costo del join naturale è dato dal costo per leggere l'input nel caso del prodotto cartesiano (($B_S * B_R$ / (M $-$ 1)) + B_S)) più il costo per scrivere $R \rhd\lhd S$ ((B_R T_S + B_S T_R) / I).Tale metodo non è molto raffinato, in quanto impone di confrontare il contenuto di ogni blocco di R con il contenuto di ogni blocco di S.

I metodi che seguono cercano di limitare il numero di blocchi delle due relazioni che devono essere confrontati, sfruttando la conoscenza dell'organizzazione fisica dei dati, ed effettuando eventualmente un preprocessamento delle due relazioni.

Join con ordinamento

Tale metodo richiede che le due relazioni siano ordinate in base al valore degli attributi comuni. Ciascuna delle due relazioni viene scandita (dalla prima all'ultima tupla) utilizzando un cursore. Ogni volta che i valori degli attributi comuni sono diversi nelle due tuple su cui sono posizionati i cursori si fa avanzare il cursore della relazione in cui il valore è più piccolo. Quando invece le tuple su cui sono posizionati i cursori hanno uguale valore per gli attributi comuni si cercano tutte le tuple delle due relazioni con quel valore e si fa il join di questi due insiemi di tuple.

Tale metodo può ridurre il costo per la lettura a B_R+B_S. Tuttavia se una o entrambe le relazioni non sono già ordinate in base agli attributi comuni, ai costi dovuti a lettura e scrittura deve essere aggiunto il costo necessario per preprocessare la relazione o le relazioni in modo da ottenere tale ordinamento.

Vediamo quindi quanto costa ordinare una relazione mediante un metodo detto ***multiway merge sort***.

Mostriamone innanzitutto il funzionamento con un esempio.

Dobbiamo ordinare i seguenti 8 blocchi:

7	8	20	4	1	16	15	23
12	2	9	11	24	17	5	22
19	14	10	21	13	3	6	28

Abbiamo 2 blocchi disponibili in memoria principale per leggere questi 8 blocchi da ordinare.

Porto a 2 a 2 i blocchi in memoria principale, e ordino le tuple contenute in ogni coppia di blocchi, ottenendo la configurazione seguente, composta da 4 "gruppi" (ciascuno formato da 2 blocchi) ordinati:

2	12	4	11	1	16	5	22
7	14	9	20	3	17	6	23
8	19	10	21	13	24	15	28

Itero il procedimento esaminando i primi due gruppi, prendendo i primi due blocchi di ciascun gruppo, ed ordinandoli fino ad esaurimento dei blocchi, in modo da ottenere 2 gruppi ordinati ciascuno formato da 4 blocchi:

2	8	11	19	1	6	16	23
4	9	12	20	3	13	17	24
7	10	14	21	5	15	22	28

fino ad ottenere un unico gruppo di 8 blocchi ordinato.

1	4	7	10	13	16	20	23
2	5	8	11	14	17	21	24
3	6	9	12	15	19	22	28

In generale, assumiamo che ci siano M blocchi disponibili in memoria principale per leggere blocchi della relazione che si desidera ordinare. Al primo passo i blocchi della relazione che si vuole ordinare vengono letti a gruppi di M e vengono ordinate le tuple all'interno di ciascun gruppo. Al p-esimo (p>1) passo vengono presi di volta in volta M gruppi di M^{p-1} blocchi ordinati per produrne uno di M^p blocchi ordinato. Il procedimento ha termine quando $M^p \geq B_R$ cioè quando $p \geq \log_M B_R$. Poiché ad ogni passo ogni blocco deve essere letto e riscritto il costo dell'ordinamento è $2*B_R\log_M B_R$.

Join con l'uso di un indice

Supponiamo di nuovo di voler calcolare il join naturale di R=AB ed S=BC, e assumiamo che:

- $dom_R(B) \subseteq dom_S(B)$
- S abbia un indice clustering su B
- R sia memorizzata in forma compatta.

In tal caso per ogni tupla di R si può utilizzare l'indice per trovare tutte le tuple di S che hanno lo stesso valore per l'attributo B della tupla di R presa in considerazione. Con le assunzioni fatte, il numero di blocchi contenenti tuple di S con un dato valore di B è dato da $B_S / I_{S.B}$.

Pertanto, poiché:

- ogni blocco di R viene letto una volta
- per ogni tupla di R devono essere letti un numero di blocchi di S pari a: $B_S/I_{S.B}$

il costo per leggere l'input è dato da

$$B_R + T_R * B_S/I_{S.B}$$

Con considerazioni analoghe si può stimare il costo per la lettura nei seguenti casi:

Tipo organizzazione operandi	Stima costo lettura
R non è memorizzata in forma compatta S ha un indice clustering su B	$T_R + T_R \ B_S/I_{S.B}$
R è memorizzata in forma compatta S ha un indice nonclustering su B	$B_R + T_R \ T_S/I_{S.B}$
R non è memorizzata in forma compatta S ha un indice nonclustering su B	$T_R + T_R \ T_S/I_{S.B}$

Join con l'uso di 2 indici

Supponiamo di nuovo di voler calcolare il join naturale di R=AB ed S=BC, e assumiamo che:

- $dom_R(B) \subseteq dom_S(B)$
- sia R che S abbiano entrambe un indice clustering su B.

In tal caso, per ogni valore di B nell'indice di R, si prendono tutte le tuple di R che contengono quel valore (contenute in $B_R / I_{R.B}$ blocchi) e le si combinano con tutte le tuple di S con lo stesso valore per B (contenute in $B_S / I_{S.B}$ blocchi). Pertanto, per ogni valore di B in $dom_R(B)$ devono essere letti $\max(1, B_R/I_{R.B})$ blocchi di R e $\max(1, B_S/I_{S.B})$ blocchi di S e, quindi il costo stimato per la lettura in questa situazione è dato da:

$$I_{R.B} * (B_R/I_{R.B} + B_S/I_{S.B})$$

Costruzione di un indice clustering

Poiché può essere conveniente effettuare un join utilizzando un indice clustering sull'attributo comune, ha senso porsi il problema di costruire un tale indice qualora non esista. Vediamo, pertanto, quanto costa creare un indice clustering.

Supponiamo di voler creare un indice clustering sull'attributo B per la relazione R. Possiamo utilizzare a tale scopo una funzione hash che partiziona R in I_B bucket numerati da 0 a I_B-1, in base al valore di B. Se abbiamo M blocchi disponibili in memoria principale e $I_B > M$, cercare di creare gli I_B bucket in una sola volta potrebbe richiedere un numero eccessivo di accessi (potrebbe essere necessario fare due accessi per ogni tupla da inserire: infatti, se nessuno degli M blocchi che si trovano in quel momento in memoria principale è un blocco del bucket in cui deve essere inserita la tupla, occorre scrivere uno degli M blocchi in memoria secondaria in modo da creare spazio sufficiente per leggere l'ultimo blocco del bucket in cui deve essere inserita la tupla). Possiamo invece procedere iterativamente nel modo seguente:

Ad ogni iterazione ciascuno degli M blocchi è associato ad un "*superbucket*".

Inizialmente ciascun superbucket rappresenta I_B/M bucket: l'i-esimo (i=0, … ,M-1) superbucket rappresenta i bucket da $(i*I_B/M)$ a $((i+1)*I_B/M) - 1$.

Ogni blocco di R viene portato in memoria principale e ciascuna delle tuple in esso contenute è assegnata al superbucket che rappresenta il bucket in cui la tupla deve essere inserita. Quando un blocco associato a un superbucket è pieno lo si trasferisce in memoria secondaria e si inizia un nuovo blocco per lo stesso superbucket.

A ciascuna iterazione successiva, il procedimento viene applicato a ciascun superbucket creato nella iterazione precedente. Pertanto alla p-esima, p>1, iterazione si passa da superbucket che rappresentano I_B/M^{p-1} bucket, a superbucket che rappresentano I_B/M^p bucket.
Il procedimento termina quando ciascun superbucket rappresenta effettivamente un solo bucket; ciò accade quando $I_B/M^p \leq 1$, cioè dopo $\lceil \log_M I_B \rceil$ passi.

Poiché ad ogni iterazione B_R blocchi devono essere letti e scritti il costo stimato per la creazione di un indice clustering sull'attributo B per la relazione R è dato da $2B_R \log_M I_B$.

Chiariamo questo concetto con un esempio.
Supponiamo che:

- $dom_R(B) = \{0,1,2,3,4,5\}$, e quindi $I_{R.B}=6$
- Ogni blocco può contenere 3 tuple di R
- Ho a disposizione M=2 blocchi in memoria principale per creare l'indice e un altro blocco temporaneo per leggere le tuple di R
- Il sistema utilizza una semplicissima funzione hash che associa un valore intero di $dom_R(B)$ al bucket che ha per indice esattamente quel valore (h(x) = x)
- La relazione R, nei campi dell'attributo B è costituita dai seguenti 8 blocchi ($B_R=8$):

1	0	3	2	0	1	4	2
3	1	4	3	5	4	2	5
5	3	0	4	0	5	1	3

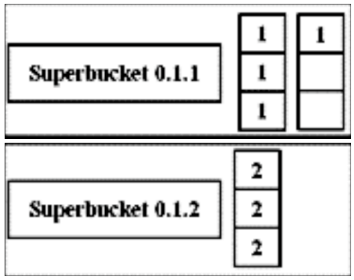
Poiché M=2, ho due blocchi di memoria principale da associare ciascuno ad un superbucket:
Il primo superbucket (i=0) è destinato a contenere i valori da 0 a 2 (che andrebbero nei relativi bucket da 0 a 2, come indicato dalla funzione hash), infatti rappresenta i bucket da $(0*I_B/M)=0$ a $((0+1)*I_B/M) - 1 = 2$.
Il secondo superbucket (i=1) è destinato a contenere i valori da 3 a 5 (che andrebbero nei relativi bucket da 3 a 5, come indicato dalla funzione hash), infatti rappresenta i bucket da $(1*I_B/M)=3$ a $((1+1)*I_B/M) - 1 = 5$.
Leggo ad uno ad uno i blocchi di R in memoria principale, e divido tra i due superbucket le sue tuple (in base al valore contenuto nel campo B), e scrivendo i blocchi completati dei superbucket in memoria secondaria fino ad ottenere la situazione seguente:

Superbucket 0	1	0	0	1	
	0	2	1	2	
	1	0	2		
Superbucket 1	3	3	4	5	3
	5	4	5	4	
	3	3	4	5	

Iteriamo il procedimento sul superbucket 0 ottenendo i due superbucket 0.0 e 0.1:

Superbucket 0.0	0	0	
	0		
	0		
Superbucket 0.1	1	1	2
	1	2	
	2	1	

Il superbucket 0.0 rappresenta ora il bucket 0
Iteriamo il procedimento sul superbucket 0.1 ottenendo i due superbucket 0.1.1 e 0.1.2:



Il superbucket 0.1.1 rappresenta ora il bucket 1 mentre il superbucket 0.1.2 rappresenta il bucket 2
Analogo procedimento viene effettuato sul suprebucket 1 e sui suoi discendenti. completando lo svolgimento dell'algoritmo, abbiamo costruito un indice clustering dividendo la nostra relazione R in bucket in $\lceil \log_M * I_B \rceil = \lceil \log_2(6) \rceil = 3$ iterazioni, pagando un costo in numero di accessi di $2*8\lceil \log_M * I_B \rceil = 2*8\lceil \log_2(6) \rceil = 2*8*3 = 48$ accessi.

Ottimizzazione di interrogazioni in QUEL

Come detto nell'introduzione, in genere i sistemi utilizzano una combinazione delle due tecniche di ottimizzazione delle interrogazioni (quella che sfrutta proprietà degli operatori dell'algebra relazionale e quella che sceglie fra i diversi modi in cui è possibile implementare un operatore dell'algebra relazionale quello con costo stimato inferiore) viste nei paragrafi precedenti. Illustreremo ciò considerando quanto viene fatto per ottimizzare un'interrogazione espressa nel linguaggio QUEL, utilizzato dal sistema INGRES.
Come vedremo, l'algoritmo oltre a fare in modo che le selezioni vengano eseguite prime possibile, utilizza:

- un metodo per calcolare particolari join di 3 o più relazioni
- un'euristica per ordinare join e prodotti cartesiani
- conversione di un join in un semijoin seguito da un join.
  (dove il Semijoin ($R \triangleright S$) è definito come: $\Pi_R(R \triangleright\triangleleft S)$)

Join di tre o più relazioni.
Supponiamo di voler calcolare il join naturale $Q \triangleright\triangleleft R \triangleright\triangleleft S$, con Q=AB, R=BC e S=CD;
Poiché il join naturale gode della proprietà associativa e di quella commutativa potremmo calcolare ad esempio:

$(Q \triangleright\triangleleft R) \triangleright\triangleleft S$ oppure

$Q \triangleright\triangleleft (R \triangleright\triangleleft S)$ oppure

$(Q \triangleright\triangleleft S) \triangleright\triangleleft R$ (questa sarebbe la scelta peggiore in quanto $Q \triangleright\triangleleft S$ è di fatto un prodotto cartesiano),

Tuttavia, come vedremo, esiste un modo alternativo per calcolare $Q \triangleright\triangleleft R \triangleright\triangleleft S$, che non consiste nel calcolo di due join binari successivi.

Per rendere più semplice il confronto tra i costi dei due metodi facciamo le seguenti assunzioni:

- Q ha un indice clustering sull'attributo comune B
- S ha un indice clustering sull'attributo comune C
- ciascuna delle relazioni Q, R ed S ha T_0 tuple (cioè $T_0=T_Q=T_R=T_S$)
- ciascuna delle relazioni Q, R ed S può essere memorizzata in forma compatta in B_0 blocchi (cioè $B_0=B_Q=B_R=B_S$)
- ciascuno degli attributi A, B, C e D ha image size I_0 in ciascuna relazione (cioè $I_0=I_{Q.A}=I_{Q.B}=I_{R.B}=I_{R.C}=I_{S.C}=I_{S.D}$)
- $I_0 \leq B_0$

Cominciamo con lo stimare il costo di $Q \triangleright\triangleleft R$.
Con le ipotesi fatte una stima del costo è data dalla somma di:

i. $B_0 + T_0 * \left\lceil \frac{B_0}{I_0} \right\rceil$ (costo per la lettura con un indice clustering) e di

ii. $\frac{B_0 T_0 + B_0 T_0}{I_0}$ (costo per la scrittura)

cioè da $B_0 + \left\lceil \frac{3 B_0 T_0}{I_0} \right\rceil$.
Notiamo che la relazione $Q \triangleright\triangleleft R$:

- ha $T_{Q \triangleright\triangleleft R} = T_0 * (T_0 / I_0) = T_0^2 / I_0$ tuple. Infatti per ognuna delle $T_Q=T_0$ tuple di Q ci sono $T_R/I_{R.B}=T_0/I_0$ tuple di R che hanno lo stesso valore nell'attributo comune B
- può essere memorizzata in forma compatta in $B_{Q \triangleright\triangleleft R} = 2\, T_0 B_0 / I_0$ blocchi.

Possiamo a questo punto stimare il costo di (Q⊳⊲R)⊳⊲S come la somma di:

iii. $B_{Q\rhd\lhd R} + T_{Q\rhd\lhd R} * \lceil \frac{B_0}{I_0} \rceil$ (costo per la lettura con un indice clustering) e di

iv. $\frac{B_0 T_{Q\rhd\lhd R} + B_{Q\rhd\lhd R} T_0}{I_0}$ (costo per la scrittura)

v. $B_0 + \lceil \frac{3B_0 T_0}{I_0} \rceil$ (costo per calcolare Q⊳⊲R)

cioè da

$$\frac{2T_0B_0}{I_0} + \frac{T_0^2}{I_0} * \frac{B_0}{I_0} + \frac{B_0\frac{T_0^2}{I_0} + \frac{T_0T_0 2B_0}{I_0}}{I_0} + B_0 + \lceil \frac{3B_0T_0}{I_0} \rceil = B_0 + 5\frac{B_0T_0}{I_0} + 4\frac{B_0T_0^2}{I_0^2} \qquad (8)$$

Si osservi che in tale costo è compreso il costo ($2B_0T_0/I_0$) per scrivere Q⊳⊲R e il costo (ancora $2B_0T_0/I_0$) per leggere Q⊳⊲R. Se i due join vengono calcolati in modo interleaved (cioè le tuple generate per il primo join vengono immediatamente utilizzate per generare tuple del secondo join) è possibile sottrarre $4(B_0T_0/I_0)$ dalla (8) ottenendo:

$$B_0 + \frac{B_0T_0}{I_0} + 4\frac{B_0T_0^2}{I_0^2} \qquad (9)$$

Metodo alternativo

Un modo alternativo per calcolare Q⊳⊲R⊳⊲S è il seguente:

Prendiamo una tupla "bc" ∈ R, con b istanza di B (attributo comune tra Q e R) e c istanza di C (attributo comune tra R ed S)

for each tupla bc di R **do**
genera tutte le tuple di $\sigma_{B=b}(Q)\rhd\lhd bc\rhd\lhd\sigma_{C=c}(S)$

Usando l'indice clustering possiamo calcolare $\sigma_{B=b}(Q)$ con B_0/I_0 accessi (per la lettura); analogamente per $\sigma_{C=c}(S)$. Entrambe le relazioni avranno T_0/I_0 tuple. Se il numero M di blocchi disponibili in memoria principale è maggiore o uguale a $2B_0/I_0$, $\sigma_{B=b}(Q)$ e $\sigma_{C=c}(S)$ possono essere mantenute contemporaneamente in memoria principale (e quindi non sono richiesti ulteriori accessi per la lettura). Il numero di tuple generate dal **corpo** del ciclo *for* è dato da T_0^2/I_0^2; tali tuple occupano $3B_0T_0/I_0^2$ blocchi. Pertanto il costo dell'input (lettura) per elaborare il corpo del ciclo è $2B_0/I_0$ e quello dell'output (scrittura) è $3B_0T_0/I_0^2$. Poiché il ciclo viene iterato T_0 volte (numero delle tuple di R), e devo leggere tutto R (B_0 accessi), il costo complessivo del metodo è dato da:

$$B_0 + T_0\lceil 2\frac{B_0}{I_0} + 3\frac{B_0T_0}{I_0^2}\rceil = B_0 + 2\frac{T_0B_0}{I_0} + 3\frac{B_0T_0^2}{I_0^2} \qquad (10)$$

Se $T_0=I_0$ ($T_0/I_0=1$), i costi dei due metodi (espressi dalla (9) e dalla (10)) coincidono. Se invece $T_0>>I_0$, il rapporto tra (9) e (10) è prossimo a 4/3.

Il metodo appena visto per calcolare $Q \triangleright\triangleleft R \triangleright\triangleleft S$ può essere generalizzato. Infatti, se si deve calcolare $R \triangleright\triangleleft S_1 \triangleright\triangleleft S_2 \ldots \triangleright\triangleleft S_n$ e

- $R \cap S_i \neq \emptyset$, i=1,…,n
- $S_i \cap S_j = \emptyset$, $i \neq j$

si può procedere nel modo seguente:

for each tupla $\square$ di R **do**
begin
    for i:=1 **to** n **do**
        $T_i := S_i \triangleright \{\square\}$;
    produci $\{\square\} \triangleright\triangleleft T_1 \triangleright\triangleleft T_2 \triangleright\triangleleft \ldots \triangleright\triangleleft T_n$
end

L'ALGORITMO DI OTTIMIZZAZIONE PER QUEL.

L'algoritmo prende in input un'interrogazione espressa in QUEL e ne genera una rappresentazione interna costituita da un ipergrafo. Incominciamo quindi a vedere la forma che assume una tipica interrogazione in QUEL e a introdurre il concetto di ipergrafo.

Interrogazioni in QUEL
Una classica query in SQL ha la forma:

SELECT $A_1, A_2, .. , A_m$
FROM $R_1, R_2, .. , R_n$
WHERE C

E viene espressa in algebra relazionale come:

$\Pi_{A1, A2, .. , Am}(\sigma_C (R_1 \times R_2 \times .. \times R_n))$

In una interrogazione espressa in QUEL all'inizio tramite la clausola *range of* (utilizzo analogo all'*alias* SQL) si dichiarano le variabili di tupla che compaiono nelle successive clausole *retrieve* e *where*, specificando per ciascuna la relazione in cui può variare (ovviamente ci possono essere più variabili di tupla che variano sulla stessa relazione e, quindi gli R_i non sono necessariamente distinti) :

range of t_1 *is* R_1
range of t_2 *is* R_2
..
range of t_n *is* R_n

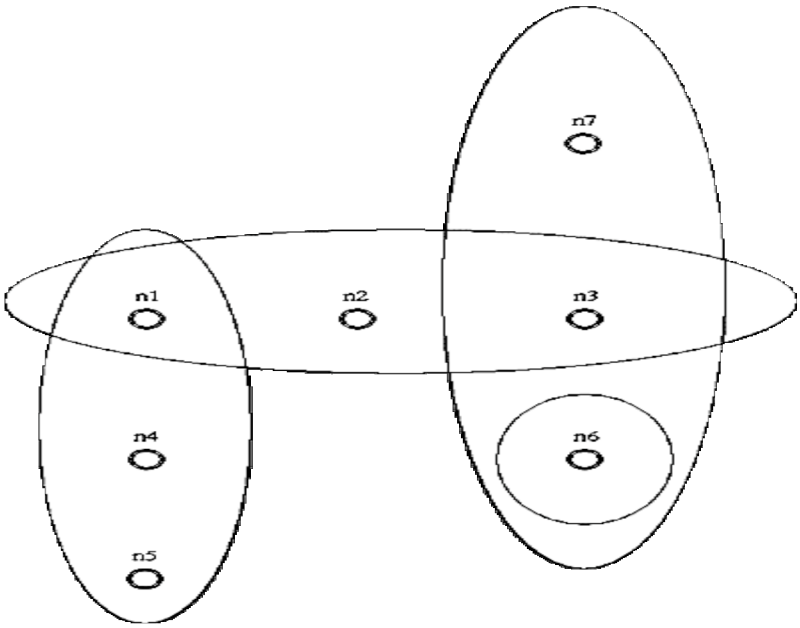
quindi tramite la clausola *retrieve* si specificano gli attributi che interessano (utilizzo analogo alla *select* SQL):

retrieve $t_{i1}.A_1, t_{i2}.A_2, .. , t_{im}.A_m$

e, tramite la clausola *where* (che ha lo stesso scopo che in SQL), si specificano le condizioni che devono essere soddisfatte dai valori degli attributi nelle variabili di tupla $t_1, t_2,\ldots, t_n$ *(es. where* $t_i.A=100$*).*

Il concetto di ipergrafo
Un *ipergrafo* $\mathcal{G}$ è una coppia ordinata (N,E), dove:
- N è un insieme i cui elementi sono detti *nodi* dell'ipergrafo
- E è una famiglia di sottoinsiemi non vuoti di N detti *edge* (o *archi*) dell'ipergrafo.

Un ipergrafo $\mathcal{G}$ è *connesso* se ogni coppia di nodi è connessa (due nodi sono *connessi* se esiste un edge che li contiene entrambi o esiste un terzo nodo connesso ad entrambi).
Nella figura seguente e' data la rappresentazione grafica dell'ipergrafo connesso $\mathcal{G}$ = (N,E), dove
- $N=\{n_1,n_2,n_3,n_4,n_5,n_6,n_7\}$ ed
- $E=\{\{n_1,n_2,n_3\},\{n_1,n_4,n_5\},\{n_3,n_6,n_7\},\{n_6\}\}$.



Analizziamo ora i passi dell'algoritmo.

1° PASSO: Costruzione dell'ipergrafo associato all'interrogazione.
Data una interrogazione del tipo $\Pi_F(R_1 \times R_2 \times \ldots \times R_n)$, in cui le $R_1,R_2,\ldots,R_n$ non sono necessariamente distinte e la condizione $F = F_1 \wedge F_2 \wedge \ldots \wedge F_m$ è in forma normale congiuntiva (FNC), l'algoritmo di ottimizzazione associa ad essa un ipergrafo (*ipergrafo di connessione*) costruito nel modo seguente:

- *Nodi*. Per ogni attributo A in R_i viene creato un nodo $R_i.A$. Se R_i e R_j sono la stessa relazione vengono creati comunque due nodi per ogni attributo della relazione. Se una condizione F_k è della forma $R_i.A=R_j.B$ allora i nodi $R_i.A$ e $R_j.B$ vengono identificati (questo processo di identificazione viene fatto in modo transitivo, cioè: se $R_i.A=R_j.B$ e $R_j.B=R_l.C$ viene creato un unico nodo che rappresenta $R_i.A$, $R_j.B$ e $R_l.C$.
- *Edge*. Abbiamo due categorie di edge. Per ogni R_j viene creato un edge che contiene tutti e soli i nodi che rappresentano gli attributi di R_j (*edge-relazione*, rappresentati da una linea continua). Per ogni condizione F_k che non è della forma $R_i.A=R_j.B$ o della forma $R_i.A=a$ viene creato un edge che contiene tutti e soli i nodi che rappresentano gli attributi che compaiono in F_k (*edge-condizione*, rappresentati da una linea tratteggiata).

L'algoritmo associa ad ogni edge-relazione E una variabile R(E), che cambierà il proprio contenuto durante l'esecuzione, e genera un programma che inizializza tali variabili.

2° PASSO: Inizializzazione delle variabili associate agli edge-relazione.
Per ogni edge-relazione E, l'assegnazione che inizializza la variabile R(E) è:

- $R(E):= R_0(E)$, dove $R_0(E)$ è **la relazione** che ha dato origine all'edge-relazione E, se non esiste nessuna condizione del tipo A='a', con A attributo di $R_0(E)$,
- $R(E):= \Pi_F(R_0(E))$, dove F è la congiunzione (AND) di tutte le condizioni del tipo A='a' con A attributo di $R_0(E)$, altrimenti.

L'algoritmo riduce quindi l'ipergrafo di connessione eliminando edge.

3° PASSO: Riduzione dell'ipergrafo associato all'interrogazione.
Ad ogni passo di riduzione se:

$\mathcal{G}$ è l'ipergrafo che viene ridotto e

$\mathcal{H}$ è l'ipergrafo risultante dalla riduzione

viene generato un programma, Prog($\mathcal{G}$), che consente di valutare l'interrogazione a cui è associato $\mathcal{G}$ una volta che sia stata valutata l'interrogazione a cui è associato $\mathcal{H}$. Nel programma generato la variabile

destinata a contenere il risultato dell'interrogazione a cui è associato l'ipergrafo $\mathcal{G}$ è Result($\mathcal{G}$). Le regole per generare Prog($\mathcal{G}$) sono le seuenti:

- **(caso 0).** Se $\mathcal{G}$ consiste di un singolo edge-relazione E (come vedremo la scelta dell'edge da eliminare è tale che un ipergrafo non può mai consistere di un singolo edge-condizione) allora Prog($\mathcal{G}$) è costituito da:

  Result($\mathcal{G}$) := R(E);

- (**caso 1**). Se $\mathcal{G}$ è non connesso e $\mathcal{H}_1$, $\mathcal{H}_2$,…, $\mathcal{H}_k$ sono le componenti connesse di $\mathcal{G}$ allora Prog($\mathcal{G}$) è costituito da:

  Prog($\mathcal{H}_1$); Prog($\mathcal{H}_2$);… Prog($\mathcal{H}_k$);
  Result($\mathcal{G}$) := Result($\mathcal{H}_1$) $\cup$ Result($\mathcal{H}_2$) $\cup$…$\cup$ Result($\mathcal{H}_k$);

- (**caso 2**) Se $\mathcal{H}$ è l'ipergrafo risultante dalla eliminazione dell'edge-condizione E da $\mathcal{G}$ allora Prog($\mathcal{G}$) è costituito da:

  Prog($\mathcal{H}$);
  Result($\mathcal{G}$) := $\sigma_{C(E)}$ (Result($\mathcal{H}$));

  dove C(E) è la condizione che ha originato l'edge E

- **(caso 3).** Se $\mathcal{H}$ è l'ipergrafo risultante dalla eliminazione dell'edge-relazione E da $\mathcal{G}$ e $\mathcal{H}_1$, $\mathcal{H}_2$,…, $\mathcal{H}_k$ sono le componenti connesse di $\mathcal{H}$ allora Prog($\mathcal{G}$) è costituito da:

  for each edge-relazione F such that F$\cap$ E $\neq \emptyset$
  do R(F) := R(F) $\triangleright$ R(E);
  Prog($\mathcal{H}_1$); Prog($\mathcal{H}_2$);… Prog($\mathcal{H}_k$);
  Result($\mathcal{G}$) := R(E) $\triangleright\triangleleft$ Result($\mathcal{H}_1$) $\triangleright\triangleleft$ Result($\mathcal{H}_2$) $\triangleright\triangleleft$…$\triangleright\triangleleft$ Result($\mathcal{H}_k$);

Vediamo ora l'applicazione di questo algoritmo parziale su di un esempio:

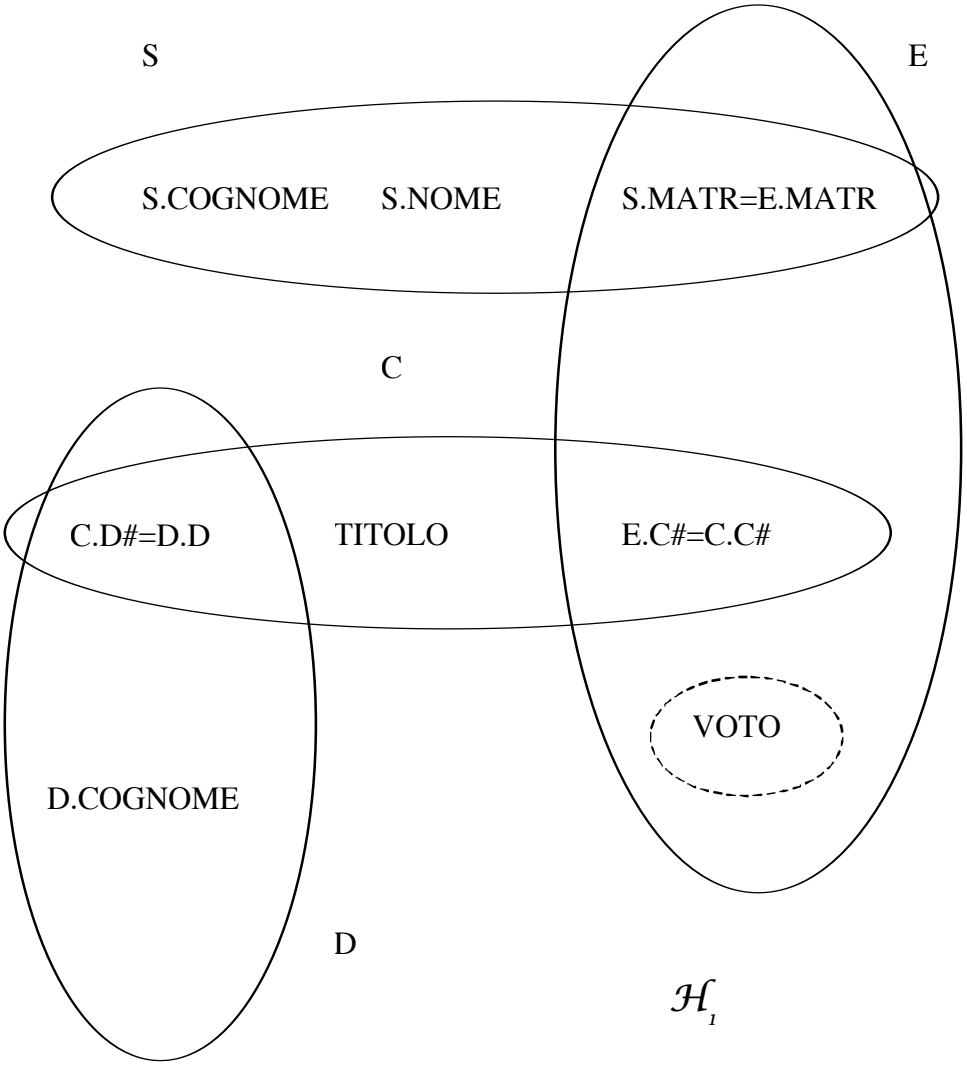
Supponiamo di avere i seguenti schemi di relazione:

STUDENTE	*(breviter S)*	*(NOME, COGNOME, MATR)*
ESAME	*(breviter E)*	*(MATR, C#, VOTO)*
CORSO	*(breviter C)*	*(C#, TITOLO, DOC#)*
DOCENTE	*(breviter D)*	*(DOC#, COGNOME)*

e di voler sapere i **Cognomi** degli **Studenti** che hanno ottenuto più di **27** in un **esame** relativo ad un **corso** tenuto dal Professor **Rossi**. Tale interrogazione può essere formulata in algebra relazionale (trascurando per ora l'operatore di proiezione) nel modo seguente:

σ_F (STUDENTE $\cup$ ESAME $\cup$ CORSO $\cup$ DOCENTE)
dove F è S.MATR=E.MATR $\cap$ E.C#=C.C# $\cap$ C.DOC#=D.DOC# $\cap$ VOTO>27 $\cap$ D.COGNOME='Rossi'

Applicazione del 1° PASSO: Costruzione dell'ipergrafo $\mathcal{H}1$:

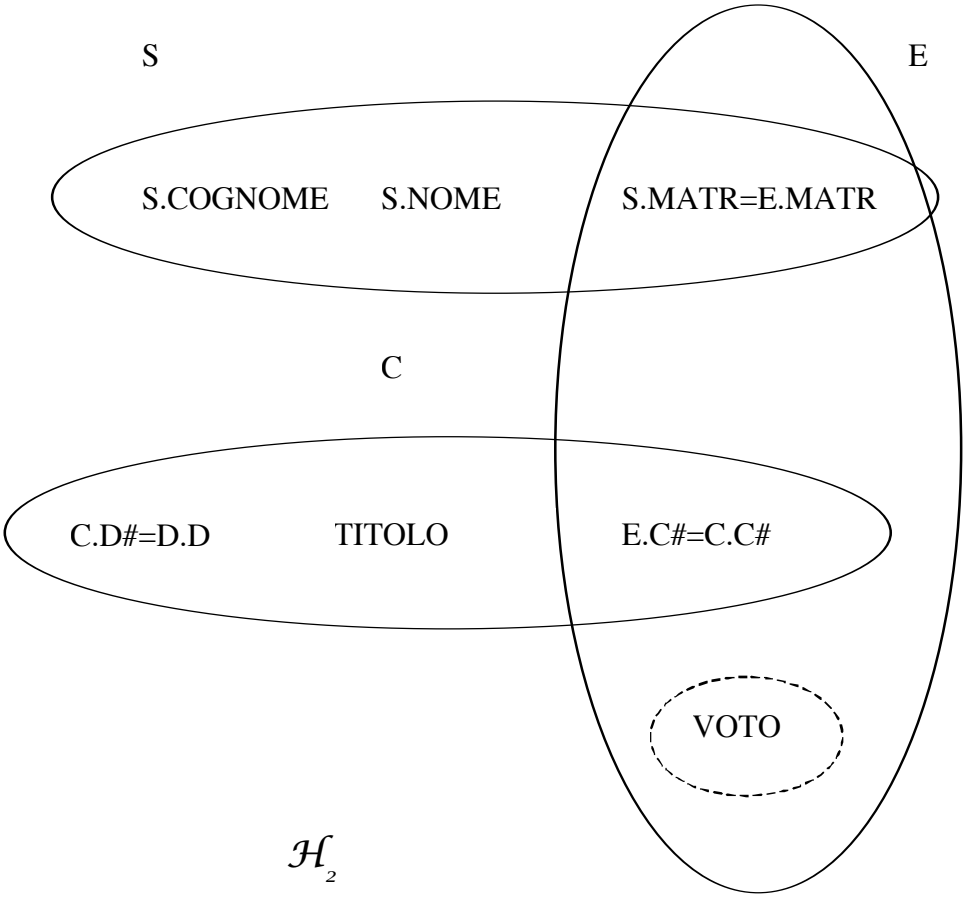


NB: la condizione D.COGNOME='Rossi' viene ignorata perché del tipo A='a'

Applicazione del 2° PASSO Inizializzazione delle variabili:

R(S) := STUDENTE
R(E) := ESAME
R(C) := CORSO
R(D) := $\sigma_{D.COGNOME='Rossi'}$ (DOCENTE)

Applicazione del 3° PASSO: Riduzione dell'ipergrafo $\mathcal{H}1$ e generazione di Prog($\mathcal{H}1$)
Eliminiamo innanzitutto l'edge-relazione D (vedremo poi il criterio per eliminare gli edge) ed otteniamo il seguente ipergrafo connesso $\mathcal{H}2$:



Avendo eliminato un edge-relazione sono nel (caso 3). L'unica relazione che ha attributi in comune con D è C, viene quindi generata l'assegnazione:

R(C) := R(C) ▷ R(D);

(che permette di ridurre R(C) prendendo solo i corsi tenuti da Rossi). Quindi Prog($\mathcal{H}1$) e' costituito dalla seguente sequenza di istruzioni:

R(C) := R(C) ▷ R(D);

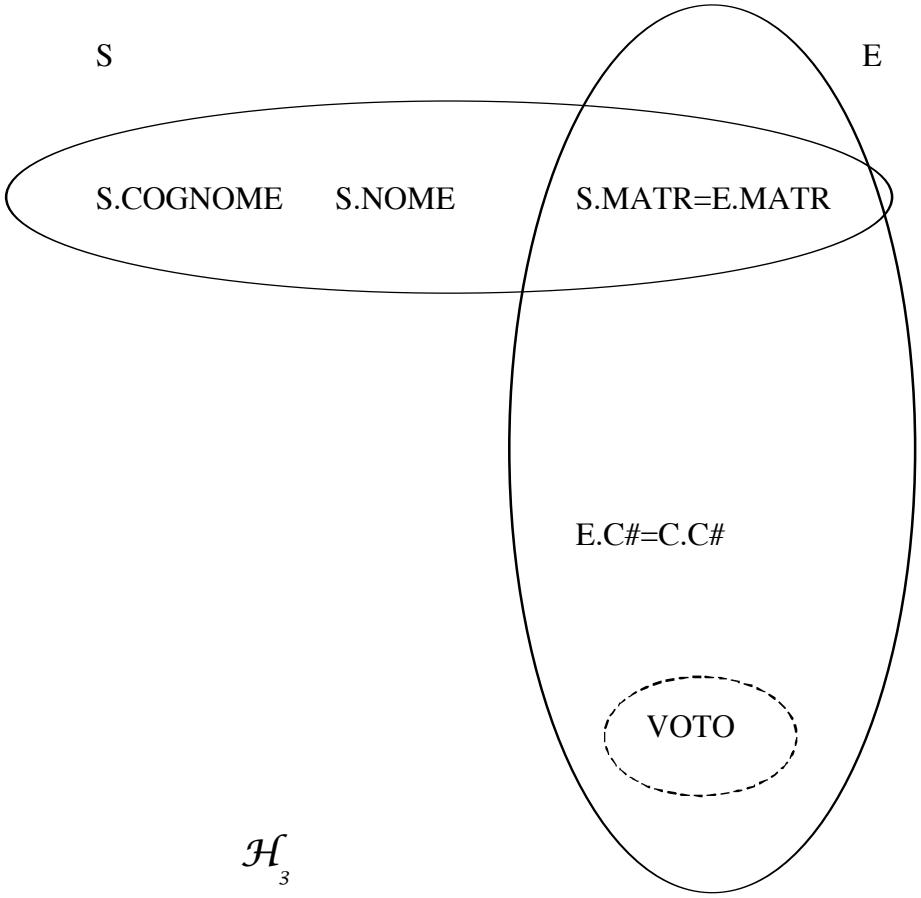
Prog($\mathcal{H}2$);

Result($\mathcal{H}1$) := R(D) ▷◁ Result($\mathcal{H}2$);

Il codice fin qui prodotto è quindi il seguente:

R(S) := STUDENTE
R(E) := ESAME
R(C) := CORSO
R(D) := $\sigma_{D.COGNOME='Rossi'}$ (DOCENTE)
R(C) := R(C) ▷ R(D);
Prog($\mathcal{H}_2$);
Result($\mathcal{H}_1$) := R(D) ▷◁ Result($\mathcal{H}_2$);

Sviluppiamo Prog($\mathcal{H}_2$).
Eliminiando l'edge-relazione C si ottiene il seguente ipergrafo connesso $\mathcal{H}_3$:



$\mathcal{H}_3$

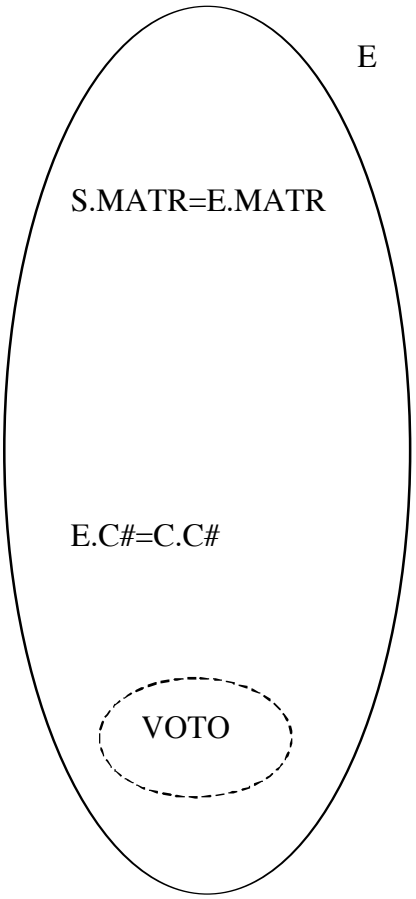
Calcoliamo Prog($\mathcal{H}_2$) a partire da Prog($\mathcal{H}_3$)
Siamo ancora nel (caso3). L'unica relazione che ha attributi in comune con C è E, Quindi Prog($\mathcal{H}_2$) e' costituito dalla seguente sequenza di istruzioni:

R(E) := R(E) ▷ R(C);
Prog($\mathcal{H}_3$);
Result($\mathcal{H}_2$) := R(C) ▷◁ Result($\mathcal{H}_3$);

Il codice fin qui prodotto è quindi il seguente:

R(S) := STUDENTE
R(E) := ESAME
R(C) := CORSO
R(D) := $\sigma_{D.COGNOME='Rossi'}$ (DOCENTE)
R(C) := R(C) ▷ R(D);
R(E) := R(E) ▷ R(C);
Prog($\mathcal{H}_3$);
Result($\mathcal{H}_2$) := R(C) ▷◁ Result($\mathcal{H}_3$);
Result($\mathcal{H}_1$) := R(D) ▷◁ Result($\mathcal{H}_2$)

Sviluppiamo Prog($\mathcal{H}_3$).
Eliminando l'edge-relazione S si ottiene il seguente ipergrafo connesso $\mathcal{H}_4$:



$\mathcal{H}_4$

Calcoliamo Prog($\mathcal{H}_3$) a partire da Prog($\mathcal{H}_4$)

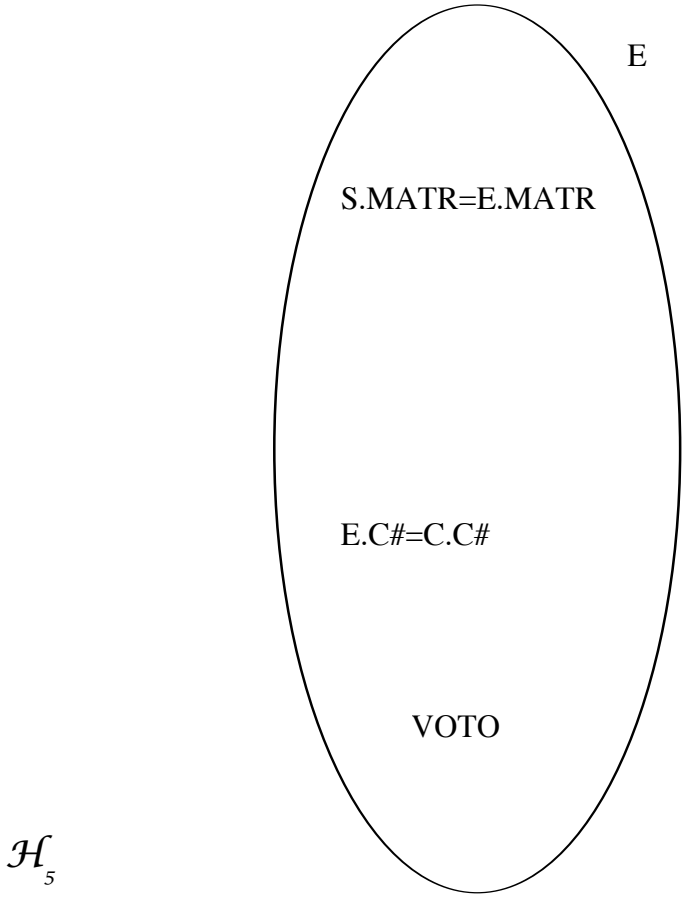
Siamo ancora nel (caso3):

R(E) := R(E) ▷ R(S);
Prog($\mathcal{H}_4$);
Result($\mathcal{H}_3$) := R(S) ▷◁ Result($\mathcal{H}_4$);

Il codice fin qui prodotto è quindi il seguente:

R(S) := STUDENTE
R(E) := ESAME
R(C) := CORSO
R(D) := $\square_{D.COGNOME='Rossi'}$ (DOCENTE)
R(C) := R(C) ▷ R(D);
R(E) := R(E) ▷ R(C);
R(E) := R(E) ▷ R(S);
Prog($\mathcal{H}_4$);
Result($\mathcal{H}_3$) := R(S) ▷◁ Result($\mathcal{H}_4$);
Result($\mathcal{H}_2$) := R(C) ▷◁ Result($\mathcal{H}_3$);
Result($\mathcal{H}_1$) := R(D) ▷◁ Result($\mathcal{H}_2$);

Sviluppiamo Prog($\mathcal{H}_4$).
Eliminiando l'edge-condizione su VOTO si ottiene il seguente ipergrafo connesso $\mathcal{H}_5$:



Calcoliamo Prog($\mathcal{H}4$)
Stavolta siamo nel (caso 2) e produciamo il seguente codice:
Prog($\mathcal{H}5$);
Result($\mathcal{H}4$) := $\sigma_{VOTO>27}$(Result($\mathcal{H}5$));

Poiché siamo nel (caso 0) Prog($\mathcal{H}5$) è costituito da:
Result($\mathcal{H}5$) := R(E);

Otteniamo quindi il codice completo:

R(S) := STUDENTE
R(E) := ESAME
R(C) := CORSO
R(D) := $\square_{D.COGNOME='Rossi'}$ (DOCENTE)
R(C) := R(C) ▷ R(D);

R(E) := R(E) ▷ R(C);

R(E) := R(E) ▷ R(S);

Result($\mathcal{H}5$) := R(E);

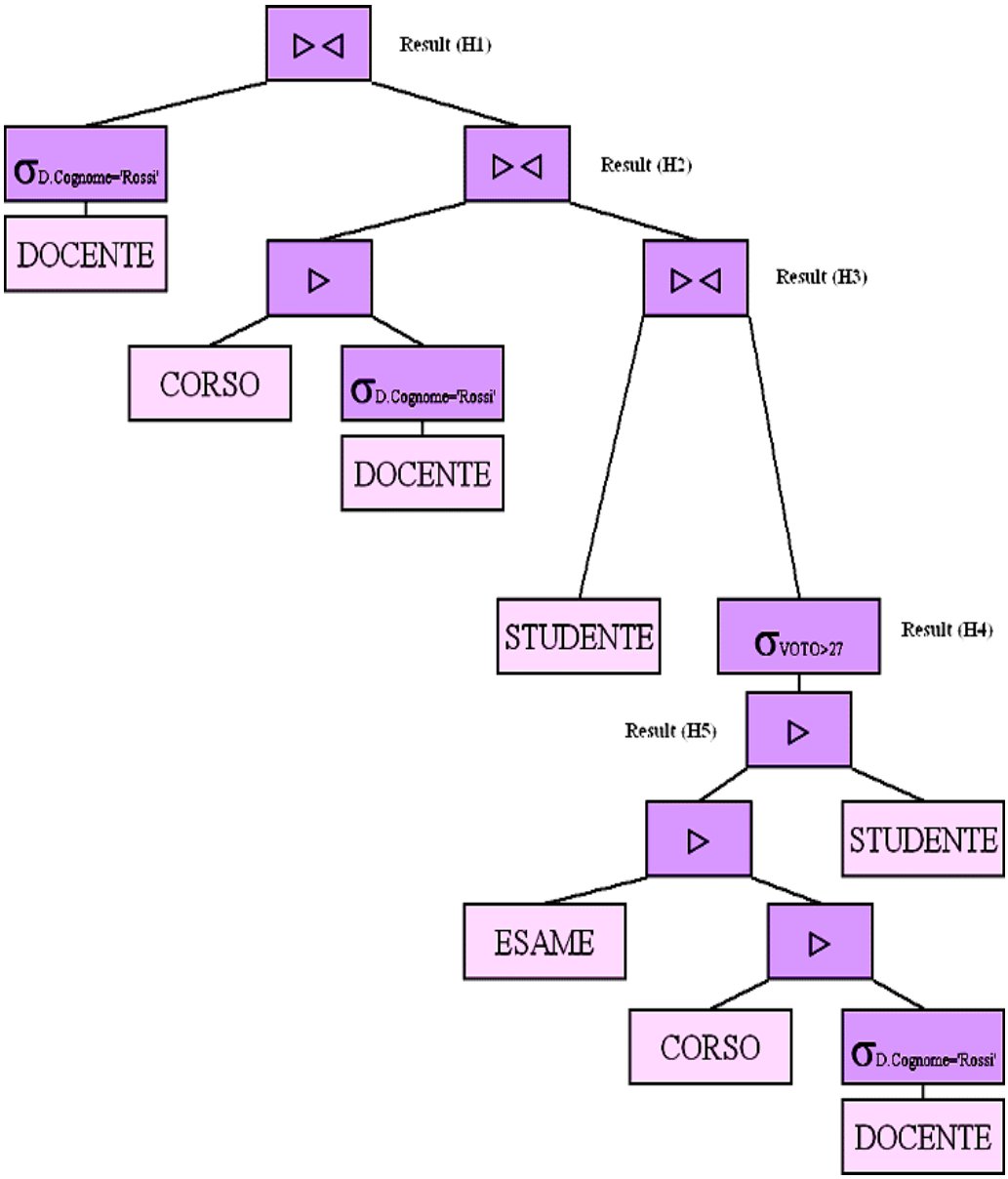
Result($\mathcal{H}4$) := $\square_{VOTO>27}$(Result($\mathcal{H}5$));

Result($\mathcal{H}3$) := R(S) ▷◁ Result($\mathcal{H}4$);

Result($\mathcal{H}2$) := R(C) ▷◁ Result($\mathcal{H}3$);

Result($\mathcal{H}1$) := R(D) ▷◁ Result($\mathcal{H}2$);

Le operazioni svolte possono essere rappresentate dal seguente albero:



Gestione delle proiezioni.
Come abbiamo detto una delle idee alla base dell'algoritmo è quella di fare le selezioni prima possibile per ridurre le dimensioni degli operandi di prodotti cartesiani e join. Questo fatto si riflette, come vedremo, nell'ordine in cui gli edge vengono eliminati. Lo stesso principio di ridurre le dimensioni degli operandi di un join ispira l'uso di semijoin prima dei join. Ovviamente c'è ancora un'altra operazione che permette di ridurre le dimensioni di una relazione: la proiezione.

Vediamo pertanto come vengono gestite le proiezioni dall'algoritmo.

La chiave per eliminare appena possibile attributi dalle relazioni conservando solo quelli che sono necessari per eseguire i successivi passi del programma (applicazione dell'operatore di proiezione) è data dal concetto di **nodo "distinto"**. Informalmente un nodo distinto è un nodo che rappresenta un attributo necessario per eseguire i successivi passi del programma (quelli più in alto nell'albero sintattico).

Formalmente:
Un nodo N di un ipergrafo $\mathcal{G}$ è *distinto* se si verifica almeno una delle seguenti condizioni:
- $\mathcal{G}$ è l'ipergrafo iniziale e N compare nella clausola *retrieve*
- $\mathcal{G}$ è l'ipergrafo risultante dall'eliminazione dell'edge E dall'ipergrafo $\mathcal{H}$ ed N è un nodo di $\mathcal{G}$ che
  - è in E, oppure
  - è distinto in $\mathcal{H}$
- $\mathcal{G}$ è una componente connessa dell'ipergrafo $\mathcal{H}$ ed N è distinto in $\mathcal{H}$.

Vediamo come sarebbe stato definito l'insieme dei nodi distinti nell'esempio appena esposto:

L'ipergrafo di partenza ($\mathcal{H}1$) ha come nodi distinti gli attributi che compaiono nella clausola *retrieve*, quindi:
$\mathcal{D}(\mathcal{H}1)$ = {S.COGNOME}

Eliminando l'edge D da $\mathcal{H}1$ si ottiene l'ipergrafo $\mathcal{H}2$ che ha come insieme di nodi distinti
$\mathcal{D}(\mathcal{H}2)$ = {D.D# ≡ C.D#} ∪ {S. COGNOME }

Eliminando l'edge C da $\mathcal{H}2$ si ottiene l'ipergrafo $\mathcal{H}3$ che ha come insieme di nodi distinti
$\mathcal{D}(\mathcal{H}3)$ = {C.C# ≡ E.C#} ∪ {S. COGNOME }

Eliminando l'edge S da $\mathcal{H}3$ si ottiene l'ipergrafo $\mathcal{H}4$ che ha come insieme di nodi distinti
$\mathcal{D}(\mathcal{H}4)$ = {S.MATR ≡ E.MATR} ∪ {C.C# ≡ E.C#}

Eliminando l'edge-condizione VOTO da $\mathcal{H}4$ si ottiene l'ipergrafo $\mathcal{H}5$ che ha come insieme di nodi distinti
$\mathcal{D}(\mathcal{H}5)$ = {VOTO} ∪ {C.C# ≡ E.C# , S.MATR ≡ E.MATR}
Comprendiamo intuitivamente perché conservarli nella computazione:

S. COGNOME serve per produrre l'output.
D.D# ≡ C.D# serve per effettuare il semijoin tra un sottoinsieme di DOCENTE e CORSO in Prog($\mathcal{H}1$)
E.C# ≡ C.C# serve per effettuare il semijoin tra ESAME e CORSO in Prog($\mathcal{H}2$)
S.MATR ≡ E.MATR serve per effettuare il semijoin tra ESAME e STUDENTE in Prog($\mathcal{H}3$)
VOTO serve ad effettuare la selezione in Prog($\mathcal{H}4$)

L'insieme di istruzioni prodotte nei casi 0-3 visti sopra deve essere quindi modificato per tener conto dei nodi distinti. A tale scopo si può fare uso di una procedura ricorsiva, Eval($\mathcal{G}$,$\mathcal{D}$), che ha come argomenti un ipergrafo $\mathcal{G}$ e l'insieme di nodi distinti $\mathcal{D}$ di $\mathcal{G}$ e produce la proiezione di ciò che avevamo indicato prima con Result($\mathcal{G}$) su $\mathcal{D}$.

Eval($\mathcal{G}$,$\mathcal{D}$) è così strutturata:

- nel **caso 0** ($\mathcal{G}$ composto da singolo edge-relazione E) la procedura esegue le seguenti istruzioni:

produci "Result($\mathcal{G}$):=$\pi_{\mathcal{D}}$(R(E))";

- nel **caso 1** ($\mathcal{G}$ non connesso) la procedura esegue le seguenti istruzioni:

for i:=1 **to** k **do**
begin
$\mathcal{D}_i$:=$\mathcal{D}$ $\cap$ Nodes($\mathcal{H}_i$);
Eval($\mathcal{H}_i$, $\mathcal{D}_i$)
end
produci "Result($\mathcal{G}$):=$\pi_{\mathcal{D}}$(Result($\mathcal{H}_1$) $\bowtie$ Result($\mathcal{H}_2$) $\bowtie$ … $\bowtie$ Result($\mathcal{H}_k$))";

- nel **caso 2** (Otteniamo $\mathcal{H}$ eliminndo un edge-condizione E da $\mathcal{G}$) la procedura esegue le seguenti istruzioni:

$\underline{\mathcal{D}}$:= ($\mathcal{D}$ $\cap$ E) $\cap$ Nodes($\mathcal{H}$);
Eval($\mathcal{H}$, $\underline{\mathcal{D}}$)
produci "Result($\mathcal{G}$):= $\pi_{\mathcal{D}}$($\sigma_{C(E)}$(Result ($\mathcal{H}$)))";

- nel **caso 3** (Otteniamo $\mathcal{H}1..\mathcal{H}k$ componenti connesse eliminando un edge-relazione E da $\mathcal{G}$) la procedura esegue le istruzioni:

for each edge-relazione F such that F$\cap$ E $\neq \emptyset$
do produci "R(F):=R(F) $\triangleright$ R(E)";
for i:=1 **to** k **do**
begin
$\mathcal{D}_i$:= ($\mathcal{D}$ $\cap$ E) $\cap$ Nodes($\mathcal{H}_i$);
Eval($\mathcal{H}_i$, $\mathcal{D}_i$)
End
produci "Result($\mathcal{G}$):=$\pi_{\mathcal{D}}$(R(E) $\triangleright\triangleleft$ Result($\mathcal{H}_1$) $\triangleright\triangleleft$ Result($\mathcal{H}_2$) $\triangleright\triangleleft$ … $\triangleright\triangleleft$ Result($\mathcal{H}_k$))";

L'algoritmo di ottimizzazione di QUEL con GESTIONE DELLE PROIEZIONI

L'algoritmo completo consiste di due fasi:

- Fase I. **Inizializzazione**
  - viene costruito l'ipergrafo $\mathcal{G}$ di connessione associato all'interrogazione
  - per ogni edge E di $\mathcal{G}$ viene prodotta l'istruzione che assegna il valore iniziale alla variabile R(E)
  - viene calcolato l'insieme $\mathcal{D}$ dei nodi distinti di $\mathcal{G}$
- Fase II. **Riduzione dell'ipergrafo**
  - Viene richiamata la procedura ricorsiva Eval passando come parametri $\mathcal{G}$ e $\mathcal{D}$.

La procedura Eval($\mathcal{G}$,$\mathcal{D}$) se l'ipergrafo $\mathcal{G}$ passato come parametro ha più di un edge-relazione ed è connesso deve decidere quale edge eliminare. A tale scopo verifica se esiste almeno un edge-relazione che soddisfa le seguenti condizioni:

i) Interseca (ha intersezione non vuota con) almeno un altro edge-relazione
ii) Non interseca alcun (ha intersezione vuota con tutti gli altri) edge-condizione.

Se un tale edge-relazione esiste, costruisce l'insieme C degli edge-relazione "candidati" all'eliminazione nel modo seguente.

Inizialmente C consiste di tutti gli edge-relazione che soddisfano i) ed ii); quindi C viene ridotto nel modo seguente:

(a) se esiste un edge-relazione che rappresenta una relazione *"piccola"* (cioè un edge E tale che R(E) è ottenuto applicando una selezione o un semijoin) vengono eliminati da C tutti gli edge-relazione che *non rappresentano* relazioni *"piccole"*

(b) se esiste un edge-relazione che disconnette $\mathcal{G}$, vengono eliminati da C tutti gli edge-relazione che *non disconnettono* $\mathcal{G}$. (si preferisce eseguire un join a tre, piuttosto che due join binari)

A questo punto un qualsiasi edge-relazione rimasto in C può essere eliminato da $\mathcal{G}$ e viene eseguita la sequenza di istruzioni (3) con le seguenti eccezioni:

- se uno degli $\mathcal{H}_i$ ha un solo edge-relazione F viene omesso il semijoin R(F):=R(F)⋉R(E)
- se l'insieme dei nodi distinti di un $\mathcal{H}_i$ è vuoto viene omesso dal join Result($\mathcal{H}_i$).

Se non esiste un edge-relazione che soddisfa i) ed ii) la procedura Eval elimina da $\mathcal{G}$ un edge-condizione.

La scelta dell' edge-condizione da eliminare viene fatta considerando per ogni edge-condizione E l'ipergrafo $\mathcal{H}_E$ ottenuto eliminando E da $\mathcal{G}$.
Se esiste un edge-condizione E tale che:

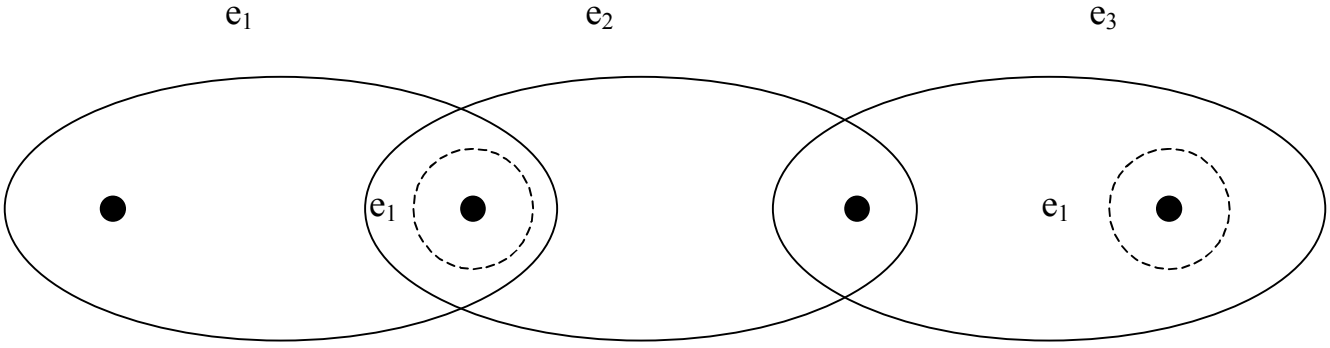
(p1) nell'ipergrafo $\mathcal{H}_E$ esiste un edge-relazione che soddisfa i) ed ii) e rappresenta una relazione "piccola"

allora viene eliminato un qualsiasi edge-condizione che soddisfa la proprietà (p1); altrimenti, se esiste un edge-condizione E tale che:

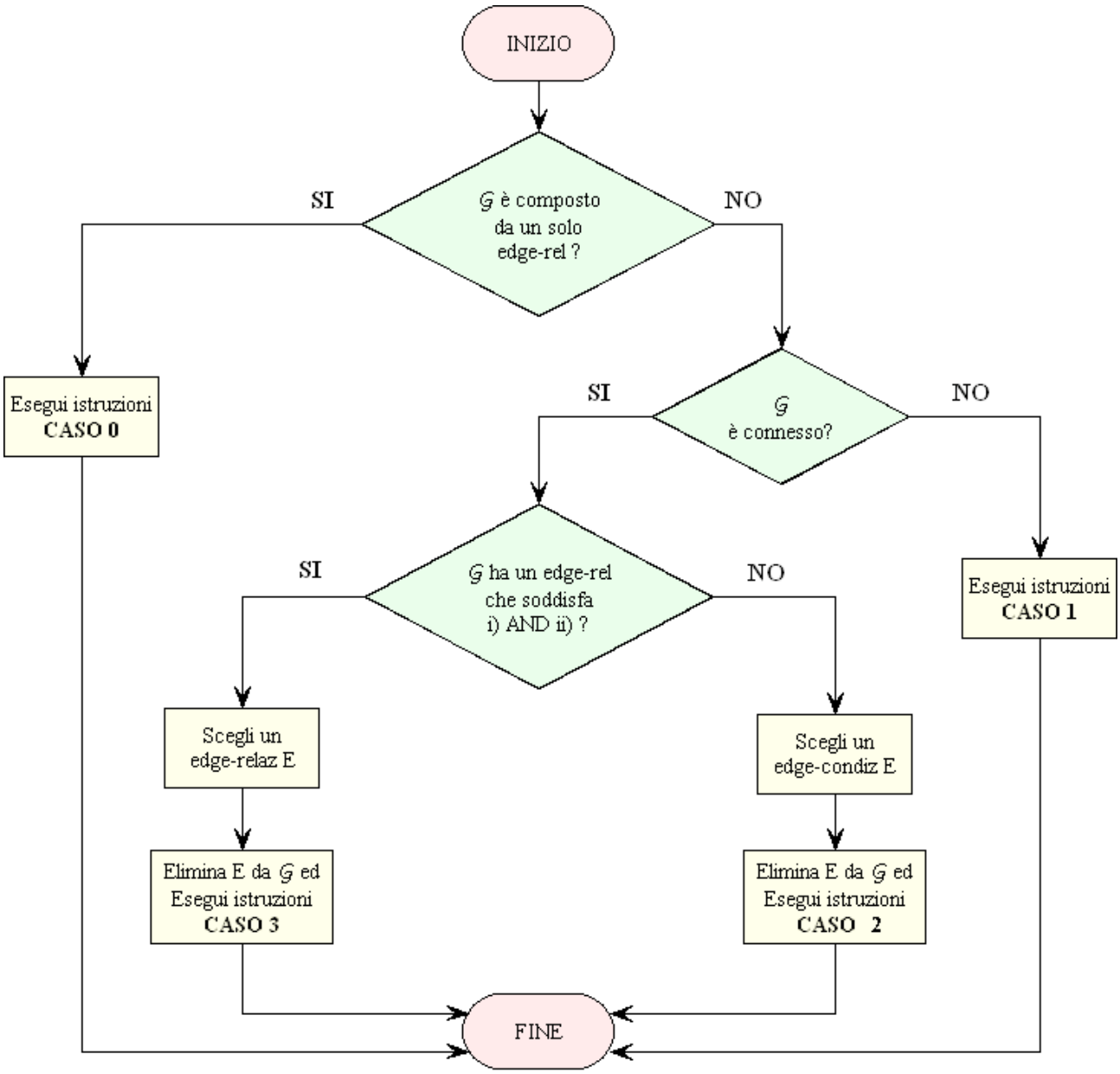
(p2) nell'ipergrafo $\mathcal{H}_E$ esiste un edge-relazione che soddisfa i) ed ii) e la cui eliminazione disconnette $\mathcal{H}_E$

allora viene eliminato un qualsiasi edge-condizione che soddisfa la proprietà (p2); altrimenti viene eliminato un qualsiasi altro edge-condizione.

Supponiamo, ad esempio che $\mathcal{G}$ sia l'ipergrafo in figura e che nessun edge-relazione rappresenti una relazione piccola:



In questo caso si elimina l'edge-condizione e_4. Infatti in He_5, l'edge-relazione e_3 soddisfa i) e ii), ma non rappresenta una relazione piccola, mentre in He_4 ci sono due edge-relazione (e_1 o e_2) che soddisfano i) e ii), e uno di essi (e_2) è tale che la sua eliminazione disconnette He_4


INIZIO
$\mathcal{G}$ è composto da un solo edge-rel ?
SI
NO
Esegui istruzioni CASO 0
$\mathcal{G}$ è connesso?
SI
NO
Esegui istruzioni CASO 1
$\mathcal{G}$ ha un edge-rel che soddisfa i) AND ii) ?
SI
NO
Scegli un edge-relaz E
Scegli un edge-condiz E
Elimina E da $\mathcal{G}$ ed Esegui istruzioni CASO 3
Elimina E da $\mathcal{G}$ ed Esegui istruzioni CASO 2
FINE

Esempi di applicazione dell'algoritmo:

Esempio 1
Nell'esempio visto precedentemente gli edge vengono eliminati nell'ordine già visto. Infatti

- nell'ipergrafo iniziale $\mathcal{H}1$ ci sono tre edge-relazione (S, C e D) che soddisfano le proprietà i) e ii) e solo una di queste (D) rappresenta una relazione "piccola" (infatti R(D):=$\square_{D.COGNOME='Rossi'}$ (DOCENTE)); quindi viene eliminato D
- nell'ipergrafo $\mathcal{H}2$ ci sono due edge-relazione (S, C) che soddisfano le proprietà i) e ii) e solo una di queste (C) rappresenta una relazione "piccola" (infatti R(C) := R(C) ▷ R(D)); quindi viene eliminato C
- nell'ipergrafo $\mathcal{H}3$ c'è un solo edge-relazione (S) che soddisfa le proprietà i) e ii) quindi viene eliminato S
- nell'ipergrafo $\mathcal{H}4$ non c'è nessun edge-relazione che soddisfa le proprietà i) e ii) e quindi viene eliminato l'unico edge-condizione { VOTO }
- l'ipergrafo $\mathcal{H}5$ è costituito da un unico edge-relazione (E) che viene eliminato.

Tenendo conto di quanto già detto sui nodi distinti degli ipergrafi $\mathcal{H}1$, $\mathcal{H}2$, $\mathcal{H}3$, $\mathcal{H}4$ e $\mathcal{H}5$, il piano di esecuzione generato è il seguente:

R(S) := STUDENTE
R(E) := ESAME
R(C) := CORSO
R(D) := $\square_{D.COGNOME='Rossi'}$ (DOCENTE)
R(C) := R(C) ▷ R(D);
R(E) := R(E) ▷ R(C);
R(E) := R(E) ▷ R(S);
Result($\mathcal{H}5$) := $\square_{VOTO,E.MATR,E.C\#}$ (R(E));
Result($\mathcal{H}4$) := $\square_{S.MATR,C.C\#}$ ($\square_{VOTO>27}$(Result($\mathcal{H}5$)));
Result($\mathcal{H}3$) := $\square_{S.COGNOME,C.C\#}$ (R(S) ▷◁ Result($\mathcal{H}4$));
Result($\mathcal{H}2$) := $\square_{S.COGNOME,D.D\#}$ (R(C) ▷◁ Result($\mathcal{H}3$));
Result($\mathcal{H}1$) := $\square_{S.COGNOME}$(R(D) ▷◁ Result($\mathcal{H}2$));

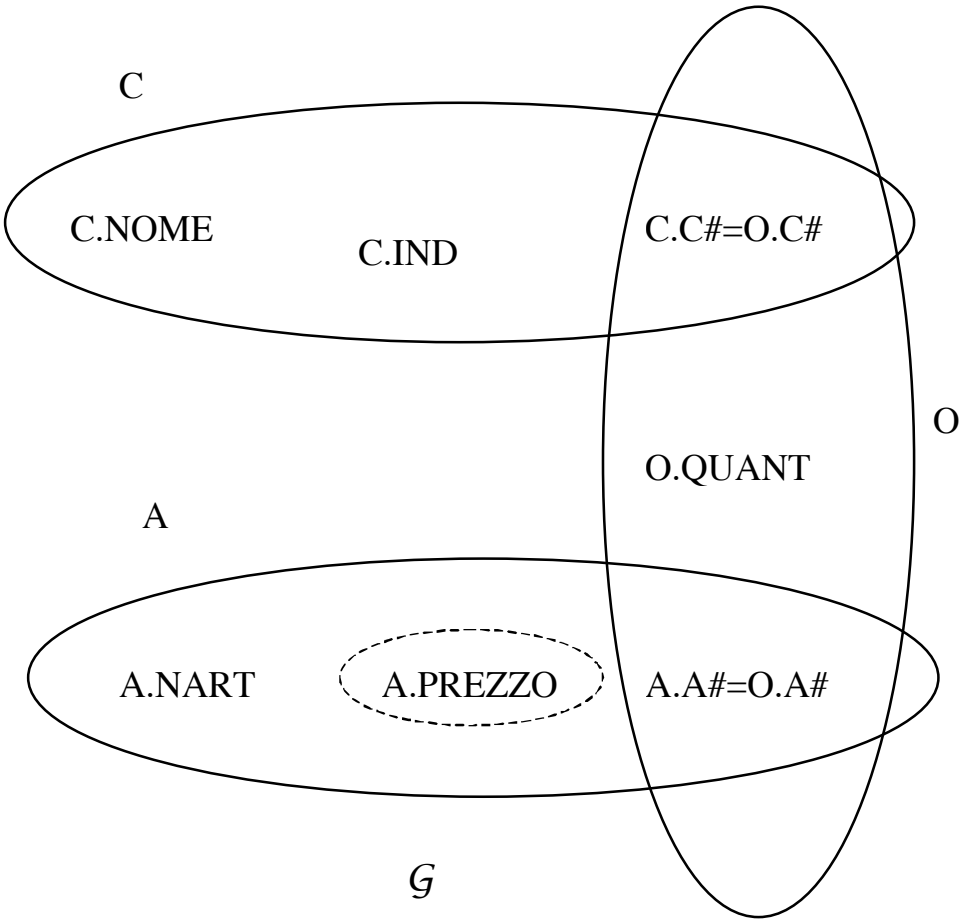
Esempio 2
Abbiamo i seguenti schemi di relazione:

CLIENTE (C#, NOME, IND)
ORDINE (C#, A#, QUANT)
ARTICOLO (A#, NART, PREZZO)

e vogliamo conoscere i **nomi** dei **clienti** che hanno **ordinato 100 pezzi** di un **articolo** con **prezzo>1000**, espressa in QUEL come:

range of	t1	is	CLIENTE
range of	t2	is	ORDINE
range of	t3	is	ARTICOLO
retrieve	t1.NOME		
where	t1.C# = t2.C# **AND** t2.A# = t3.A# **AND** t2.QUANT=100 **AND** t3.PREZZO>1000		

Fase I□Costruzione dell'ipergrafo $\mathcal{G}$□



Calcolo dell'insieme $\mathcal{D}$ dei nodi distinti:

$\mathcal{D}$ ($\mathcal{G}$)={NOME}

Inizializzazione:
R(C):= CLIENTE
R(O):= $\sigma_{QUANT=100}$ (ORDINE)
R(A):= ARTICOLO

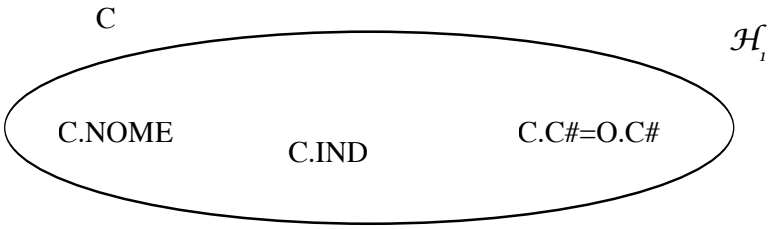
Fase II: Chiamata di Eval($\mathcal{G}$,$\mathcal{D}$)

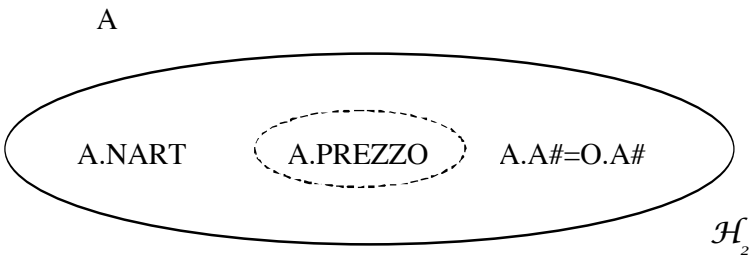
Eval($\mathcal{G}$,$\mathcal{D}$);

$\mathcal{G}$ è costituito da un solo edge-relazione? NO.
$\mathcal{G}$ è connesso? SI
Esiste un edge-relazione che soddisfa i) e ii) ? SI.
C = {C,O}
Esiste un edge-relazione in C che rappresenta una relazione "piccola"? SI (O)
Elimino C e in C resta solo O che va eliminato dall'ipergrafo $\mathcal{G}$.

Abbiamo eliminato un edge-relazione siamo quindi nel caso **(3).**
Qui siamo di fronte ad un caso particolare in cui è inutile fare i semijoin, in quanto eliminando O si ottiene un ipergrafo con due componenti connesse ciascuna delle quali ha un unico edge-relazione.. Quindi il primo "for" non viene eseguito.

Eliminando l'edge-relazione O ottengo l'ipergrafo $\mathcal{H}$ composto da due componenti connesse $\mathcal{H}1$ e $\mathcal{H}2$.
Il secondo "for" calcola quindi gli insiemi di nodi distinti $\mathcal{D}1$ e $\mathcal{D}2$ (di $\mathcal{H}1$ e $\mathcal{H}2$, rispettivamente) e effettua le chiamate Eval($\mathcal{H}1$, $\mathcal{D}1$) e Eval($\mathcal{H}2$, $\mathcal{D}2$). Infine viene prodotta l'istruzione Result($\mathcal{G}$):=$\pi_{\mathcal{D}}$(R(O)⋉ Result($\mathcal{H}_1$) ⋉ Result($\mathcal{H}_2$))





$\mathcal{D}1$ = {C.C# = O.C#} ∪ {NOME};
$\mathcal{D}2$ = {A.A# = O.A#};

Eval($\mathcal{H}1$, $\mathcal{D}1$)

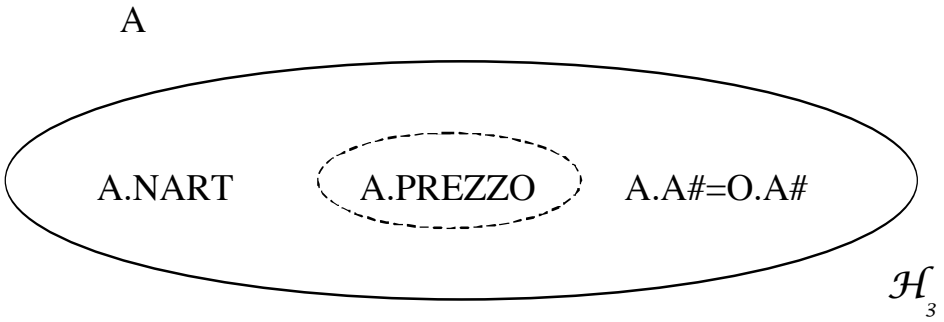
$\mathcal{H}1$ è composto da un unico edge-relazione? SI. Siamo nel caso **(0),** per cui viene prodotta l'assegnazione

Result($\mathcal{H}1$):= $\pi_{NOME, C\#}$ (R(E1))

termina la chiamata di Eval($\mathcal{H}1$, $\mathcal{D}1$)

Eval($\mathcal{H}2$, $\mathcal{D}2$)

$\mathcal{H}2$ è composto da un unico edge-relazione? NO (c'è un edge-condizione)
$\mathcal{H}2$ è connesso? SI
Esiste un edge-relazione Ei che soddisfa i) e ii) ? NO
Esiste un edge-condizione che soddisfa p1)? NO
Esiste un edge-condizione che soddisfa p2)? NO
Viene eliminato l'unico edge-condizione {PREZZO} ottenendo l'ipergrafo $\mathcal{H}3$.
Siamo nel caso **(2)**,
Viene calcolato quindi l'insieme di nodi distinti $\mathcal{D}3$ di $\mathcal{H}3$ e viene effettuata la chiamata Eval($\mathcal{H}3$, $\mathcal{D}3$) .
Infine viene prodotta, l'istruzione Result($\mathcal{H}2$):= $\pi_{\mathcal{D}2}$($\sigma_{PREZZO>1000}$(Result ($\mathcal{H}3$)))



$\mathcal{D}3$ = {PREZZO} $\cup$ {A#};

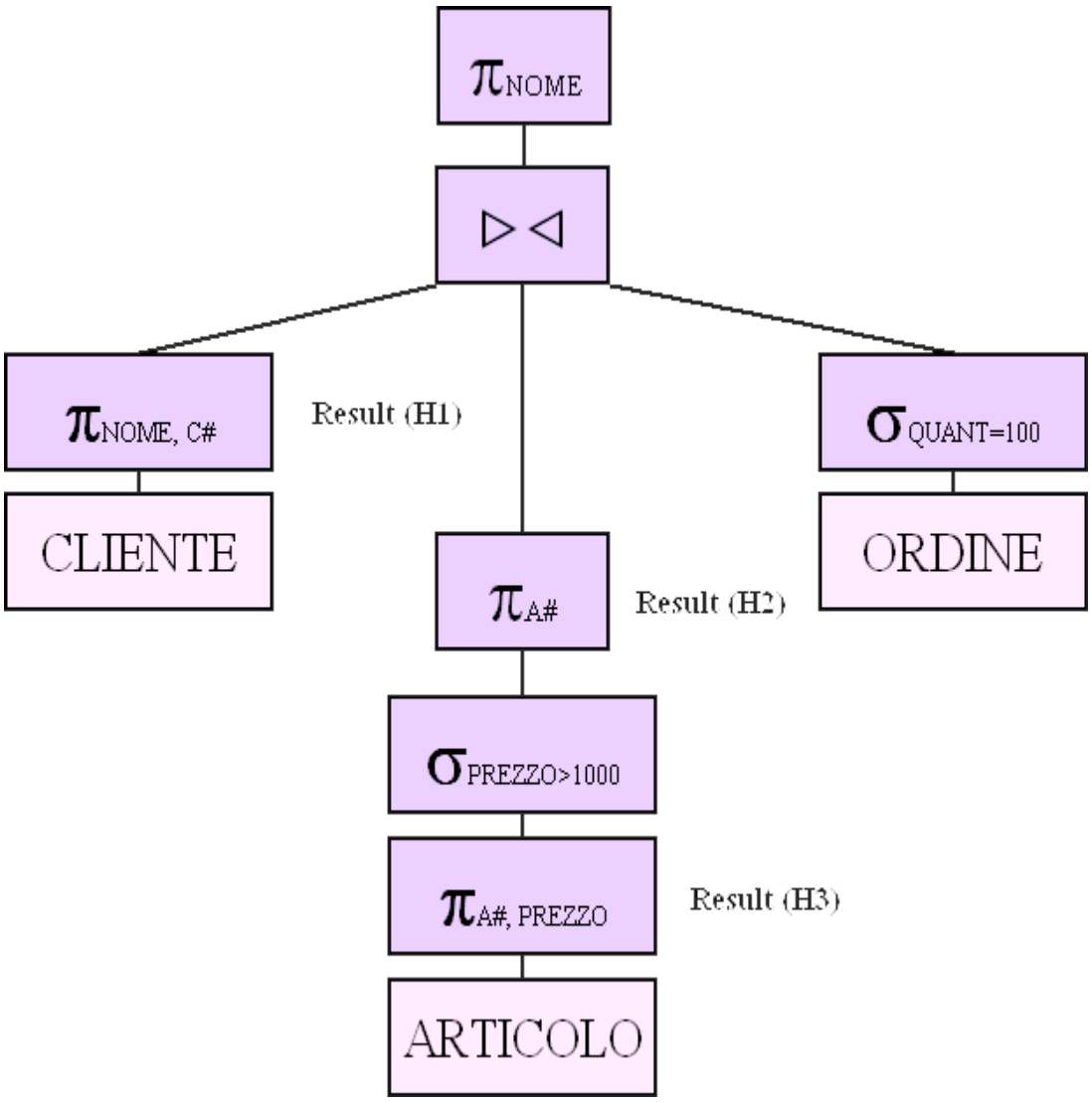
Eval($\mathcal{H}3$, $\mathcal{D}3$)

$\mathcal{H}3$ è composto da un unico edge-relazione. Siamo nel caso **(0)**, e quindi viene prodotta l'assegnazione:
Result($\mathcal{H}3$):= $\pi_{PREZZO, A\#}$ (R(E3));
termina la chiamata di Eval($\mathcal{H}3$, $\mathcal{D}3$)

Il codice prodotto è quindi il seguente:

R(C):= CLIENTE
R(O):= $\sigma_{QUANT=100}$ (ORDINE)
R(A):= ARTICOLO
Result($\mathcal{H}1$):= $\pi_{NOME, C\#}$ (R(C));
Result($\mathcal{H}3$):= $\pi_{PREZZO, A\#}$ (R(A));
Result($\mathcal{H}2$):= $\pi_{A\#}$ ($\sigma_{PREZZO>1000}$(Result ($\mathcal{H}3$)));
Result($\mathcal{G}$):=π_{NOME}(R(O)$\triangleright\!\triangleleft$ Result($\mathcal{H}_1$) $\triangleright\!\triangleleft$ Result($\mathcal{H}_2$));

L'albero sintattico prodotto dall'espressione ottimizzata è illustrato qui di seguito:


πNOME
⋈
πNOME, C#
Result (H1)
σQUANT=100
CLIENTE
πA#
Result (H2)
ORDINE
σPREZZO>1000
πA#, PREZZO
Result (H3)
ARTICOLO