Corso di Basi di dati
Prova scritta parziale
31 marzo 2000

Compito A
Cenni sulle soluzioni

Tempo a disposizione: due ore. Libri e appunti aperti.

Domanda 1 (25%)
Si consideri una relazione IMPIEGATO(Matricola,Cognome,Nome,DataNascita) con un numero di ennuple pari a N abbastanza stabile nel tempo e una dimensione di ciascuna ennupla (a lunghezza fissa) pari a L byte.
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.
Calcolare (approssimativamente) il numero di accessi a memoria secondaria (nell'unità di tempo) supponendo per la suddetta relazione varie organizzazioni (quelle che a prima vista si ritengono preferibili) nel caso in cui le operazioni principali siano le seguenti:

1. ricerca sul numero di matricola, con frequenza f_1
2. ricerca sul cognome (o una sua sottostringa iniziale, abbastanza selettiva, in media una sottostringa identifica $S = 4$ ennuple) con frequenza f_2
3. ricerca sulla base di un intervallo della data di nascita (poco selettivo), con frequenza f_3 molto minore di f_1 e f_2

In particolare, indicare la soluzione preferita (assumendo $N = 10000$, $L = 50$ e $B = 512$) nei casi

(a) $f_1 = 100$, $f_2 = 500$, $f_3 = 1$
(b) $f_1 = 500$, $f_2 = 100$, $f_3 = 1$

Commentare anche quanto possa essere opportuno fare scelte diverse se il numero di ennuple N può variare significativamente nel tempo.
Soluzione (cenni)
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).

(i) La struttura che meglio privilegerebbe la 1 è quella hash su matricola: la struttura hash è la più efficiente 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 2, si potrebbe utilizzare un indice secondario sul cognome.

(ii) Per l'operazione 2 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 1, sarebbe utile un indice secondario sulla matricola.

Valutazione analitica. Ignoriamo gli arrotondamenti. Supponiamo che siano necessari $K = 3$ byte per la chiave e $P = 2$ 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: $F_i = B/(K + P) = 100$
esaminiamo le due alternative sopra illustrate

(i) costo unitario delle tre operazioni:

1. accesso hash sulla matricola: costo costante, circa 1.3 accessi a blocchi in media;

2. 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 $\log_{F_i}(N/F_i)$; un'operazione ha costo pari alla profondità aumentata di S (l'indice è secondario, quindi gli accessi a cognomi consecutivi portano a blocchi diversi);
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:

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

Nei due casi citati nel testo:

(a) $f_1 = 100$, $f_2 = 500$, $f_3 = 1$:

$$100 * 1.3 + 500 * (4 + \log_{100}(10000/100)) + 1 * (10000 * 1.5)/10 = 4130$$

(b) $f_1 = 500$, $f_2 = 100$, $f_3 = 1$

$$500 * 1.3 + 100 * (4 + \log_{100}(10000/100)) + 1 * (10000 * 1.5)/10 = 2710$$

(ii) costo unitario delle tre operazioni:

1. 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 $1 + \log_{F_i}(N/F_i)$
2. 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);
3. accesso sequenziale come sopra (ma senza il fattore 1.5)

costo totale:

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

(a) $f_1 = 100$, $f_2 = 500$, $f_3 = 1$:

$$100 * (1 + \log_{100}(10000/100)) + 500 * (1 + \log_{100}((10000/10)/100)) + 10000/10 = 2800$$

(b) $f_1 = 500$, $f_2 = 100$, $f_3 = 1$

$$500 * (1 + \log_{100}(10000/100)) + 100 * (1 + \log_{100}((10000/10)/100)) + 10000/10 = 2200$$

Caso della dimensione fortemente variabile nel tempo. Se non sono previsti tempi morti, la struttura hash non è accettabile. In questo caso, diventa probabilmente preferibile una struttura disordinata con due indici secondari.

Domanda 2 (20%)
Si consideri il seguente problema: uno stadio ha un certo numero di aree (identificate da lettere) ognuna con più settori (identificati da numeri, unici nell'ambito delle aree); ciascun settore ha più posti (identificati da numeri, unici all'interno del settore). Vengono accettate prenotazioni per (i) singoli posti; (ii) interi settori; (iii) intere aree. Illustrare una struttura di dati che permetta di rappresentare in modo compatto le prenotazioni e di rispondere in modo efficiente alle richieste di prenotazione. Suggerimento: utilizzare le idee che sono alla base del lock gerarchico; si noti però che l'albero non va rappresentato per intero (altrimenti la struttura non sarebbe compatta).

Soluzione (cenni)
Il lock gerarchico è realizzato con una tabella che include ennuple solo per gli elementi bloccati (al rispettivo livello di granularità). La risposta alla domanda doveva proporre una struttura di tale tipo; ad esempio tre tabelle con tre campi "identificanti" (non nel senso relazionale del termine, perché uno o due valori potrebbero essere "nulli") e un campo che indichi se si tratta di "prenotazione" al livello di granularità in questione (in un certo senso "lock") oppure ad un livello più fine (corrispondente all'intenzione di lock).

Domanda 3 (15%)
Descrivere (informalmente o in pseudocodice) un algoritmo efficiente per l'effettuazione del join di due relazioni in cui l'attributo di join ha un dominio costituito da pochi valori diversi (per esempio gli interi da 1 a 10). Nota: È possibile ottenere un algoritmo con complessità lineare (rispetto alle dimensioni del risultato), variante della tecnica"hash-based" per la realizzazione dei join.

Soluzione (cenni)
Un algoritmo con complessità lineare può essere realizzato in due fasi:

1. una scansione separata di ciascuna relazione e la costruzione di 10 liste, una per ciascun valore.
2. per ogni coppia di liste corrispondenti a ciascun valore, si producono le ennuple combinando tutti gli elementi dell'una con tutti gli elementi dell'altra

La prima fase ha complessità lineare rispetto all'input (somma delle dimensioni delle due relazioni), la seconda rispetto all'output (dimensione del join).

Domanda 4 (20%)
Le seguenti affermazioni sono tutte vere. Provare a dare una breve spiegazione per ciascuna di esse

1. i primi DBMS a oggetti non prevedevano linguaggi di interrogazione
2. successivamente, ci si è resi conto che un DBMS a oggetti non può non avere un linguaggio di interrogazione
3. OQL non prevede funzionalità per l'aggiornamento.

Soluzione (cenni)
Tutte e tre le questioni sono legate al rapporto fra linguaggio di interrogazione, che considera tutti i dati "a disposizione," e concetto di incapsulamento, che viceversa limita le operazioni e la visibilità a ciò che i metodi esplicitamente permettono.

1. Si riteneva che un linguaggio di interrogazione violasse l'incapsulamento.
2. Altrimenti si sarebbe riscontrato un "passo indietro" rispetto ai sistemi relazionali.
3. In questo modo si conciliano in qualche modo le due esigenze precedenti: si ha un linguaggio di interrogazione ad alto livello, ma si mantiene l'incapsulamento per gli aggiornamenti.

Domanda 5 (20%)
Progettare uno schema multidimensionale a stella (star) o a fiocco di neve (snowflake) per un semplice data warehouse che voglia rappresentare informazioni relativamente agli incassi registrati da un insieme di film. I fatti di interesse sono le proiezioni, le misure gli incassi e il numero di spettatori, le dimensioni il luogo (cinema, con città, regione e area geografica [nord, centro, sud, isole]), il tempo (giorno, mese, trimestre), il film proiettato (con titolo, genere, nazione). Mostrare sia lo schema ER sia quello relazionale.

Soluzione (cenni)
Schema relazionale a stella:

- PROIEZIONE(Luogo,Tempo,Film,NumSpettatori,Incasso)
- LUOGO(Codice,Cinema,Città,Regione,Area)
- TEMPO(Codice,Giorno,Mese,Trimestre)
- FILM(Codice,Titolo,Genere,Nazione)

Lo scehma a fiocco di neve si ottiene normalizzando le relazioni LUOGO e TEMPO (eventualmente introducendo codici); quello E-R a stella ha l'entità PROIEZIONE al centro e le altre tre collegate ad essa con relationship uno a molti.