Esercizio 10.1 Calcolare il fattore di blocco e il numero di blocchi occupati da una relazione con $T = 1000000)$ di tuple di lunghezza fissa pari a $L = 200$ byte in un sistema con blocchi di dimensione pari a $B = 2$ kilobyte.

Soluzione

Dimensioni tabella (D_T):
$D_T = T * L$
$D_T = 1000000 * 200 = 200000000$ byte $= 190.73$ MB

Numero blocchi:
$N_B = D_T/B$
$N_B = 200000000/2048 = 97.66$ blocchi

Fattore di blocco:
$F_B = B/L$
$F_B = 2048/200 = 10.24$ record / blocco

Esercizio 10.2

Si consideri una relazione IMPIEGATO(Matricola, Cognome, Nome, DataNascita) con un numero di tuple pari a N abbastanza stabile nel tempo e una dimensione di ciascuna tupla (a lunghezza fissa) pari a L byte, di cui K per la chiave.
Supporre di avere a disposizione un DBMS che permetta strutture fisiche disordinate (heap) e hash e che preveda la possibilità di definire indici secondari e operi su un sistema operativo che utilizza blocchi di dimensione B e con puntatori ai blocchi di P caratteri.
Le operazioni principali siano le seguenti:

1. ricerca esatta sul numero di matricola con frequenza f1
2. ricerca sul cognome (anche su sottostringa iniziale) con frequenza f2; mediamente una richiesta restituisce 4 record.

Individuare alcune (almeno una) possibili organizzazioni fisiche per tale relazione e calcolare (approssimativamente) il numero di accessi a memoria secondaria (nell'unità di tempo) supponendo N = 5.000.000 tuple, L = 125 byte, K = 5 byte, B = 1.000 byte, P = 4 byte, f1 = 100 volte al minuto, f2 = 1.000 volte al minuto.

Soluzione:

Una prima organizzazione fisica per tale relazione è una struttura hash sulla matricola ed un indice secondario (un albero B+) sull'attributo cognome.
Per fare delle verifiche, dobbiamo fare delle ipotesi sulla dimensione dell'attributo cognome, che supponiamo pari a 20 caratteri.
Il nostro F per l'albero B+ sarà quindi 1.000/(20+4) = 41.6 arrotondato a 41. La profondità del nostro albero per N = 5.000.000 sarà pari a 5 livelli.
Analizzando il numero di accessi per le interrogazioni ottengo che:
1. l'operazione 1 ha un costo di 1 accesso in memoria per 100 volte, pari a 100 accessi in memoria.
2. l'operazione 2 ha un costo di 5 accessi in memoria per trovare una corrispondenza, alla quale vanno aggiunti 4 accessi (scorrendo l'albero) per ottenere i quattro record forniti mediamente. Moltiplicando quindi + accessi per una ricerca per la frequenza f2 di 1.000 ottengo un totale di 9.000 accessi in memoria.

Questa organizzazione ha un costo di 9.000 + 100 = 9.100 accessi

Una seconda realizzazione fisica per questa relazione consiste in un albero B+ sulle matricole e un indice primario sui cognomi.
Il nostro F per l'albero B+ sarà quindi 1.000/(5+4) = 11.11 arrotondato a 11. La profondità del nostro albero per N = 5.000.000 sarà pari a 4 livelli.
Avendo un albero B+ di 4 livelli organizzato sulla matricola, l'operazione 1 costerebbe 100 x 5 accessi (4 accessi + quello finale), ossia 500 accessi.
L'operazione 2, sapendo che un blocco contiene 8 tuple e che una lettura media ha un costo di 1,5 accessi in memoria (in quanto una semplice analisi dimostra che i dati possono essere con uguale probabilità su uno o su due blocchi), ha quindi un costo di: f2*(costo medio + numero di risultati) = accessi in memoria, pari a 1000 * (1,5 + 4) = 5.500
In totale la soluzione 2 ha un costo di 6.000 accessi in memoria.

La soluzione più efficiente è quindi la seconda.

Esercizio 10.3 Si considerino:

- un sistema con blocchi di $B = 1000$ byte e indirizzi ai blocchi di $p = 2$ byte; il sistema (in effetti un po' obsoleto) prevede indici, primari o secondari, a un solo livello e solo sul campo chiave;
- una relazione con $N = 1000000$ tuple, ciascuna di $l = 100$ byte, di cui $k = 8$ byte per la chiave.

Calcolare:

1. il numero dei blocchi necessari per un indice primario sul campo chiave;
2. il numero dei blocchi necessari per un indice secondario;
3. il numero di accessi a memoria secondaria necessari sequenziale sulla chiave;
4. il numero di accessi a memoria secondaria necessari utilizzi un indice primario;
5. il numero di accessi a memoria secondaria necessari utilizzi un indice secondario.

Calcolare il numero di accessi sia nel caso peggiore sia medio, nell'ipotesi che le ricerche abbiano successo in casi (cioè otto volte su dieci il record cercato esiste).

Soluzione

1. Dimensione record indice primario:
   | valore chiave | valore ind memoria | $KP = K + P$
   numero di blocchi (e non di record poiché la struttura è ordinata) a cui deve puntare l'indice primario (sparso):

   $NB = (K * N)/B = 8 * 1000000/1000 = 8000$ blocchi
   dimensione indice = $(K + P) * NB = (8 + 2) * 8000 = 80000$byte
2. Dimensione record indice secondario
   | valore chiave | valore ind memoria | $KP = K + P$
   numero di record (e non di blocchi perch la struttura non è ordinata) a cui deve puntare l'indice secondario (denso) pari a l
   dimensione indice $= l * (p + k) = 1000000 * 10 = 10000000$ byte
3. Supponendo l'assenza di buffer, il numero di accessi alla memoria secondaria necessari per un FULLSCAN della tabella sono pari, nel caso peggiore, al numero dei blocchi della tabella stessa e quindi:
   Accessi =$l * n/b = 100 * 1000000/1000 = 100000$
4. Utilizzando l'indice primario (separato dal file) si fa riferimento ad una struttura ordinata ad un solo livello
   numero blocchi indice = dimensione_indice$/B = 80000/1000 = 80blocchi$
   80 accessi nel caso peggiore.
   É necessario, infine, un ulteriore accesso per recuperare il record puntato.
5. Utilizzando l'indice secondario ad un solo livello la complessità è data dal numero di blocchi dell'indice. Si tiene inoltre conto dell'opportuna probabilità di esistenza del record.
   numero blocchi indice = dimensione_indice $/B = 10000000/1000 = 10000$blocchi

Essendo la struttura ordinata, la probabilità di visitare tutto l'indice è data, nel caso peggiore, dalla probabilità di presenza della chiave in ultima posizione (s).
Si ha accessi_medi $= 10000 * s = 8000$accessi
E' necessario, infine, un ulteriore accesso per recuperare il record puntato.

Esercizio 10.4 Si considerino un sistema con blocchi di dimensione $B = 1000$ byte e puntatori ai blocchi di $P = 2$byte e una relazione R(A,B,C,D,E) di cardinalità pari circa a $N = 1000000$, con tuple di $L = 50$byte e campo chiave A di $K = 5$ byte. Valutare i pro e i contro (in termini di numero di accessi a memoria secondaria e trascurando le problematiche relative alla concorrenza) relativamente alla presenza di un indice secondario sulla chiave A e di un altro, pure secondario, su B, in presenza del seguente carico applicativo:

1. inserimento di una nuova tupla (con verifica del soddisfacimento del vincolo di chiave), con frequenza $f_1 = 500$ volte al minuto;
2. ricerca di una tupla sulla base del valore della chiave A con frequenza $f_2 = 500$ volte al minuto;
3. ricerca di tuple sulla base del valore di B con frequenza $f_3 = 100$.

Soluzione
Si suppone la disponibilità di buffer che permettano di mantenere stabilmente in memoria due livelli per ciascun indice e considerando che la relazione possa essere memorizzata in forma contigua (assumendo un rapporto 100:1 fra tempo di posizionamento della testina e tempo di lettura).

Si suppone inoltre $L_B = 3$.
È necessario calcolare il costo delle quattro operazioni, in presenza e assenza degli indici. Notazioni:

N_T numero di blocchi della relazione T; è pari circa a $N/(B/L)$ = 50.000
$cont$ riduzione di costo dovuta alla contiguità; è pari a 100.
seq costo della scansione sequenziale $1 + (NT1)/cont$; è pari a circa 500.
$prof_A$ il fattore di blocco dell' indice è 1.000/7, circa 140 e quindi, con un riempimento di circa il 60-70%, il fan-out è circa 100 e quindi i livelli necessari sono 3.
$prof_B$ il fattore di blocco dellindice è 1.000/5, circa 200 e quindi, con un riempimento di circa il 60-70%, il fan-out è circa 135 e quindi i livelli necessari sono ancora 3.

Op_1 È influenzata da entrambi gli indici anche se in modo diverso. Vediamo i diversi comportamenti.

Nessun indice: è necessaria la scansione sequenziale per verificare il vincolo di chiave. Costo pari a $seq = 500$; nessun costo per la manutenzione degli indici.

Indice su A: ed accesso tramite lindice. Costo pari alla profondità dellindice su A, più uno (per accedere al blocco del file) meno due (grazie ai buffer); costo pari a 1.

Indice su B: scansione sequenziale per la verifica della chiave e accesso allindice per laggiornamento.
Indici sia su A sia su B: accesso ai due indici.

Op_2 È influenzata dal solo indice su A accessi diretti o sequenziali.

Op_3 È influenzata dal solo indice su B. E' sufficiente aggiungere al risultato precedente il fattore relativo alla molteplicità.

Esercizio 10.5

Alcuni DBMS permettono una tecnica di memorizzazione chiamata “co-clustering” o “clustering eterogeneo” in cui un file contiene record di due o più relazioni e tali record sono raggruppati (eventualmente, ma non necessariamente ordinati) secondo i valori di opportuni campi dell’una e dell’altra relazione. Ad esempio, date due relazioni

- Ordini(CodiceOrdine, Cliente, Data, Importo)
- LineeOrdine(CodiceOrdine, Linea, prodotto, Quantità, Importo)

Questa tecnica (con riferimento agli attributi CodiceOrdine delle due relazioni) permetterebbe una memorizzazione contigua di ciascun ordine con le rispettive “linee d’ordine”, cioè dei prodotti ordinati (ciascun ordine fa riferimento a più prodotti, ognuno su una “linea”).
Con riferimento all’esempio, indicare quali delle seguenti operazioni possono trarre vantaggio dall’uso di questa opportunità e quali ne possono essere penalizzate (spiegare la risposta anche in termini quantitativi, individuando valori opportuni per i principali parametri di interesse; supporre che siano utilizzati indici su CodiceOrdine, in tutti i casi, due per la memorizzazione tradizionale e uno nel caso di utilizzo del cluster eterogeneo):

1. stampa dei dettagli (cioè delle linee d’ordine) di tutti gli ordini (ordinati per codice)
2. stampa dei dettagli di un ordine
3. stampa delle informazioni sintetiche (codice, cliente, data, totale) di tutti gli ordini.

Soluzione:

1. Per quanto riguarda la prima operazioni, non ci sono dubbi che una struttura co-cluster porti notevoli benefici, in quanto è sufficiente posizionarsi sul primo record della relazione e scorrerne tutto il contenuto. Quindi, la prima operazione è VANTAGGIOSA.
2. La seconda operazione è più delicata, in quanto se si ha a disposizione una struttura hash o ad albero sulla relazione ordini è vantaggiosa, mentre se si ha una struttura sequenziale o disordinata è svantaggiosa. Quindi, in questo caso DIPENDE DALLA STRUTTURA DELLA RELAZIONE ORDINI.
3. La terza operazione è sicuramente svantaggiosa, perché in ogni caso devo accedere a tutte le tuple del co-cluster nonostante abbia bisogno dei soli dati della relazione ordini. Quindi, la terza operazione è SVANTAGGIOSA.

Esercizio 10.6 Si consideri una relazione IMPIEGATO (Matricola, Cognome, Nome, Data- Nascita) con un numero di ennuple pari a Nabbastanza stabile nel tempo (pur con molti inserimenti ed eliminazioni) e una dimensione di ciascuna ennupla (a lunghezza fissa) pari a L byte, di cui K per la chiave Matricola e C per il campo Cognome.
Supporre di avere a disposizione un DBMS che permetta strutture fisiche disordinate (heap), ordinate (con indice primario sparso) e hash e che preveda la possibilità di definire indici secondari e operi su un sistema operativo che utilizza blocchi di dimensione B e con puntatori ai blocchi di P caratteri.
Indicare quale possa essere lorganizzazione fisica preferita nel caso in cui le operazioni principali siano le seguenti:

1. ricerca sul cognome (o una sua sottostringa iniziale, abbastanza selettiva, si supponga che mediamente una sottostringa identifichi $S = 10$ ennuple) con frequenza f_1;
2. ricerca sul numero di matricola, con frequenza f_2;
3. ricerca sulla base di un intervallo della data di nascita (poco selettivo, restituisce circa il 5% delle ennuple), con frequenza f_3 molto minore di f_1 e f_2;

assumendo $N = 10000000$ ennuple, $L = 100$ byte, $K = 5$ byte, $C = 20$ byte, $B = 1000$ byte, $P = 4$ byte, $f_1 = 100$ volte al minuto, $f_2 = 2000$ volte al minuto, $f_3 = 1$ volte al minuto. Individuare le alternative più sensate sulla base di ragionamenti qualitativi e poi valutarle quantitativamente, ignorando i benefici derivanti dai buffer e dalla contiguità di memorizzazione.

Soluzione

Dividiamo la risposta in tre parti:
Scelta qualitativa delle strutture fisiche più indicate:

Le strutture fisiche debbono privilegiare le operazioni 1 e 2 che sono più frequenti della 3 (che per giunta è poco selettiva).

1. Per l'operazione 1 la struttura hash su cognome non va bene, perché le ricerche non sono esatte (cioè sono spesso fatte con parte del cognome). Poiché nel testo non è chiarito se oltre ad essere abbastanza stabile la dimensione sono limitati anche gli inserimenti e le eliminazioni, non si può dire se sia preferibile una struttura ordinata su cognome (con un indice sparso) oppure una disordinata (con indice denso su cognome); nel prosieguo, supponiamo che il grado di dinamicità sia limitato (o che sia possibile avere tempi morti per la riorganizzazione) e quindi sia possibile una struttura ordinata (che darebbe grandi benefici per le ricerche su sottostringa). In ogni caso, nell'interesse dell'operazione 2, sarebbe utile un indice secondario sulla matricola.
2. La struttura che meglio privilegerebbe l'operazione 2 è quella hash su matricola: la struttura hash è la più effciente per l'accesso diretto puntuale (cioè con un valore di chiave completo), a patto che la dimensione della relazione

sia abbastanza stabile; in questo caso entrambe le condizioni sono soddisfatte. In questo contesto, per supportare adeguatamente anche l'operazione 1, si potrebbe utilizzare un indice secondario sul cognome.

Valutazione analitica:
Ignoriamo gli arrotondamenti. Sono necessari $K = 5$ byte per la chiave e $P = 4$ byte per i puntatori ai blocchi.
In tutti i casi abbiamo: fattore di blocco del file (numero di record per blocco): $F_f = B/L = 10$
fattore di blocco dell'indice: $F_i = B/(K + P) = 111$
Esaminiamo le due alternative sopra illustrate
Costo unitario delle tre operazioni:

1. accesso per cognome tramite l'indice secondario; l'indice ha N record (poiché denso) e quindi l'albero ha N/F_i foglie e profondità pari a $log_{F_i}(N/Fi)$; un'operazione ha costo pari alla profondità aumentata di S (l'indice è secondario, quindi gli accessi a cognomi consecutivi portano a blocchi diversi);
2. accesso hash sulla matricola: costo costante, circa 1.3 accessi a blocchi in media;
3. accesso sequenziale: costo pari al numero di blocchi del file: $(N * 1.5)/F_f$ (il fattore 1.5 è motivato dagli spazi liberi necessari per la struttura hash).

costo totale:
$f_2 * 1.3 + f_1 * (S + log_{F_i}(N/F_i)) + f_3 * (N * 1.5)/F_f$

con $f_1 = 100$, $f_2 = 2000$, $f_3 = 1$

$2000*1.3+100*(10+log_{111}(10000000/111))+1*(10000000*1.5)/10 = 78800$

Costo unitario delle tre operazioni:

1. accesso per cognome tramite l'indice primario; l'indice ha N/F_f record (poiché sparso) e quindi l'albero ha $(N/F_f)/F_i$ foglie e profondità pari a $log_{F_i}((N/F_f)/F_i)$; un'operazione ha costo pari alla profondità aumentata di 1 (in questo caso cognomi consecutivi portano a record adiacenti);
2. accesso per matricola tramite l'indice secondario; come sopra, ma con un solo blocco da accedere invece di S, visto che le matricole sono "esatte": un'operazione ha costo pari $1 + log_{F_i}(N/F_i)$
3. accesso sequenziale come sopra (ma senza il fattore 1.5)

Costo totale:

$$f_1 * (1 + log_{F_i}((N/F_f)/F_i)) + f_2 * (1 + log_{F_i}(N/F_i)) + f_3 * N/F_f$$

$$f_1 = 100, f_2 = 2000, f_3 = 1$$

$100 * (1 + log_{111}((10000000/10)/111)) + (2000 * (1 + log_{111}(10000000/111)) + 10000000/111 = 6107$

Esercizio 10.7 Illustrare brevemente vantaggi e svantaggi (relativamente a vari tipi di operazioni) delle seguenti strutture fisiche, definite con riferimento ad uno stesso attributo A non chiave:

1. indice primario (sparso) multilivello statico;
2. indice secondario multilivello statico;
3. B+ tree secondario;
4. hash.

Soluzione

1. Indice primario (sparso) multilivello statico: va bene per ricerche puntuali e su intervallo, ma soffre in caso di aggiornamenti.
2. Indice secondario multilivello statico: va bene per ricerche puntuali e su intervallo (un po' meno del precedente, perché l'indice è più grande e soprattutto perché il file non è ordinato, il che è rilevante perchè il campo non è chiave e sono considerate ricerche su intervalli), ma soffre in caso di aggiornamenti.
3. B+ tree secondario: come i precedenti, un po' meno effciente ma senza svantaggi particolari in presenza di aggiornamenti.
4. Hash: molto effciente per accessi puntuali, ma non per intervalli; degenera se le dimensioni variano signifcativamente.

Esercizio 10.8 Indicare, in ciascuno dei quattro casi dell'esercizio 10.7(sia in forma simbolica sia in forma numerica), il costo di operazioni di ricerca basate su A ("trovare i record con un certo valore per l'attributo A"), con riferimento ad una relazione R contenente $L = 100000000$ ennuple di $R = 25$ byte ciascuna, di cui $a = 12$ per l'attributo A. Si considerino mediamente $m = 3$ record con lo stesso valore su A; supporre che i blocchi abbiano dimensione $B = 2$KB, approssimabile come $B = 2000$e che i puntatori ai blocchi abbiano lunghezza $p = 4$.

Soluzione

1. $\text{LIV}_1 = 4$, dove LIV_1 è la profondità dell'indice, pari a $log_F L/(B/R) = 3$, dove F è il fattore di blocco dell'indice, pari a $B/(a + p)$;
2. $\text{LIV}_2 + m = 7$, dove LIV_2 è la profondità dell'indice, pari a $log_F L = 4$; si noti che il termine m è necessario qui perchè il file è disordinato, mentre non è necessario nel caso precedente, perché i record sono tutti nello stesso blocco (salvo poche eccezioni).
3. $\text{LIV}_3 + m = 8$, dove LIV_3 è la profondità dell'indice, pari a $log_{F'} L = 5$ dove $F' = 2/3F$ (supponendo un riempimento medio del 66%).
4. poco più di 1.