RIASSUNTO

Devo controllare la clausola **WHERE** e decidere se sto lavorando su :

una chiave primaria o secondaria => **B+tree primario o secondario (*NL*,*g* e *h* diversi)**

clustered o unclustered => ho un piano di accesso diverso!!

Piano di accesso

scansione sequenziale costo su tabella R

$$C_{acc}(seqR) = NP + \alpha \cdot NT$$

con indice IX(R.A) ossia Indice IX su attributo A di R

clustered $C_{acc}(IX(R.A)) = h - 1 + f_p \cdot NL + f_p \cdot NP + \alpha \cdot f_p \cdot NT$

unclustered $C_{acc}(IX(R.A)) = h - 1 + f_p \cdot NL + EK \cdot \phi(\frac{NT}{NK}, NP) + \alpha \cdot f_p \cdot NT$

dove $EL = f_p \cdot NL$, $EP = f_p \cdot NP$, $ET = f_p \cdot NT$

NP = numero pagine su disco delle tabelle

NT = NR = numero di tuple totali (n° record)

NL = numero di foglie dell'indice

EL = numero atteso di pagine foglia da accedere

EP = numero atteso di pagine dati da accedere

ET = numero di tuple per cui il predicato p è verificato

fp = selettività del predicato

EK = numeri di valori di chiave che verificano il predicato

NK = numero di valori distintivi di chiave

D = dimensione pagina in byte

u = fattore di riempimento foglie

h = altezza albero

g = ordine albero

alpha = parametro costi I/O e CPU

Costo accesso alle tabelle

Indice B-Tree

$$h = int.sup[\log_{(2g+1)}(NK+1)] \leqslant h \leqslant int.sup[1+\log_{(2g+1)}\frac{(NK+1)}{2}]$$

Costo Ricerca : $1 \leq C(search) \leq h$

Costo Inserimento :

senza split C(min) = h +1

con split C(max) = 3h +1

costo medio $C(avg) = h + 1 + \frac{2}{g}$

Ricerca binaria solo su file ordinato

peggiore $int.inferiore[\log_2(NP)]+1$ blocchi

Ricerca dicotomica

IP: blocco di memoria

costo di riempimento di 1 dato con accesso casuale $int.sup[\log_2(IP)]+1$ con primary key

$int.sup[\log_2(IP)]+accessi$ con secondary key

(accessi in meno ottenuti coi puntatori trovati)

NP : Numero di pagine

$$NP = int.sup[\frac{(NR \cdot len(t))}{(u \cdot D)}]$$

NK : numero di chiavi

$$NK = (2g+1)^h - 1$$

$$\log_N(X) = \frac{(\log(X))}{(\log(N))}$$

Indice B⁺-tree

Albero bilanciato => foglie contengono puntatori ai record , i nodi interni i valori chiave per individuare la foglia

D	= dimesione pagina
NK	= numero valori distinti di chiave
NT = NR	= numero tuple da indicizzare
len(p)	= dimensione puntatore
len(k)	= dimensione separatore chiave
u	= fattore riempimento nodi

*Caso B⁺-tree **PRIMARY***

Ordine : $g = int.inferiore[\frac{(D - len(p))}{(2 \cdot [len(k) + len(p)])}]$

N° foglie : $NL = int.sup[\frac{NR \cdot [len(k) + len(p)]}{(D \cdot u)}]$

NB: qui NR=NK=NT in quanto siamo nel caso di indice primario dove gli elementi del campo sono quindi unici (non ripetuti)

Altezza (senza considerare il puntatore alla foglia successiva) :

$$h = 1 + int.sup[\log_{(2g+1)}(NL)] \leq h \leq 2 + int.sup[\log_{(2g+1)}(\frac{NL}{2})]$$

Altezza (generica) :

$$h = 1 + int.sup[\log_{(2g+1)}(NN)] \leq h \leq 2 + int.sup[\log_{(2g+1)}(\frac{NN}{2})]$$

con NN = min (NL,NK)

*Caso B⁺-tree **SECONDARY***

Ordine : $g = [\frac{(D - N_{PUNTATORI} \cdot len(p))}{(2 \cdot (len(k) + N_{PUNTATORI} \cdot len(p)))}]$ con $N_{PUNTATORI} = \frac{NR}{NK_p}$

N° foglie : $NL = [\frac{(NK \cdot len(k) + NR \cdot len(p))}{(D \cdot u)}]$

NB: qui NR != NK

Altezza :

$$h = 1 + int.sup[\log_{(2g+1)}(NN)] \leq h \leq 2 + int.sup[\log_{(2g+1)}(\frac{NN}{2})]$$

con NN = min (NL,NK)

Indice Clustered

Le tuple della tabella sono ordinare rispetto all'attributo dell'indice.

Costo di accesso

$c_T = EL + EP + \alpha \cdot ET$, $0{,}01 < \alpha < 0{,}3$ serve per quantizzare il tempo di elaborazione edi I/O

$c_T = EL + EP$, è l'espressione per calcolare solo il teorico non considerando i ritardi fisici

$EL = (h-1) + int.sup[f_p \cdot NL]$, con $f_p = \frac{EK}{NK}$ *NB: NL è il numero delle foglie dell'indice*

$EP = int.sup(f_p \cdot NP)$, con : *NP pari al numero di pagine disco della tabella*

Indice Unclustered

$EL = (h-1) + int.sup[f_p \cdot NL]$, con $f_p = \frac{EK}{NK}$ *NB: NL è il numero delle foglie dell'indice*

$EP = EK \cdot \phi(\frac{NT}{NK}, NP)$ con : *NT / NK la selettività della chiave*

Cardenas

$\phi(\frac{NT}{NK}, NP) = NP \cdot (1-(1-\frac{1}{NP})^{\frac{NT}{NK}}) \leqslant min\ ET, NP$

$ET = int.sup[f_p \cdot NT]$, $ER = \frac{NT}{NK}$

NP = n° pagina del file dati

NT = n° delle tuple

Fattore di Selettività

il **fattore di selettività** (f_p) è sempre valutato nell'ipotesi che sia una distribuzione uniforme

$0 \leqslant f_p \leqslant 1$, $f_p = \frac{EK}{NK}$ dove EK è il numero di chiave residue dopo la relazione

- Predicato "=" $F_{(a=V)} = \frac{1}{NK_a}$
- Predicato "IN" $f_{(A \in SET)} = \frac{(cardinalità(SET))}{NK_a}$
- Predicato "<" $f_{a<v} = \frac{\frac{(v - min(a))}{(max(a) - min(a))} \cdot (NK_a - 1)}{NK_a} \simeq \frac{(v - min(a))}{(max(a) - min(a))}$
- Predicato ">" $f_{a>v} = \frac{\frac{(max(a) - v)}{(max(a) - min(a))} \cdot (NK_a - 1)}{NK_a}$
- Predicato "between" $f_{(a \in [v1, v2])} = \frac{(v2 - v1)}{(max(a) - min(a))}$

Esempio :

Employee(EmpID,DeptNo, Salary, Job, Sex)

$NT_{employee}$ = 20000, NK_{deptno} = 100 , NK_{job} = 10 , NK_{sex} = 2 , NK_{salary} = 10

min(salary) = 5000 , max(salary) = 50000

calcolo predicato selettività :

DeptNo = 51 $f_p = \frac{1}{100}$, ET = 200

Salary > 10000 $f_p = \frac{(50 - 10)}{(50 - 5)} \cdot (\frac{9}{10}) = \frac{8}{10}$, ET = 16000

C SORT – S&M a Z Vie

Costo ordinamento algo S-M a Z vie : usato in caso di ORDER BY se non esistono indici sul campo da ordinare.

$$C_{sort} = 2 \cdot NP \cdot (1 + int.sup(\log_z(\frac{NP}{NB})))$$

NP = n° pagine file

NB = numero di blocchi in mem. Centrale

Z = NB – 1 (grado di fusione)

NESTED LOOP JOIN

R = esterna , S = interna

$$C_{acc} = C_{acc}(R) + ET_R \cdot C_{acc}(S)$$

avrò $ET_R = f_{TOTR} \cdot NT_R$ dove f_{TOTR} è il fattore di seletività calcolato su tutti i predicati ad esclusione di quelli di join.

A	AND	A	$f_{S(A)} * f_{S(A)}$	
A	OR	A	$f_{S(A)} + f_{S(A)}$	
A	OR	B	$f_{S(A)} + f_{S(B)} - f_{S(A)} * f_{S(B)}$	(Sono in un JOIN) slide 38 piani accesso
A	AND	B	$f_{S(A)}$	nel caso di clausola WHERE su A

Come scegliere tra interna ed esterna : $\frac{NT_R}{NP_R} < \frac{NT_S}{NP_S}$ se ho un indice su una tabella allora è meglio che quella tabella sia interna.

{da qui in poi uso scansioni sequenziali}

NESTED BLOCK JOIN

Ho un buffer di B pagine in memoria centrale in cui metto le pagine di R

$$C_{acc}(senza\ f_R) = NP_R + int.sup\left(\frac{NP_R}{(B-1)}\right) \cdot NP_S$$

SORT MERGE JOIN

Sia R che S sono ordinati sugli attributi di JOIN o hanno un index su quei campi

$$C_{acc} = NP_R + NP_S$$

altrimenti se non valgono le precondizioni

$$C_{acc} = sort(R) + sort(S) + NP_R + NP_S$$

SIMPLE HASH JOIN

Hash di valori dell'attributo di join di una tabella (S) => costruisci l'hash table => per ogni tupla della tabella R si accede all'hash table applicando la funzione di hash

$$C_{acc} = NP_S + 2 \cdot NT_S + NP_R + NT_R$$