ESECUZIONE DI INTERROGAZIONI

- Abbiamo visto finora come organizzare i dati in un D B

- Normalmente le decisioni sulle strutture da allocare sono determinate durante la progettazione fisica della DB

- La modifica di tali strutture in seguito puo' essere costosa

- Quindi quando una query e' presentata al sistema occorre determinare il modo piu' efficiente per eseguirla usando le strutture disponibili

- Per queries complesse esistono piu' strategie possibili

- Anche se il costo di determinare la strategia ottima puo' essere alto, e' tuttavia molto spesso utile eseguire l'ottimizzazione

- Le tecniche di ottimizzazione sono state sviluppate principalmente per il modello relazionale
  (tecniche per gli OODBMSs costituiscono un'area di ricerca aperta)

PASSI NELL'ESECUZIONE DI UNA INTERROGAZIONE

- Parsing
  Viene controllata la correttezza sintattica della query e ne viene generata una rappresentazione interna (parse tree)

- Trasformazioni algebriche
  Questo e' il primo passo di ottimizzazione
  La query viene trasformata in una query equivalente ma piu' efficiente da eseguire (ci si basa sulle proprieta' dell'algebra relazionale)

- Selezione della strategia
  Si determina in modo preciso come la query sara' eseguita (per esempio si determina che indici si useranno)
  La scelta della strategia e' fatta principalmente in base al numero di accessi a disco

- Esecuzione della strategia scelta

E' possibile eseguire alcuni dei passi a tempo di compilazione del programma (DB2 e System R usano questa strategia) o a tempo di esecuzione (Oracle usa questa strategia)

EQUIVALENZA DI ESPRESSIONI
Selezione

- Relazioni esempio:
  customer (c-name, street, c-city)
  deposit (b-name, account-number, c-name, balance)
  branch(b-name, assets, b-city)

- Supponiamo di voler determinare assets e b-name per le filiali che hanno clienti residenti a Port Chester

- Questa interrogazione e' espressa in RA come segue
  $\Pi_{\text{b-name, assets}} (\sigma_{\text{c-city="Port Chester"}}$
  $(\text{customer } |x| \text{ deposit } |x| \text{ branch}))$ (Q1)

- Questa espressione genera una relazione come join naturale delle tre relazioni

- Questa relazione intermedia puo' risultare troppo grande per risiedere in MM e quindi puo' essere necessario memorizzarla su disco

EQUIVALENZA DI ESPRESSIONI
Selezione

- E' possibile eseguire Q1 piu' efficientemente se si riesce a ridurre la dimensione della relazione intermedia generata come join naturale

- Una espressione equivalente ma piu' efficiente e' la seguente:
  $\Pi_{\text{b-name, assets}}$ (($\sigma_{\text{c-city="Port Chester"}}$ (customer))
  |x| deposit |x| branch)

- L'espressione precedente suggerisce una prima regola per trasformare queries

  Eseguire le operazioni di selezione (σ) *il piu' presto possibile* (R1)

EQUIVALENZA DI ESPRESSIONI
Selezione

- Consideriamo una query uguale alla precedente con in piu' la restrizione che i clienti abbiano un conto con un bilancio maggiore di 1000

- Questa interrogazione e' espressa in RA come segue

  $\Pi_{\text{b-name, assets}}$ (σ $_{\text{c-city="Port Chester" AND balance >1000}}$
  (customer |x| deposit |x| branch)) (Q2)

- Non e' possibile applicare la selezione

  c-city="Port Chester" AND balance >1000

  direttamente alla relazione customer poiche' il predicato coinvolge attributi di customer e deposit

- Osserviamo che:

  (customer |x| deposit |x| branch) ≡
  ((customer |x| deposit) |x| branch)

- Pertanto Q1 puo' essere riscritta come segue

  $\Pi_{\text{b-name, assets}}$ ((σ $_{\text{c-city="Port Chester" AND balance >1000}}$
  (customer |x| deposit)) |x| branch)

EQUIVALENZA DI ESPRESSIONI
Selezione

- Consideriamo la sotto-query

  $\sigma_{\text{c-city="Port Chester" AND balance >1000}}$
  (customer |x| deposit) (E1)

- Questa sottoquery puo' essere riscritta come segue

  $\sigma_{\text{c-city="Port Chester"}}$ ($\sigma_{\text{balance >1000}}$
  (customer |x| deposit)) (E2)

- L'espressione E2 permette di applicare la regola R1

- Pertanto la sottoquery puo' essere riscritta come segue:

  ($\sigma_{\text{c-city="Port Chester"}}$ (customer)) |x|
  ($\sigma_{\text{balance >1000}}$(deposit))

- Le trasformazioni precedenti sono basate sulla seguente regola

  Trasforma espressioni della forma $\sigma_{P1 \text{ AND } P2}$ (e)
  in espressioni della forma σ_{P1} (σ_{P2}(e))
  dove P1 e P2 sono predicati ed e e' una espressione algebrica (R2)

- R2 e' basata sulle seguenti equivalenze tra RA espr.

  σ_{P1} (σ_{P2}(e)) $\equiv$ σ_{P2} (σ_{P1}(e)) $\equiv$ $\sigma_{P1 \text{ AND } P2}$(e))

EQUIVALENZA DI ESPRESSIONI
Join naturale

- In generale trasformare le interrogazioni in modo che le selezioni siano applicate il prima possibile permette di ridurre la dimensione dei risultati intermedi

- Un altro modo per ridurre tale dimensione e' di determinare un ordine ottimale nell'esecuzione delle operazioni di join naturale

- Questa strategia e' basata sulla proprieta' del join naturale di essere una operazione associativa:
  $(r1 \; |x| \; r2) \; |x| \; r3 \equiv r1 \; |x| \; (r2 \; |x| \; r3)$

- Sebbene le due precedenti espressioni sono equivalenti, il costo di calcolarle puo' essere diverso

- Consideriamo nuovamente la query
  $\Pi_{\text{b-name, assets}} ((\sigma_{\text{c-city="Port Chester"}} (\text{customer}))$
  $|x| \; \text{deposit} \; |x| \; \text{branch})$

- Una possibilita' e' di eseguire il join naturale
  deposit |x| branch e di eseguire il join naturale del risultato con il risultato della sottoquery
  $(\sigma_{\text{c-city="Port Chester"}} (\text{customer}))$

EQUIVALENZA DI ESPRESSIONI
Join naturale

- T1 = deposit |x| branch
  (σ $_{\text{c-city="Port Chester"}}$ (customer)) |x| T1

- E' da notare che pero' la relazione temporanea T1 e' molto probabilmente di dimensioni elevate in quanto contiene una tupla per ogni deposito

- Invece la relazione che si otterebbe dalla sottoquery
  σ $_{\text{c-city="Port Chester"}}$ (customer)
  e' molto probabilmente piccola in quanto per una banca con molte filiali e' improbabile che tutti i clienti abbiano la residenza nello stesso posto

- Quindi se calcoliamo
  (σ $_{\text{c-city="Port Chester"}}$ (customer))|x| deposit
  otteniami una tupla per ogni conto intestato ad un cliente che abita a Port Chester

- Pertanto la relazione temporanea che si deve memorizzare e' molto piu' piccola

- $\Pi_{\text{b-name, assets}}$
  (((σ $_{\text{c-city="Port Chester"}}$ (customer)) |x| deposit)
  |x| branch) (Q3)

EQUIVALENZA DI ESPRESSIONI
Join naturale

- Un'altra proprieta' del join e' di essere commutativo
  r1 |x| r2 ≡ r2 |x| r1

- Pertanto nel caso di Q3 potremmo considerare una strategia alternativa
  $\Pi_{\text{b-name, assets}}$
  $(((\sigma_{\text{c-city="Port Chester"}}$ (customer)) |x| branch)
  |x| deposit)

- Questa strategia non e' molto buona in quanto customer e branch non hanno attributi a comune e quindi il join natturale diventa un prodotto Cartesiano

- Se customer ha c tuple e branch ne ha b si ottiene un totale di b*c tuple

- Quindi si otterrebbe una relazione temporanea di dimensione elevata

EQUIVALENZA DI ESPRESSIONI
Proiezione

- L'operazione di proiezione ha la proprieta' di ridurre la dimensione della relazione: quindi e' conveniente applicare le proiezioni il prima possibile

- Consideriamo la query
  $\Pi_{\text{b-name, assets}}$
  $(((\sigma_{\text{c-city="Port Chester"}}(\text{customer})) \; |x| \; \text{deposit})$
  $|x| \; \text{branch})$ (Q3)

- Quando si calcola la sottoquery
  $((\sigma_{\text{c-city="Port Chester"}}(\text{customer})) \; |x| \; \text{deposit})$
  otteniamo una relazione il cui schema e'
  (c-name, c-city, b-name, account-number, balance)

- E' possibile eliminare alcuni attributi da questo schema. Gli unici attributi da non eliminare sono quelli che
  - appaiono nel risultato della query
  - sono necessari in operazioni successive

- Pertanto Q3 puo' essere riscritta come segue:
  $\Pi_{\text{b-name, assets}}((\Pi_{\text{b-name}}$
  $((\sigma_{\text{c-city="Port Chester"}}(\text{customer})) \; |x| \; \text{deposit}))$
  $|x| \; \text{branch})$

EQUIVALENZA DI ESPRESSIONI

Altre operazioni

- Le equivalenze viste sono le piu' utili in quanto coinvolgono operazioni usate molto spesso

- Altre equivalenze utili sono:

  - $\sigma_P(e1 \text{ U } e2) \equiv \sigma_P(e1) \text{ U } \sigma_P(e1)$
  - $\sigma_P(e1 - e2) \equiv \sigma_P(e1) - \sigma_P(e1)$
  - $(e1 \text{ U } e2) \text{ U } e3 \equiv e1 \text{ U } (e2 \text{ U } e3)$
  - $e1 \text{ U } e2 \equiv e2 \text{ U } e1$

EQUIVALENZA DI ESPRESSIONI
Sommario delle leggi di equivalenza

1. Leggi commutative per il join e il prodotto Cartesiano

   $e1\ |x|_F\ e2 \equiv r2\ |x|_F\ e1$

   $e1\ |x|\ e2 \equiv e2\ |x|\ e1$

   $e1 \times e2 \equiv e2 \times e1$

2. Leggi associative per il join e il prodotto Cartesiano

   $(e1\ |x|_{F1}\ e2)\ |x|_{F2}\ e3 \equiv e1\ |x|_{F1}\ (e2\ |x|_{F2}\ e3)$

   $(e1\ |x|\ e2)\ |x|\ e3 \equiv e1\ |x|\ (e2\ |x|\ e3)$

   $(e1 \times e2) \times e3 \equiv e1 \times (e2 \times e3)$

3. Cascata di proiezioni

   $\Pi_{A1,....,An}(\Pi_{B1,....,Bm}(e)) \equiv \Pi_{A1,....,An}(e)$

   Notare che $\{A1,.....,An\} \subseteq \{B1,.....,Bm\}$ affinche' la cascata sia legale

4. Cascata di selezioni

   $\sigma_{P1}(\sigma_{P2}(e)) \equiv \sigma_{P1\ AND\ P2}(e))$

   inoltre poiche' P1 AND P2 = P2 AND P1

   segue che la selezione gode della proprieta' commutativa

   $\sigma_{P1}(\sigma_{P2}(e)) \equiv \sigma_{P2}(\sigma_{P1}(e))$

EQUIVALENZA DI ESPRESSIONI
Sommario delle leggi di equivalenza

5. Commutazione di selezione e proiezione

   Se una selezione con predicato P coinvolge solo gli attributi A1,......,An, allora

   $$\Pi_{A1,....,An}(\sigma_P(e)) \equiv \sigma_P(\Pi_{A1,....,An}(e))$$

   Piu' in generale, se il predicato P coinvolge anche gli attributi B1,.....,Bm che non sono tra gli attributi A1,......,An allora

   $$\Pi_{A1,....,An}(\sigma_P(e)) \equiv$$
   $$\Pi_{A1,....,An}(\sigma_P(\Pi_{A1,....,An,B1,....,Bm}(e)))$$

6. Commutazione di selezione e prodotto Cartesiano

   Se una selezione con predicato P coinvolge solo gli attributi di e1, allora

   $$\sigma_P(e1 \times e2) \equiv \sigma_P(e1) \times e2$$

   Come corollario, se P=P1 AND P2 dove P1 coinvolge solo gli attributi di e1 e P2 quelli di e2, usando le regole (1), (4), e (6) si ottiene

   $$\sigma_P(e1 \times e2) \equiv \sigma_{P1}(e1) \times \sigma_{P2}(e2)$$

   Inoltre se P1 coinvolge solo attributi di e1, mentre P2 coinvolge attributi di e1 e di e2

   $$\sigma_P(e1 \times e2) \equiv \sigma_{P2}(\sigma_{P1}(e1) \times e2)$$

EQUIVALENZA DI ESPRESSIONI
Sommario delle leggi di equivalenza

7. Commutazione di selezione e unione

   Se e = e1 U e2 possiamo assumere che gli attributi di e1 ed e2 hanno gli stessi nomi degli attributi di e, o almeno, che esiste una corrispondenza che associa ad ogni attributo di e rispettivamente un unico attributo di e1 ed un unico attributo di e2

   $$\sigma_P(e1 \cup e2) \equiv \sigma_P(e1) \cup \sigma_P(e2)$$

   Se gli attributi di e1 e/o di e2 sono diversi P deve essere modificato usando il nome appropriato

8. Commutazione di selezione e differenza

   $$\sigma_P(e1 - e2) \equiv \sigma_P(e1) - \sigma_P(e2)$$

9. Commutazione di proiezione e prodotto Cartesiano

   Sia A1,....,An una lista di attributi di cui gli attributi B1,.....,Bm siano attributi di e1, e i rimanenti C1,.....,Ck siano attributi di e2

   $$\Pi_{A1,....,An}(e1 \times e2) \equiv \Pi_{B1,....,Bm}(e1) \times \Pi_{C1,....,Ck}(e2)$$

10. Commutazione di proiezione e unione

    $$\Pi_{A1,....,An}(e1 \cup e2) \equiv \Pi_{A1,....,An}(e1) \cup \Pi_{A1,....An}(e2)$$

STIMA DEL COSTO DI ESECUZIONE

- La strategia scelta dipende dalla dimensione di ogni relazione e dalla distribuzione dei valori nelle varie colonne

- Normalmente i DBMSs mantengono statistiche per ogni relazione memorizzata:
  1. n_r numero di tuple nella relazione r
  2. S_r dimensione di una tupla della relazione r in bytes (per tuple a lunghezza fissa, altrimenti si usano valori medi)
  3. V(A, r) il numero di valori distinti che appaiono nella relazione r per l'attributo A

- Le prime due statistiche permettono di stimare il costo di un prodotto Cartesiano
  - r x s conterra' un numero di tuple pari a $n_r * n_s$
  - una tupla di r x s ha dimensione pari a
    $S_r + S_s$
  - pertanto la dimensione di r x s e' pari a
    $n_r * n_s * (S_r + S_s)$

STIMA DEL COSTO DI ESECUZIONE

- La terza statistica permette di stimare quante tuple verificano un predicato di selezione della forma:
  <attribute-name> = <value>

- Per eseguire una stima corretta e' necessario determinare la frequenza di ogni valore per una data colonna

- Sotto l'ipotesi che ogni valore appare con la stessa probabilita', allora si stima che $\sigma_{A=a}(r)$ ha una seleziona un numero di tuple pari a $n_r/V(A, r)$
  (La quantita' $1/V(A,r)$ e' detta *fattore di selettivita' del predicato*)

- In alcune situazioni non e' realistico assumere l'equiprobabilita' dei valori

- Per esempio se consideriamo la relazione deposit e l'attributo b-name, ci si puo' aspettare che le filiali piu' grandi abbiano piu' depositi e quindi alcuni nomi di filiale appariranno con maggiore probabilita' di altri

- Tuttavia l'ipotesi di equiprobabilita' e' una buona approssimazione in molto casi

STIMA DEL COSTO DI ESECUZIONE
Join Naturale

Siano r1 e r2 due relazioni di schema rispettivamente R1 e R2

- se $R1 \cap R2 = \emptyset$, allora r1 |x| r2 e' lo stesso di r1 x r2 e si puo' usare la stessa formula di stima usata per il prodotto Cartesiano

- se $R1 \cap R2$ e' una chiave per R1, allora una tupla di r2 e' connessa tramite il join con al piu' una tupla di r1; quindi $n_{(r1\ |x|\ r2)} \leq n_{r2}$

- se $R1 \cap R2$ non e' una chiave per nessuno dei due schemi, si deve usare la terza statistica. Consideriamo una tupla t di r1, e supponiamo che $R1 \cap R2 = \{A\}$
  Il numero di tuple di r2 che hanno t[A] come valore per A sono $n_{r2}/V(A, r2)$
  Pertanto la tupla t produce $n_{r2}/V(A, r2)$ tuple di r1 |x| r2
  Se consideriamo tutte le tuple in r1, allora otteniamo $n_{r1} * n_{r2}/V(A, r2)$ (i)

- Scambiando i ruoli

STIMA DEL COSTO DI ESECUZIONE
Join Naturale

- Scambiando i ruoli nel calcolo della stima otteniamo
  $n_{r1} * n_{r2} / V(A, r1)$ (ii)

  Espressioni (i) ed (ii) sono uguali se V(A, r2) =V(A, r1)

  Se V(A, r2) ~=V(A, r1) sono differenti allora esiste qualche tupla *dangling* che non partecipa al join

  In questo caso il minore dei due valori costituisce una stima migliore

- Esempio (1):

r1	A	B	r2	A	C	
	a1	b1		a1	c1	n_{r1}= 4
	a1	b2		a2	c2	V(A,r1)=2
	a2	b3		a1	c3	
	a2	b4		a1	c6	n_{r2}= 6
				a2	c5	V(A,r2)=2
				a2	c7	

$n_{r1} * n_{r2} / V(A, r2) = (4*6)/2=12$

STIMA DEL COSTO DI ESECUZIONE
Join Naturale

- Esempio (2):

r1	A	B	r2	A	C	
	a1	b1		a1	c1	n_{r1}= 6
	a1	b2		a2	c2	V(A,r1)=3
	a2	b3		a1	c3	
	a2	b4		a1	c6	n_{r2}= 6
	a3	b2		a2	c5	V(A,r2)=2
	a3	b3		a2	c7	

$n_{r1} * n_{r2} / V(A, r2) = (6*6)/2=18$
$n_{r1} * n_{r2} / V(A, r1) = (6*6)/3=12$

- Le stime precedenti non vanno molto bene se le due relazioni hanno pochi valori in comune per l'attibuto in comune; in tal caso la stima della dimensione del join risulta troppo alta

- Nelle pratica, d'altra parte, questi casi non si verificano molto spesso

- Nel caso in cui si dovessero avere molte tuple dangling, occorre applicare dei fattori di correzione

STIMA DEL COSTO DI ESECUZIONE
Strategie di join

- Abbiamo visto come stimare la dimensione del risultato di un'operazione di join

- Il costo dell'esecuzione di un join e' influenzato da vari fattori:
  - l'ordine fisico delle tuple in una relazione
  - la presenza e il tipo di indici
    (*clustering* vs *non-clustering*)
  - il costo di costruire un indice temporaneo allo scopo di eseguire il join

- Consideriamo l'espressione
  deposit |x| customer
  e supponiamo che non ci siano indici; inoltre
  - $n_{deposit} = 10{,}000$
  - $n_{customer} = 200$

- Strategie possibili
  - iterazione semplice
  - iterazione orientata ai blocchi
  - merge-join
  - uso di indici
  - three-way join

STIMA DEL COSTO DI ESECUZIONE
Iterazione semplice

- Si accede una tupla di deposit (*outer relation*) e si confronta con ogni tupla di customer (*inner relation*)

- **for each** tuple d **in** deposit **do**

```
begin
    for each tuple c in customer do
        begin
        test pair (d,c) to see if a tuple should be
        added to the result
        end
end
```

- Questa strategia richiede leggere ogni tupla di deposit una sola volta; questo puo' richiedere un massimo di 10,000 accessi

- Se le tuple sono clusterizzate allora il costo diminuisce sensibilmente;
  supponiamo di avere 20 tuple di deposit per blocco, il numero totale di accessi per la relazione deposit e' 10,000/20=500

STIMA DEL COSTO DI ESECUZIONE
Iterazione semplice

- Per quanto riguarda le tuple di customer queste vengono accedute per ogni tupla di deposit; quindi ogni tupla di customer viene acceduta 10,000 volte

- Poiche' le tuple di customer sono 200, il totale degli accessi e' 10,000*200 = 2,000,000

- Se le tuple di customer sono clusterizzate, il totale degli accessi si riduce

- Supponiamo di avere 20 tuple di customer per blocco; la scansione dell'intera relazione richiede solo l'accesso di 10 blocchi

- Quindi il costo della scansione della relazione customer per tutte le tuple di deposit e' 10,000*10=100,000

- Il costo totale dell'esecuzione del join nel caso in cui entrambe le relazioni siano clusterizzate e' 500 + 100,000 = 100,500

STIMA DEL COSTO DI ESECUZIONE
Iterazione orientata ai blocchi

- E' possibile migliorare la strategia precedente se si elaborano le relazioni sulla base dei blocchi e non delle tuple

- E' utile principalmente quando le tuple di una stessa relazione sono clusterizzate

-
```
for each block Bd of  deposit do
   begin
         for each block Bc of  customer do
               begin
                 for each tuple b in Bd do
                   begin
                     for each tuple c in Bc do
                           begin
               test pair (b,c) to see if  a tuple should be
               added to the result
                           end
                   end
               end
   end
```

STIMA DEL COSTO DI ESECUZIONE
Iterazione orientata ai blocchi

- Questa strategia esegue il join esaminando un intero blocco di tuple di deposit alla volta

- Quindi il costo dell'accesso di deposit e' lo stesso della strategia di iterazione semplice (i.e. 500)

- Questa strategia implica che dobbiamo comunque scandire la relazione customer piu' volte (ad un costo di 10 a scansione)

- Tuttavia rispetto alla strategia di iterazione semplice, dobbiamo scandire la relazione customer tante volte quanti sono i *blocchi* di deposit e non quante sono le tuple di deposit

- Quindi il numero di scansioni della relazione customer e' 500; quindi il costo totale di accesso a customer e' 500*10=5000 accessi

- Il costo totale di questa strategia e' quindi
  500 + 5000 = 5500

STIMA DEL COSTO DI ESECUZIONE
Iterazione orientata ai blocchi

- La scelta di deposit come outer relation e customer come inner e' stata arbitraria

- Se avessimo usato customer come outer relation e deposit come inner il costo finale della strategia sarebbe stato 5010

- Un vantaggio nell'usare la relazione piu' piccola come inner e' che se la relazione e' piccola abbastanza puo' risiedere tutto il tempo in MM

- Se ad esempio customer fosse abbastanza piccola di risiedere in memoria, la strategia richiederebbe solo 500 accessi per leggere deposit e 10 per leggere customer, per un totale di 510.

STIMA DEL COSTO DI ESECUZIONE
Merge-Join

- Nel caso in cui nessuna delle due relazioni sia piccola abbastanza da poter risiedere in MM e' possibile eseguire efficientemente il join se entrambe le relazioni sono ordinate in base al valore dell'attributo di join

- Relazioni customer e deposit ordinate in base all'attributo customer-name

- L'operazione di merge-join richiede di associare inizialmente un puntatore ad ogni relazione

- I puntatori inizialmente puntano alla prima tupla di ogni relazione

- Poiche' le tuple sono ordinate in base all'attributo di join ogni tupla viene letta esattamente una volta

- Nel caso delle relazioni customer e deposit il costo totale sarebbe 510 accessi
  (nell'ipotesi che le tuple di ogni relazione sono clusterizzate)

STIMA DEL COSTO DI ESECUZIONE
Merge-Join

- L'algoritmo non richiede che la relazione entri tutta in MM; e' sufficiente che tutte le tuple con lo stesso valore dell'attributo di join stiano in MM

- Questo e' possibile anche per relazioni di ampie dimensioni

- Lo svantaggio principale di questo metodo e' che richiede che le relazioni siano ordinate

- Tuttavia, poiche' e' molto efficiente puo' convenire ordinare le relazioni prima di eseguire il join

STIMA DEL COSTO DI ESECUZIONE
Merge-Join: algoritmo

```
pd := address of first tuple of deposit;
pc := address of first tuple of customer;
while (pc ~= null) do
    begin
        tc :=tuple to which pc points;
        Sc:= {tc};
        set pc to point to  next tuple of customer;
        done := false;
        while (not done) do
            begin
                tc ':=tuple to which pc points;
                if tc [customer-name] =tc '[customer-name]
                then begin
                        Sc:= Sc U {tc'};
                        set pc  to point to next tuple of customer;
                end
                else done:=true;
            end
        td:=tuple to which pd points;
        while (td [customer-name] < tc [customer-name]) do
            begin
                set pd to point to  next tuple of deposit;
                td:=tuple to which pd points;
            end
        while (td [customer-name] = tc [customer-name]) do
            begin
                for each t in Sc do
                    begin
                        compute t |x| td and add this to the result;
                    end
                set pd to next tuple of deposit;
                td:=tuple to which pd points;

            end
end.
```

STIMA DEL COSTO DI ESECUZIONE
Merge-Join: Esempio

D	N	B	C	N	D
a1	b1	i1	a1	c1	i6
a1	b2	i2	a1	c3	i7
a2	b4	i3	a3	c5	i8
a2	b5	i4	a4	c7	i9
a5	b7	i5	a5	c9	i10

(i1,......,i10 sono indirizzi)

pd=i1 pc= i6

tc=<a1,c1> Sc={<a1,c1>}

pc=i7 tc'=<a1,c3>

poiche' tc'[N]=tc[N] Sc={<a1,c1>, <a1,c3>}

pc=i8

pd=i1 td=<a1,b1>

td[N]<tc[N] ? NO

td[N]=tc[N] ? SI

Si esegue il join tra la tupla td corrente e tutte le tuple in Sc

R={<a1,b1,c1>, <a1,b1,c3>}

STIMA DEL COSTO DI ESECUZIONE
Merge-Join: Esempio

pd=i2 td=<a1,b2>
td[N]=tc[N] ? SI
R = {<a1,b1,c1>, <a1,b1,c3>, <a1,b2,c1>, <a1,b2,c3>}

pd=i3 td=<a2,b4>
td[N]=tc[N] ? NO

tc=<a3,c5> Sc={<a3,c5>}
pc=i9 tc'=<a4,c7>
tc[N]=tc'[N]? NO

td[N] < tc[N] ? SI (td[N]=a2 tc[N]=a3)
pd=i4 td=<a2,b5>
td[N] <tc[N]? SI (td[N]=a2 tc[N]=a3)

pd=i5 td=<a5,b7>
td[N]<tc[N]? NO (td[N]=a5 tc[N]=a3)
td[N]=tc[N]? NO

pc=i9 tc=<a4,c7> pc=10
td[N]<tc[N]? NO (td[N]=a5 tc[N]=a4)
td[N]=tc[N]? NO

pc=10 tc=<a5,c9> td[N]=tc[N]? SI
<a5,b7,c9> viene aggiunto al risultato

STIMA DEL COSTO DI ESECUZIONE

Uso di indici

- Molto spesso l'attributo di join e' la chiave di ricerca di un indice di una delle relazioni

- Esempio: supponiamo che esista un indice sull'attributo c-name della relazione customer

- La strategia di iterazione semplice e' molto piu' efficiente in tal caso

- ```
  for each tuple d of  deposit do
      begin
            for each tuple c of  customer do
                  begin
                   test pair (b,c) to see if  a tuple should be
                  added to the result
                  end
      end
  ```

- Data una tupla d di deposit non e' piu' necessario scandire l'intera relazione customer, ma e' sufficiente eseguire una ricerca sull'indice con il valore di c-name dato da d[c-name]
  ```

STIMA DEL COSTO DI ESECUZIONE
Uso di indici

- Senza l'uso dell'indice la strategia di iterazione semplice richiede un costo di 2 milioni di accessi per la relazione customer

- Usando l'indice e senza fare alcuna ipotesi sulla memorizzazione fisica della relazione, il join puo' essere calcolato con un numero significativamente minore di accessi

- Si devono eseguire comunque 10,000 accessi per leggere la relazione deposit

- Se $n_{customer}$ = 200 (come nel caso precedente) e ogni blocco dell'indice contine 20 puntatori, la scansione dell'indice ha un costo di 2 accessi

- Quindi e' necessario eseguire 3 accessi per ogni tuple di deposit, invece di 200; il costo totale della strategia e' di 40,000 accessi

- Sebbene il costo di 40,000 puo' sembrare alto e' da notare che nel caso della iterazione semplice si ha un costo minore solo nel caso in cui le tuple siano clusterizzate; in caso contrario puo' essere utile costruire un indice prima di eseguire il join

STIMA DEL COSTO DI ESECUZIONE
Three-way join

- Consideriamo branch |x| deposit |x| customer
  - $n_{deposit}$ = 10,000
  - $n_{customer}$ = 200
  - n_{branch}= 50

- E' necessario scegliere non solo il tipo di strategia di join (iterazione, merge-join, index) da eseguire ma anche in che ordine eseguire il join

- Strategia 1.
  - Si esegue (deposit |x| customer) usando una delle strategie viste; poiche' la c-name e' chiave di customer, il risultato di questo join e' di 10,000 tuple
  - Se si costruisce un indice sull'attributo b-name di branch si puo' calcolare
    branch |x| (deposit |x| customer)
    considerando ogni tupla t di
    (deposit |x| customer) e poi eseguendo una ricerca sull'indice con il valore t[b-name]
  - Poiche' b-name e' chiave per branch-name e' necessario esaminare una sola tupla per ognuna delle 10,000 tuple della relazione intermedia ottenuta come (deposit |x| customer)

STIMA DEL COSTO DI ESECUZIONE
Three-way join

- Strategia 2.
  Si calcola il join senza costruire alcun indice. Questo richiede esaminare 50*10000*200 possibilita', con un costo totale di 100,000,000

- Strategia 3.
  Invece di eseguire due joins in cascata, si eseguono contemporaneamente.
  - Questa tecnica richiede di costruire prima due indici
    sull'attributo b-name per branch
    sull'attributo c-name per customer
  - Quindi si considera ogni tupla t in deposit; per ogni t, si determinano le tuple in customer e le tuple in branch
  - Pertanto ogni tupla di deposit e' esaminata una volta soltanto

Questa strategia puo' essere conveniente, specialmente quando gli indici sulle relazioni sono gia' esistenti e non occorre costruirli appositamente

OTTIMIZZAZIONE - System R

Ipotesi:

Tipo delle interrogazioni

- Le interrogazioni hanno la forma seguente

  SELECT ListaDiAttributi
  FROM ListaDiRelazioni
  WHERE Condizione

  dove:

  - ListaDiAttributi indica gli attributi che interessano nel risultato (* sta per tutti gli attributi)

  - ListaDiRelazioni indica le relazioni coinvolte (per semplicita' consideriamo il caso di due relazioni)

  - Condizione e' una congiunzione di disgiunzioni di condizioni semplici del tipo

    Attributo Op Valore con *Op* ∈ { >, ≥, <, ≤, =, ≠}
    Attributo **isin** [$Valore_1$, $Valore_2$,.....,$Valore_N$]
    Attributo **within** ($Valore_1$, $Valore_2$)
    Attributo = Attributo′ con gli attributi appartenenti a relazioni diverse
    (tale predicato e' detto *predicato di join*)

<u>Tipo delle interrogazioni</u> (continua)

- Si definisce *predicato di ricerca* (o *predicato risolubile*) un predicato per il quale:
  - esiste un indice utilizzabile per trovare le tuple che soddisfano il predicato
  - l'indice e' utile per ridurre il costo della verifica della condizione

  predicati del tipo
  $C+D=1.000$ oppure $D \neq 20$ con C e D attributi con indici non sono predicati di ricerca in quanto gli indici non aiutano a limitare il numero di tuple da visitare

  Pertanto i predicati risolubili sono del tipo:
  Attributo ConIndiceOp Valore
  (con *Op* diverso da $\neq$)
  Attributo **isin** [$Valore_1$, $Valore_2$,.....,$Valore_N$]
  Attributo **within** ($Valore_1$, $Valore_2$)
  Predicato di join in cui almeno un attributo abbia un indice
- Si definisce *fattore booleano* un predicato, che se falso, rende falsa tutta l'interrogazione; tutte le tuple del risultato quindi verificano i fattori booleani.

OTTIMIZZAZIONE - System R

Tipo delle interrogazioni (continua)

- Esempio
  Relazione Impiegati(Codice, Nome, Lavoro, Salario, Obiettivo, Citta') con indici su tutti gli attributi

  Interrogazione
  SELECT Nome, Salario
  FROM Impiegati
  WHERE Lavoro = "Programmatore" AND
  (Citta'="Bologna" OR Salario >2000000)

  Il fattore booleano e' uno

Granularita' dell'ottimizzazione

L'ottimizzazione e' eseguita separatamente per ogni interrogazione, come accade in tutti i sistemi commerciali; pertanto e' bene formulare interrogazioni complesse e non scomporle in tante interrogazioni piu' semplici perche' cio' fa perdere i vantaggi dell'ottimizzazione.

OTTIMIZZAZIONE - System R

<u>Organizzazione fisica dei dati</u>

Nel recupero dei dati soddisfacenti un'interrogazione si utilizza la scansione sequenziale o si utilizzano indici allocati su singoli attributi

Gli indici per chiave primaria sono densi organizati come B+-tree

Gli indici per chiavi secondarie sono indici densi organizzati come B+-tree; le foglie contengono i valori degli attributi seguiti ciascuno dagli identificatori interni (TID) delle tuple che hanno quel valore per gli attributi

Gli indici (sia primari che secondari) si suddividono in *clustered* e *non-clustered*

Nel primo caso le tuple sono fisicamente ordinate in base ai valori dell'attributo dell'indice

Nel secondo caso le tuple non sono fisicamente ordinate in base ai valori dell'attributo dell'indice

Una pagina puo' contenere tuple di relazioni diverse

OTTIMIZZAZIONE - System R

Modello dei costi

Il costo di esecuzione di un piano di accesso e' espresso come una combinazione lineare del numero di accessi a MS, indicato con NPAG, e il numero di record NREC da esaminare in MM per generare il risultato finale

CA = NPAG + H*NREC

dove H e' il coefficiente del costo di CPU

(1/H indica il numero di confronti di tuple che sono considerati equivalenti al costo di un accesso da una pagina di disco)

Nel caso di scansione sequenziale NPAG=NPAG(R)

Nel caso di accesso con indici, NPAG e' costituito da due termini

NPAG=CI+CD

dove CI e' costo di scansione degli indici

e CD il costo dovuto all'accesso alle pagine contenenti i dati

OTTIMIZZAZIONE - System R

Statistiche

L'ottimizzatore del system R assume le seguenti ipotesi:

- uniformita' della distribuzione dei valori
- non correlazione tra valori di attributi diversi

Per ogni relazione R

- CARD(R): numero dei records in R
- NPAG(R): numero delle pagine
- NA(R): numero degli attributi della relazione R
- CARD(A): numero di valori distinti dell'attributo A

Per ogni indice I

- NLEAF(I): il numero di foglie dell'indice il cui valore e' stimato come segue

$$NLEAF(I) = [CARD(R) * L_{TID} + CARD(A) * L(A)] / [P * p_f]$$

dove:

- R e A sono rispettivamente la relazione e l'attributo su cui e' allocato l'indice
- L_{TID} e' la lunghezza di un TID
- L(A) e' la lunghezza di un valore dell'attributo
- p_f e' il fattore di riempimento di un nodo dell'indice
- P e' la dimensione delle pagine

OTTIMIZZAZIONE - System R

Statistiche (continua)

Nel System R queste statistiche sono aggiornate in seguito al caricamento delle relazioni o alla creazione di un indice

E' possibile aggiornare i valori usando il comando Update Statistics on Relazione

Le statistiche non vengono aggiornate dopo ogni singola operazione di inserzione, cancellazione, o modifica in quanto si appesantirebbe il costo delle operazioni di aggiornamento

Algoritmo di ottimizzazione
L'algoritmo usa al piu' un indice per relazione

Interrogazione su una singola relazione- passi
- selezione di un predicato di ricerca
- controllo della condizione sui dati

Il predicato di ricerca scelto deve essere un fattore booleano

OTTIMIZZAZIONE - System R
Selezione di un predicato di ricerca

Nel corso di questo passo occorre:
- analizzare la condizione per ricavarne tutti i fattori booleani
- individuare i predicati di ricerca fra i fattori booleani
- valutare il costo di accesso con l'impiego dell'indice, per ogni predicato di ricerca
- scegliere il predicato di ricerca piu' conveniente, cioe' quello con costo minimo

Per eseguire questo passo, la condizione viene rappresentata come un albero in cui

- i nodi terminali sono associati alle condizioni semplici

- gli altri nodi agli operatori AND e OR

- i nodi associati ai predicati sono marcati con l'etichetta R se si tratta di predicati di ricerca

- altrimenti sono marcati con N

OTTIMIZZAZIONE - System R
Selezione di un predicato di ricerca

(a) Si costruisce l'insieme LP dei predicati risolubili applicando il seguente algoritmo ricorsivo a partire dalla radice dell'albero

1) se il nodo e' un predicato di tipo R, allora LP=[predicato]

2) se il nodo e' un predicato di tipo N, allora LP=[]

3) se il nodo e' AND si applica l'algoritmo ricorsivamente ai sottoalberi sinistro e destro, producendo due insiemi che uniti danno l'insieme LP associato al nodo

4) se il nodo e' OR si procede come per il nodo AND

Alla fine del passo (a) alla radice dell'albero e' associato l'insieme dei predicati per i quali esiste un indice utile per l'ottimizzazione della query

OTTIMIZZAZIONE - System R
Selezione di un predicato di ricerca

(b) Si eliminano dall'insieme LP tutti i predicati che hanno un antenato OR, producendo l'insieme LPA dei predicati che sono anche fattori booleani

A questo punto si possono verificare tre casi

- LPA e' vuoto: non esistono indici da usare per limitare il numero di tuple da visitare. Si marca la radice dell'albero con l'etichetta N (non risolto)

- LPA contiene un solo predicato: esiste allora un indice per fare la restrizione. Si marca la radice con S (semi-risolto), oppure con R (risolto) se tutta la condizione coincide con il predicato

- LPA contiene piu' predicati: occorre allora scegliere il predicato piu' conveniente, stimando per ognuno di essi il costo di accesso CA=CI+CD. Una volta selezionato il predicato di ricerca con costo minimo, la radice viene marcata con S.

OTTIMIZZAZIONE - System R
Selezione di un predicato di ricerca

Per stimare il costo di accesso associato ad un predicato di ricerca, si stima il fattore della selettivita' dell condizione (indicato con F(condizione)): cioe' la stima della frazione di tuple che soddisfano la condizione, rispetto al numero totale di tuple presenti nella relazione

Assumendo l'uniformita' di distribuzione dei valori di ogni attributo, il fattore di selettivita' si stima come segue:

- F(A=Valore) = 1/CARD(A)

  (se per qualche motivo CARD(A) non e' noto, il sistema assume F=1/10)
- F(A>Valore) = (max(A) - Valore)/(max(A)-min(A))

  F(A<Valore) = (Valore-min(A))/(max(A)-min(A))

  se si conoscono i valori di max e min dell'attributo e l'attributo e' numerico. Se tali informazioni non sono note, o l'attributo non e' di numerico, si stima F=1/3

  Non c'e' alcuna significativita' in questo numero a parte il fatto che e' maggiore di 1/10 e minore di 1/2 (i progettisti del System R assumono che poche queries che hanno predicati di selezione sono soddisfatte da piu' della meta' delle tuple)

OTTIMIZZAZIONE - System R
Selezione di un predicato di ricerca

- F(A **isin** [$Valore_1$, $Valore_2$,.....,$Valore_N$])=
  = N * F(A=Valore)
  F(A **within** ($Valore_1$, $Valore_2$))=
  = ($Valore_2$ -$Valore_1$)/(max(A)-min(A))
  se si conoscono i valori di max e min dell'attributo sono noti e l'attributo e' numerico. Se tali informazioni non sono note, o l'attributo non e' numerico si considera F=1/4
- F(C_1 AND C_2) = F(C_1) *F(C_2)
  F(C_1 OR C_2) = F(C_1) +F(C_2) - F(C_1) *F(C_2)

Per come e' stato definito, e' chiaro che tanto piu' il fattore di selettivita' e' minore di uno, tanto piu' e' selettivo il predicato a cui si riferisce

Una volta calcolato il fattore di selettivita' del predicato, occorre calcolare il costo di accesso all'indice e il costo di accesso ai dati

Indicato con C_A un predicato di ricerca, e dato F(C_A) fattore di selettivita' di tale predicato, il costo di accesso all'indice I allocato su A, e' dato da:
CI = F(C_A) * NLEAF(I)
(quindi F(C_A) da' anche la stima della frazione di foglie accedute)

OTTIMIZZAZIONE - System R
Selezione di un predicato di ricerca

- La stima del costo di accesso ai dati e' piu' complessa e dipende dalla proprieta' dell'indice di essere clustering oppure no

OTTIMIZZAZIONE - System R
Selezione di un predicato di ricerca

- Stima del costo di accesso ai dati:
  - clustered index
    $CD = F(C_A) * NPAG$
  - non-clustered index
    In questo caso poiche' la lista di riferimenti e' ordinata, non si puo' verificare il caso che per un valore dell'attributo A una pagina venga visitata piu' volte

    Pero' si puo' verificare che una stessa pagina venga visitata piu' volte per valori dell'attributo compresi in un certo intervallo (piu' liste da visitare). Pertanto
    CD = N.Liste di TIDs * N.Pagine da visitare per ogni lista
    N.Pagine da visitare per ogni lista =
    Φ(NTK(A), NPAG, CARD(R))

    (Φ e' la funzione di Yao)

$$\Phi = NPAG * \left[1 - \prod_{i=1}^{NTK(A)} \frac{CARD(R) - (CARD(R)/NPAG) - i + 1}{CARD(R) - i + 1}\right]$$

OTTIMIZZAZIONE - System R
Selezione di un predicato di ricerca

- Stima del costo di accesso ai dati (continua):

CD= $F(C_A)$ * CARD(A) * $\Phi($NTK(A), NPAG, CARD(R))

dove NTK(A) e' il numero medio di TID associati ad un valore dell'attributo; e' ottenuto come:

CARD(R)/CARD(A)

Pertanto, il costo globale di accesso e' dato dalle seguenti espressioni:

- CA = $F(C_A)$* [NLEAF(I) + NPAG(R)]
  se l'indice e' clusterizzato

- CA = $F(C_A)$* [NLEAF(I) +
  CARD(A) * $\Phi($NTK(A), NPAG, CARD(R))]
  se l'indice e' non-clusterizzato

OTTIMIZZAZIONE - System R
Selezione di un predicato di ricerca

<u>Osservazione</u>

- Se si usa un indice per accedere alle tuple della relazione, le tuple vengono reperite e restituite nell'ordine dei valori dell'attributo su cui e' costruito l'indice

- Spesso pero' e' richiesto che le tuple siano in un ordine diverso; per esempio l'utente ha specificato una clausola ORDER BY o GROUP BY

- Pertanto, prima di restituire le tuple all'utente, e' necessario ordinarle se non lo sono gia'

- In questo caso e' opportuno distinguere tra vie di accesso (indici o scansione sequenziale) che producono l'ordinamento richiesto e vie di accesso che non lo producono

- Nel secondo caso e' quindi necessario calcolare anche il costo di ordinamento della relazione risultato

OTTIMIZZAZIONE - System R
Selezione di un predicato di ricerca

Stima del risultato

- La dimensione di tale relazione e' calcolata in base a:
  - la dimensione di una tupla risultato (si ottiene facendo la somma delle dimensioni degli attributi presenti nella *ListaDiAttributi*)

  - il numero delle tuple risultanti CRES $CRES = CARD(R) * F_1 * F_2 * \ldots\ldots\ldots * F_n$ dove F_1, F_2, ..., F_n sono le selettivita' di tutti i congiunti della interrogazione che si riesce a stimare

- In base alla stima della dimensione della relazione intermedia, e' possibile stimare il numero di pagine per contenere il risultato e quindi il costo di ordinamento

- Pertanto un indice clusterizzato e' di solito da preferire ad uno non clusterizzato, a meno che il predicato risolubile con l'indice non clusterizzato e' molto selettivo oppure le tuple sono richieste ordinate in base ai valori dell'attributo su cui e' allocato l'indice non clusterizzato

OTTIMIZZAZIONE - System R
Controllo della condizione sui dati

- Una volta calcolati i costi di ciascun predicato e selezionato quello con costo minimo, si esegue il passo successivo nell'esecuzione dell'interrogazione

- Il procedimento di controllo dipende da come e' marcata la radice dell'albero

- Se l'etichetta e' N, allora si procede con la scansione sequenziale della relazione e si estraggono le tuple che verificano la condizione

- Se l'etichetta e' S, si usa l'indice associato al predicato (scelto al passo precedente) solo se il suo costo e' inferiore a quello della scansione sequenziale, stimato pari a NPAG (numero delle pagine cha hanno tuple della relazione); in caso contrario si procede con la scansione sequenziale

- Se l'etichetta e' R, si procede come al caso precedente, la differenza e' che, se si usa l'indice, i riferimenti sono solo quelli delle tuple che soddisfano l'interrogazione

OTTIMIZZAZIONE - System R
Controllo della condizione sui dati

Conclusione

Per quanto riguarda il costo del metodo si hanno due stime:

- CA=NPAG se l'etichetta della radice e' N
- se l'etichetta della radice e' S o R, supponendo di scartare la scansione sequenziale, e trascurando il costo di ordinamento esistono due sottocasi:

  - $CA = F(C_A)^* [NLEAF(I) + NPAG(R)]$
    se l'indice selezionato e' clusterizzato

  - $CA = F(C_A)^* [NLEAF(I) + CARD(A) * \Phi(NTK(A), NPAG, CARD(R))]$
    se l'indice selezionato e' non-clusterizzato

OTTIMIZZAZIONE - System R
Esempio

Si consideri la relazione
Impiegato(Codice, Nome, Qualifica, Stipendio, Progetto, Citta')

supponendo che:

- la relazione sia ordinata su Stipendio, e sia costituita da 2.000 tuple, contenute in 100 pagine, ciascuna delle quali contiene 20 tuple
  CARD(Impiegati)=2000
  NPAG=100

- esista un indice I clusterizzato su Stipendio
  CARD(Stipendio)=200
  NLEAF(I') = 9
  Min(Stipendio) = 1.200.000
  Max(Stipendio)=2.700.000

- esista un indice I' non clusterizzato su Qualifica
  CARD(Qualifica)=40
  NLEAF(I') = 7

OTTIMIZZAZIONE - System R
Esempio

Si vuole selezionare il Nome e lo Stipendio dei programmatori che guadagnano tra 1.500.000 e 2.000.000, e cha lavorano al progetto "Banche" o "Comuni"

- Questa interrogazione e' formulata come segue
  SELECT Nome, Stipendio
  FROM Impiegati
  WHERE Qualifica = "Programmatore" AND
  Stipendio within (1.500.000, 2.000.000) AND
  Progetto isin ["Banche"; "Comuni"]

Albero della condizione

AND LP=[Qualifica, Stipendio]

R AND LP=[Stipendio]

Qualifica="Programmatore"

R N

Stipendio within (1.500.000, 2.000.000) Progetto isin["Banche", "Comuni"]

OTTIMIZZAZIONE - System R
Esempio

- Si ha che LPA=[Qualifica, Stipendio]

- Strategie alternative
  a) scansione sequenziale
  b) accesso con indice I (su Stipendio)
  c) accesso con indice I' (su Qualifica)

- I fattori di selettivita' sono:
  F(Qualifica = "Programmatore") = 1/40=0.025
  F(Stipendio within (1.500.000, 2.000.000)) = 0.33

  NTK(Qualifica) = CARD(Impiegati)/
  /CARD(Qualifica) = 2.000/40=50

- Quindi i costi per le tre alternative sono:
  a) $CA_{sequenziale}$= 100
  b) $CA_{Qualifica}$= 0.025*[7 + 40 *Φ(50,100, 2000)]=40
  c) $CA_{Stipendio}$= 0.33 *(9 + 100) = 36

- Viene quindi preferito l'indice su Stipendio

- In questo caso si nota come un predicato poco selettivo possa dar luogo ad un costo di scansione inferiore attraverso indice inferiore a quello prodotto usando non indice non clusterizzato

OTTIMIZZAZIONE - System R
Esempio

- Supponiamo di avere anche un indice I'' non clusterizzato su Progetto
  CARD(Progetto) = 300 e NLEAF(I'') = 16
  si ha
  F(Progetto isin ["Banche", "Comuni"])=2/300=0.0066

  NTK(Progetto)=2000/300 = 7 da cui
  $CA_{Progetto}$= 0.0066 * [16 + 300 *Φ(7,100, 2000)] =
  1 + 14=15

- In questo caso l'indice su Progetto e' da preferire a quello su Stipendio

OTTIMIZZAZIONE - System R
Joins

- Il System R usa come metodi di join sia il nested-loop (iterazione) che il merge-join

- Inoltre vengono considerate tutte le possibili permutazioni quando si deve eseguire il join di piu' di due relazioni

- Per restringere lo spazio della ricerca non vengono considerate permutazioni che implicano dei prodotti cartesiani: in altra parole si cerca di eseguirli il piu' tardi possibile

  Per esempio:
  date R1(a,b) R2 (d,e,f), R3(g,h) relazioni e una query della forma
  select R1.a, R2.f, R3.h **from** R1, R2, R3 **where**
  R1.b=R2.d AND R2.e=R3.g

  esiste un join tra R1 e R2 e un join tra R2 e R3
  il primo join e' su un attributo di R2 diverso dall'attributo su cui e' definito il secondo join

  Le permutazioni che non vengono considerate sono:
  (R1-R3)-R2 (R3-R1)-R2

OTTIMIZZAZIONE - System R
Joins

- Per determinare la strategia ottima per il join di n relazioni, viene generato un albero delle possibili strategie

- L'approccio seguito consiste nel determinare una soluzione ottima per ogni sottoinsieme dei join della query, e poi determinando il modo migliore per eseguire il join della relazione composita ottenuta con una ulteriore relazione

- Questo approccio parte dal considerare i join di due relazioni ed ogni passo incrementa il numero di relazioni

- Una soluzione consiste di:
  - una lista ordinata di relazioni su cui si esegue il join
  - il metodo di join usato
    (nested loop, merge join)
  - un piano indicante come ogni relazione deve essere acceduta
    (indice, scansione sequenziale)
  - indicazione sulla necessita' di ordinamento delle relazioni prima dell'esecuzione del join

OTTIMIZZAZIONE - System R
Joins

- Un aspetto importante riguarda se le relazioni devono essere ordinate. Si definiscono ordinamenti *interessanti*, gli ordinamenti eseguiti su attributi che compaiono in clausole di Group by e di Order by, e sugli attributi di join

- Infatti in alcuni casi, puo' essere preferibile scegliere un indice non clustered rispetto ad un indice clustered, se il primo ritrova le tuple secondo un ordinamento interessante

- L'albero di ricerca e' costruito tramite iterazione sul numero di relazioni "joined"

- Al primo passo, si determina il modo piu' efficiente per accedere ogni singola relazione per ogni ordinamento interessante e per il caso non ordinato

- Al secondo passo, si determina il modo migliore per eseguire il join di ogni singola relazione con ogni altra singola relazione (alcune combinazioni non sono valutate in base all'euristica di ritardare l'esecuzione del prodotto cartesiano)

OTTIMIZZAZIONE - System R
Joins

- Il secondo passo produce soluzioni per eseguire i joins di coppie di relazioni

- Al terzo passo si determina il modo migliore per eseguire i joins di tre relazioni

- Il terzo passo viene eseguito considerando ogni relazione composita, ottenuta al passo precedente, e determinando il modo migliore per collegarla con ogni altra relazione (soggetta alla euristica)

- Dopo che tutte le soluzioni complete sono determinate (il join di tutte le relazioni e' eseguito), l'ottimizzatore sceglie la soluzione meno costosa che restituisce le tuple nell'ordine richiesto (se e' stato specificato un ordine)

- Notare che se esiste una soluzione che da' l'ordinamento richiesto, non deve essere eseguito alcuno ordinamento extra per le clausole di Order by e Group by, a meno che la soluzione ordinata sia piu' costosa della soluzione piu' economica non ordinata con il costo addizionale dell'esecuzione dell'ordinamento

OTTIMIZZAZIONE - System R
Joins

- Il numero di soluzioni da memorizzare e' al piu' 2**n (il numero di possibili sottoinsiemi di n relazioni) moltiplicato il numero di ordinamenti interessanti

- Il tempo necessario per generare l'albero e' proporzionale a tale numero

- In realta' questo numero e' spesso ridotto usando l'euristica sul prodotto cartesiano

- L'esperienza indica che il tempo di CPU (su 370/168) necessario per eseguire l'ottimizzazione e' dell'ordine di poche decine di secondi

- Formule di stima della dimensione delle relazioni intermedie
  Il System R usa delle formule piu' semplici rispetto a quelle viste precedentemente

OTTIMIZZAZIONE - System R
Joins - esempio

OTTIMIZZAZIONE - System R
Joins - esempio

Indici:

relazione EMP: indice su DNO, indice su JOB
relazione DEPT: indice su DNO
relazione JOB: indice su JOB

Passo 1

Si determina il modo piu' efficiente per eseguire l'accesso ad ogni singola relazione

Relazione EMP - tre possibili strategie
a) indice su DNO
b) indice su JOB
c) scansione sequenziale
Gli ordinamenti interessanti sono DNO, JOB

Supponiamo che l'indice su JOB sia la strategia di accesso piu' efficiente

Questo permette di eliminare la scansione sequenziale; la strategia (a) viene comunque mantenuta perche' rappresenta un ordinamento interessante

OTTIMIZZAZIONE - System R
Joins - esempio

Passo 1

Relazione DEPT- due possibili strategie
a) indice su DNO
b) scansione sequenziale

Supponiamo che l'accesso basato sull'indice su DNO sia la strategia piu' efficiente

Questo permette di eliminare la scansione sequenziale

Relazione JOB - due possibili strategie
a) indice su JOB
b) scansione sequenziale

Supponiamo che la scansione sequenziale sia piu' efficiente

In questo caso la strategia (a) viene mantenuta in quanto da' un ordinamento efficiente

OTTIMIZZAZIONE - System R
Joins - esempio

Sommario del passo 1

OTTIMIZZAZIONE - System R
Joins - esempio

Risultati del passo 1

(nella figura C<set of relations> (path) e' usato per rappresentare il costo di eseguire la scansione della relazione in base al cammino di accesso indicato da path;
Ni indica le cardinalita' delle relazioni intermedie)

OTTIMIZZAZIONE - System R
Joins - esempio

Passo 2

Si determinano soluzioni per tutte le possibili coppie di relazioni

Si esegue il join di una seconda relazione con ogni singola relazione nell'albero di ricerca ottenuto al passo 1

Si considerano prima tutte le strategie basate sul nested loop

In questo esempio si assume che e' piu' economico eseguire il join EMP-JOB usando l'indice su JOB

(infatti data una tupla di EMP possiamo trovare subito la tupla di JOB che ha lo stesso valore per l'attributo JOB)

OTTIMIZZAZIONE - System R
Joins - esempio

Passo 2

Albero di ricerca strategie nested-loop

OTTIMIZZAZIONE - System R
Joins - esempio

Passo 2

- Si determinano tutte le soluzioni basate sul merge-join

- Poiche' esiste una cammino di accesso che ordina EMP in base al DNO e DEPT in base al DNO e' possibile eseguire un merge join tra queste relazioni, senza eseguire alcun ordinamento

- Notare che un'altra possibilita' e' di accedere EMP in base al indice sull'attributo JOB, eseguire l'ordinamento su DNO, e quindi eseguire il merge

- Per eseguire il merge di JOB con EMP, consideriamo solo l'indice sulla colonna JOB di EMP poiche' e' la strategia di accesso piu' efficiente, indipendentemente dall'ordine

- Usando l'indice su JOB per la relazione JOB e' possibile eseguire il merge senza alcun ordinamento preliminare

- Un'alternativa e' di eseguire una scansione sequenziale di JOB e poi eseguirne l'ordinamento (quindi non si usa l'indice sulla relazione JOB)

OTTIMIZZAZIONE - System R
Joins - esempio

Passo 2

Albero di ricerca strategie merge-join

OTTIMIZZAZIONE - System R
Joins - esempio

<u>Passo 2</u>

Una volta calcolati i costi di tutte le strategie sia nel caso nested-loop che nel caso merge-join, viene fatto un confronto per determinare per ogni coppia di relazioni la *soluzione piu' economica equivalente* (due soluzioni sono equivalenti se hanno le stesse tabelle e l'ordine delle tuple nel risultato e' lo stesso)

Risultato del pruning

OTTIMIZZAZIONE - System R
Joins - esempio

Passo 3

Si determina la soluzione per eseguire il join di tre relazioni

Le soluzioni possibili sono determinate eseguendo il join di una terza relazione con le relazioni intermedie (ottenute come join di due relazioni alla fine del passo 2)

Come al passo 2, si determinano prima le strategie nested-loop e poi quelle merge-join

Supponiamo che C5U e' un costo minore di C5D e C5J (se non e' cosi' la strategia basata sulla scansione sequenziale puo' essere eliminata)

Supponiamo inoltre che per eseguire il join di JOB con il join ottenuto da EMP e DEPT sia preferibile usare l'indice su JOB per la relazione JOB

OTTIMIZZAZIONE - System R
Joins - esempio

Passo 3

Albero di ricerca strategie nested-loop