Basi di dati II

Esercizi di autovalutazione — 16 marzo 2012
Cenni sulle soluzioni

Domanda 1 Si considerino un sistema con blocchi di dimensione $B = 1000$ byte e puntatori ai blocchi di $p = 5$ byte e una base di dati sulle seguenti relazioni, ognuna delle quali (i) ha una struttura heap (ii) ha un indice secondario sulla chiave (iii) non ammette valori nulli

- $R_1(\underline{A}BC)$, contenente $S_1 = 1.000.000$ ennuple di $l_1 = 40$ byte, di cui $l_A = 5$ per il campo chiave A
- $R_2(\underline{D}EF)$, contenente $S_2 = 400.000$ ennuple di $l_2 = 100$ byte, di cui $l_D = 4$ per il campo chiave D
- $R_3(\underline{G}HL)$, contenente $S_3 = 500.000$ ennuple di $l_3 = 25$ byte, di cui $l_G = 5$ per il campo chiave G

e con una vista definita come segue:

- `CREATE VIEW V AS SELECT * FROM (R1 LEFT JOIN R2 ON B=D) JOIN R3 ON C=G`

In tale contesto,

- mostrare un possibile piano di esecuzione (in termini di operatori dell'algebra relazionale e loro realizzazioni) per ciascuna delle seguenti interrogazioni
  1. `SELECT A, L FROM V`
  2. `SELECT A FROM V`
  3. `SELECT A, E FROM V`
- stimare il costo, in termini di numero di accessi a memoria secondaria (ignorando la presenza di eventuali buffer) per l'operazione 1.

Discussione (si suggerisce di verificare il comportamento, almeno qualitativo, su un DBMS)
Si ricorda che la vista

- `CREATE VIEW V AS SELECT * FROM (R1 LEFT JOIN R2 ON B=D) JOIN R3 ON C=G`

non viene necessariamente calcolata tutta: il processo di ottimizzazione viene applicato alla espressione ottenuta sostituendo la definizione della vista ad ogni sua occorrenza.

1.`SELECT A, L FROM V`

- poiché il join su R_2 è esterno, esso non influisce sul risultato e viene omesso; quindi viene eseguito solo il join $R_1 \bowtie_{C=G} R_3$ (si può ragionare sulle proiezioni per ridurre i risultati intermedi, ma non è importante, perché non è detto che vi siano passi intermedi)
- per quanto riguarda i costi, si può pensare ad un nested-loop con accesso diretto alla tabella interna e quindi, ignorando i buffer, si ha un numero di accessi a blocchi stimabile come segue (dove LIV_3 è il numero di livelli dell'indice di R_3 e s è la frazione di valori di C che compare in $R_3.G$, che approssimiamo a 1, in assenza di informazioni):

  $$S_1/(B/l_1) + S_1(\text{LIV}_3 + s) = \text{ ca } 1.000.000/(1.000/40) + 1.000.000(3+1) = \text{ ca } 4.000.000$$

  in presenza di buffer probabilmente un paio di livelli dell'indice possono essere trascurati

  $$S_1/(B/l_1) + S_1(\text{LIV}_3 - 2 + s) = \text{ ca } 1.000.000/(1.000/40) + 1.000.000(1+1) = \text{ ca } 2.000.000$$

  trattandosi di tabelle grandi, probabilmente convengono altri algoritmi che possono sfruttare meglio i buffer: con 200 buffer entrambe le relazioni possono essere ordinate in due passate e quindi si può pensare ad un mergejoin che sostanzialmente scandeisce ciascuna relazione tre volte (e la riscrive due, per salvare i risultati intermedi):

  $$5(S_1/(B/l_1) + S_2(B/l_2)) = \text{ ca } 5(40.000 + 40.000) = \text{ ca } 400.000$$

  Un hashjoin avrebbe un costo paragonabile

2. `SELECT A FROM V`

- per gli stessi motivi del caso precedente, il join su R_2 può essere omesso; invece quello su R_3 è necessario (per verificare l'appartenenza delle ennuple al risultato); non sarebbe stato necessario se ci fosse stato il vincolo di integrità referenziale fra C di R_1 ed R_3; (nota, per curiosità: in quust'ultimo caso basterebbe una scansione dell'indice di R_3, senza accesso ai blocchi di R_3 stessa);

3. `SELECT A, E FROM V`

- servono entrambi i join.

Estensione: che cosa succede se è definito un vincolo di integrità referenziale fra F di R_2 e la chiave G di R_3?
I join con R_3 servono solo se debbono essere restituiti valori dei suoi attributi.

Domanda 2 Siano r_1 ed r_2 due relazioni contenenti rispettivamente N_1 e N_2 ennuple, con fattore di blocco rispettivamente F_1 e F_2. Si supponga che il sistema abbia a disposizione un buffer di dimensione pari a $M = 101$ blocchi. Calcolare il numero di accessi a memoria secondaria necessario per eseguire un join $r_1 \bowtie_{A_1=A_2} r_2$ (con A_1 attributo di r_1 e A_2 attributo di r_2), nei seguenti casi, da considerare separatamente l'uno dall'altro e assumendo che il DBMS sia in grado di eseguire il join solo con il metodo *nested-loop* (eventualmente utilizzando l'accesso diretto tramite un indice invece della scansione interna) e che utilizzi solo strutture primarie disordinate. Si supponga infine che il blocco abbia dimensioni $B = 1$ Kbyte, che i puntatori occupino $p = 4$ byte e i valori dei due attributi in questione occupino $k = 20$ byte ciascuno.

1. $N_1 = 100.000$ e $N_2 = 1.000.000$, con $F_1 = F_2 = 10$; A_2 è la chiave di r_2 mentre i valori di A_1 in r_1 si ripetono mediamente in $e = 6$ ennuple; non vi sono indici
2. $N_1 = 100.000$ e $N_2 = 1.000.000$, con $F_1 = F_2 = 10$; A_1 è la chiave di r_1 mentre i valori di A_2 in r_2 si ripetono mediamente in $e = 6$ ennuple; sono definiti un indice su $r_1(A_1)$ e uno su $r_2(A_2)$
3. $N_1 = 100.000$ e $N_2 = 1.000.000$, con $F_1 = F_2 = 10$; gli attributi coinvolti non sono chiave e hanno, ciascuno, valori che si ripetono mediamente in $e = 6$ ennuple; è definito un indice su $r_1(A_1)$
4. $N_1 = 5.000$ e $N_2 = 1.000.000$, con $F_1 = F_2 = 20$; A_1 è la chiave di r_1 e A_2 è la chiave di r_2; sono definiti un indice su $r_1(A_1)$ e uno su $r_2(A_2)$

Discussione (molto schematica e a ovvio rischio di imprecisioni)
Indichiamo con $B_1 = N_1/F_1$ e $B_2 = N_2/F_2$ il numero di blocchi delle due relazioni.

1. Non esistendo indici (e supponendo che il sistema non li costruisca appositamente per l'interrogazione), l'unica soluzione è una doppia scansione; con M blocchi nel buffer, se ne possono dedicare $M-1$ alla relazione esterna (cioè quella considerata nel ciclo esterno) e 1 a quella interna, al fine di ridurre il numero di scansioni di quest'ultima. In questo modo, scegliendo r_1 come esterna (perché più piccola) servono

   - B_1 accessi per la scansione relazione esterna
   - $B_2 \times B_1/(M-1)$ accessi per le $M-1$ scansioni della relazione interna

   In totale: poiché $B_1 = N_1/F_1 = 10.000$ e $B_2 = N_2/F_2 = 100.000$ e $M - 1 = 100$ abbiamo più di 10.000.000 accessi.
   Si potrebbe notare che, essendo coinvolta la chiave di r_2, si potrebbero risparmiare (mediamente, nell'ipotesi di valori in r_1 presenti anche in r_2) metà degli accessi.

2. Esistendo due indici, la scelta dovrebbe ricadere su quello che fa risparmiare di più. Di solito, per questo motivo conviene scegliere la relazione più piccola come esterna. Quindi il costo potrebbe essere

   - $B_1 + N_1 \times (I_2 + e)$ dove I_2 è la profondità dell'indice di r_2 e il termine e viene aggiunto perché i dati con stesso valore dell'attributo di join si trovano in blocchi diversi; tenendo conto dei buffer, I_2 (che è pari a 4, perché il fan-out dell'indice è circa 40 e quindi si hanno 40 record dell'indice al primo livello, 1600 al secondo 64.000 al terzo e milioni al quarto) può essere ridotto di 2, perché due livelli dell'indice entrano sicuramente nel buffer (il fan-out dell'indice è circa 40); il tutto (considerando quindi I_2 pari a 2) potrebbe portare a circa 800.000 accessi
   - vista la presenza dell'addendo e, può valere la pena considerare anche l'alternativa in cui r_2 è esterna, in cui il costo è $B_2 + N_2 \times (I_1 + 1)$, circa 3.000.000 quindi maggiore (ma si deve sempre ricordare che queste valutazioni sono molto approssimate)

3. simile al precedente, ma con una sola alternativa: r_2 esterna e costo $B_2 + N_2 \times (I_1 + e)$, circa 8.000.000

4. la differenza di dimensioni, vista la parità di tutte le altre condizioni, porta a preferire la scelta di r_1 come esterna: $B_1 + N_1 \times (I_2 + 1)$, pari circa a 15.000

Domanda 3 Alcuni DBMS prevedono la possibilità di includere in un indice i valori di altri attributi delle ennuple, oltre a quelli degli attributi su cui l'indice è realizzato. Ad esempio, una istruzione del tipo

```
CREATE INDEX contoCorrenteIX ON contoCorrente (numero) INCLUDE (saldo)
```

crea un indice sulla relazione `contoCorrente` includendo nelle foglie dell'indice, oltre ai valori di `numero` (su cui l'indice è realizzato), anche quelli di `saldo`. Confrontare (con riferimento ad esempio a specifiche operazioni di ricerca e di aggiornamento che potrebbero essere avvantaggiate o penalizzate) questa soluzione con le due seguenti

```
CREATE INDEX contoCorrenteIX ON contoCorrente (numero)
CREATE INDEX contoCorrenteIX ON contoCorrente (numero, saldo)
```

Discussione Si richiamano le tre formulazioni

1. `CREATE INDEX contoCorrenteIX ON contoCorrente (numero) INCLUDE (saldo)`
2. `CREATE INDEX contoCorrenteIX ON contoCorrente (numero)`
3. `CREATE INDEX contoCorrenteIX ON contoCorrente (numero, saldo)`

La (1) favorisce le operazioni che prevedano ricerche su `numero` con accesso ai valori di `saldo`; se però sono necessari valori di altri attributi, non si ha nessun beneficio. Poiché `numero` è chiave, la soluzione (3) è sostanzialmente equivalente, da questo punto di vista, a parte piccole differenze sull'aggiornamento. Comunque, la soluzione (3) può essere vista come soluzione di "ripiego," nel caso in cui non sia disponibile la (1). La soluzione (2) favorisce l'accesso diretto su `numero` ma richiede l'accesso al file per il recupero dei valori sugli altri campi. Le soluzioni (1) e (3) penalizzano gli aggiornamenti (la (3) un po' di più, anche se non moltissimo, perché `numero` è chiave; se non lo fosse, la differenza sarebbe maggiore).