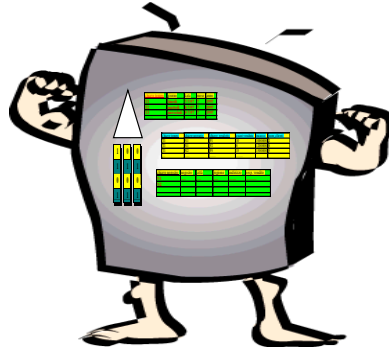


Organizzazioni dei dati - parte III



Dario Maio

<http://bias.csr.unibo.it/maio/>

Organizzazioni dei dati- parte III



1

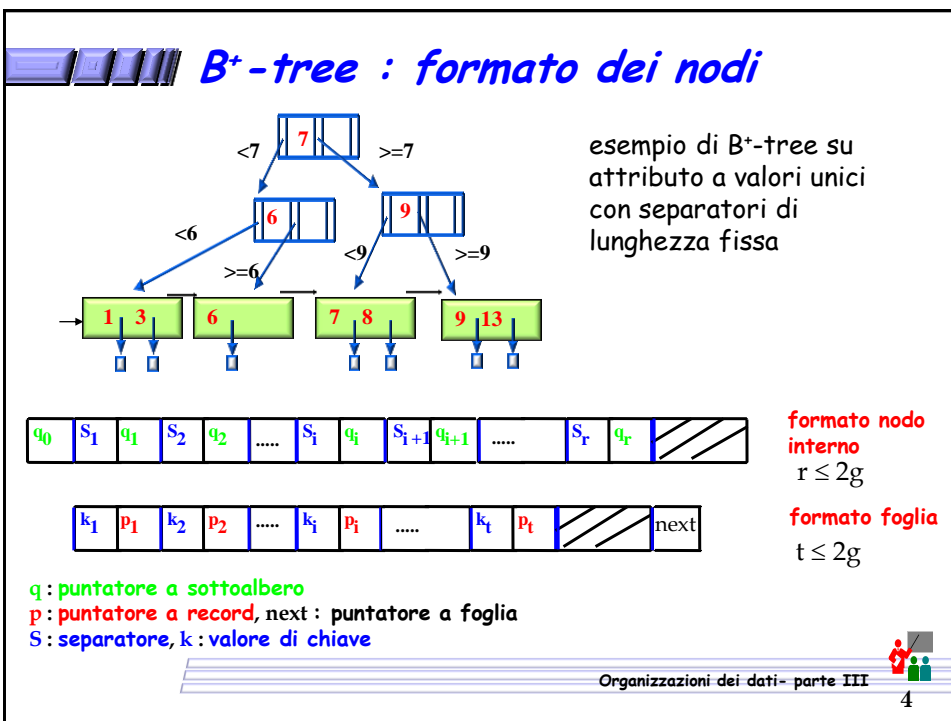
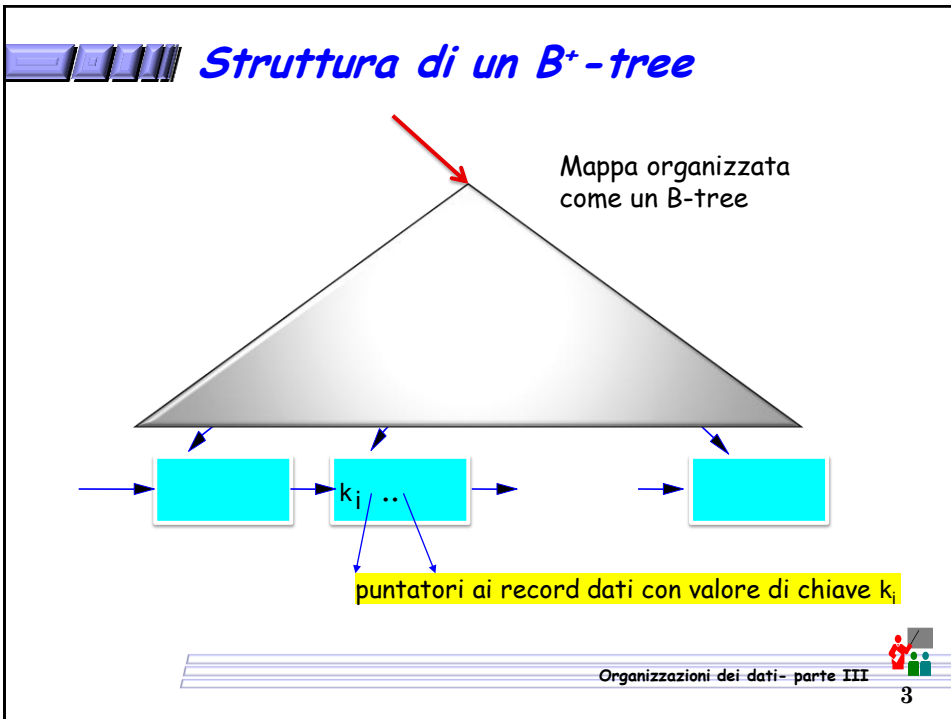
B⁺-tree

- In un B-tree, i valori di chiave svolgono una duplice funzione:
 - ✚ come **separatori** permettono di determinare il cammino da seguire in fase di ricerca;
 - ✚ come **valori di chiave** permettono di accedere all'informazione a essi associata.
- Nei B⁺-tree queste funzioni sono mantenute separate:
 - ✚ Le foglie contengono tutti i valori di chiave.
 - ✚ I nodi interni, organizzati come un B-tree, costituiscono solo una "**mappa**" per consentire una rapida localizzazione delle chiavi, e memorizzano "**separatori**" di cammino.
- Al fine di facilitare elaborazioni sequenziali e su intervalli, **le foglie sono tra loro concatenate in una lista**. È inoltre presente un puntatore alla testa della lista.

Organizzazioni dei dati- parte III

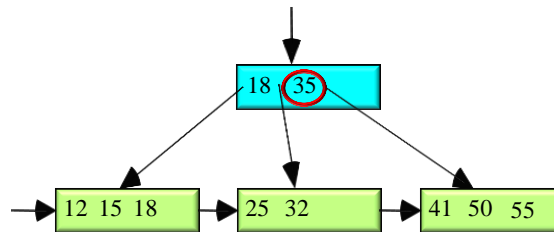


2



Separatori (1)

- In un B⁺-tree, la sola funzione dei separatori è determinare il giusto cammino quando si ricerca un valore di chiave. Dunque è anche possibile che un separatore non sia un valore di chiave presente nel file dati.
- Il sottoalbero sinistro di un separatore contiene valori di chiave minori o uguali al separatore. Il sottoalbero destro contiene valori di chiave strettamente maggiori del separatore (è possibile anche adottare la convenzione che i valori uguali siano nel sottoalbero destro).



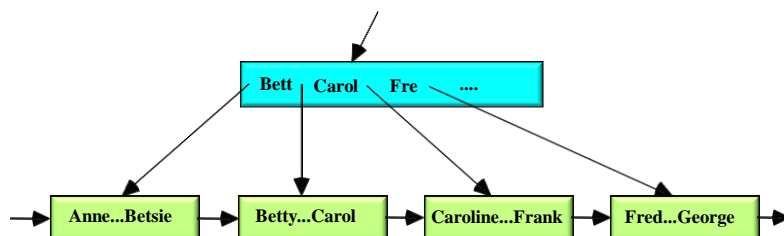
Organizzazioni dei dati- parte III



5

Separatori (2)

- In un B⁺-tree, nel caso di chiavi alfanumeriche la scelta dei separatori è particolarmente importante, in quanto, facendo uso di separatori di lunghezza ridotta si risparmia spazio e si riduce, eventualmente, l'altezza dell'albero.
- Esistono molte varianti in cui si esegue una "compressione" dei separatori; tra queste si ricordano i **Simple Prefix B⁺-tree** (di cui la figura è un esempio) e i **Prefix B⁺-tree**.



Organizzazioni dei dati- parte III



6



Ordine di un B⁺-tree

- In un B⁺-tree l'ordine è un concetto ancora significativo solo se si fa uso di separatori di lunghezza fissa. Negli altri casi si può considerare, con un certo grado di approssimazione, la lunghezza media dei separatori stessi.
- Se è possibile avere nodi formati da più blocchi accessibili con una singola operazione di lettura, allora valgono anche per i B⁺-tree le considerazioni fatte per i B-tree sull'influenza dell'ordine sulle prestazioni.
- Si supponga, viceversa, che la dimensione di un nodo sia fissata a **D** byte (es. 4 KB) e che ogni puntatore, **q**, a nodi del B⁺-tree richieda **len(q)** byte. Si supponga inoltre che i separatori siano gli stessi valori di chiave, di lunghezza **len(k)** byte.
Poiché $2g \times \text{len}(k) + (2g+1) \times \text{len}(q) \leq D$ si deriva che l'ordine di un B⁺-tree è:

$$g = \left\lfloor \frac{D - \text{len}(q)}{2(\text{len}(k) + \text{len}(q))} \right\rfloor$$

- Con pagine di **D=4096 byte**, **len(q)=4 byte**, e chiavi con **len(k)=10** caratteri, si ha un B⁺-tree di ordine **g=146**, con 4 byte inutilizzati in ogni nodo.
- Con chiavi di 40 caratteri l'ordine si riduce a 46, con 42 byte inutilizzati in ogni nodo.



Primary B⁺-tree: numero foglie

- In un B⁺-tree **per chiave primaria** il numero delle foglie dipende dal numero dei record, **NR**, presenti nel file, dalla dimensione dei nodi, **D**, e dall'utilizzazione delle foglie stesse.
- Si può dimostrare (e si verifica sperimentalmente) che l'utilizzazione media delle foglie, **u**, è pari a circa **ln 2 ≈ 0.69**.
- Trascurando, per semplicità, la presenza del puntatore alla foglia successiva, e considerando chiavi di lunghezza **len(k)** e puntatori ai record dati (TID) di lunghezza **len(p)**, il numero di foglie, **NL**, si può calcolare come:

$$NL = \left\lceil \frac{NR \times (\text{len}(k) + \text{len}(p))}{D \times u} \right\rceil$$

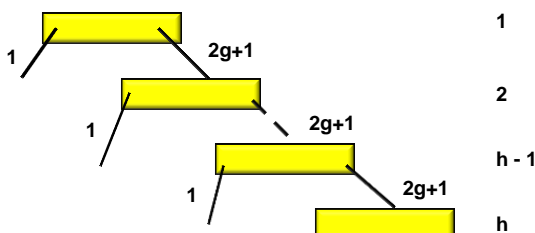




Primary B⁺-tree: altezza minima

- Si ottiene quando tutti i nodi dei livelli intermedi (radice compresa) sono pieni. Poiché al penultimo livello sono necessari almeno NL puntatori alle foglie, si ottiene:

$$(2g+1)^{h-1} \geq NL$$



- da cui:

$$h \geq 1 + \lceil \log_{2g+1} NL \rceil$$



Primary B⁺-tree: altezza massima

- Nel caso peggiore la radice contiene 2 soli puntatori e ogni nodo dei livelli intermedi ne contiene g+1, da cui segue:

$$2 \times (g+1)^{h-2} \leq NL$$

e quindi

$$h \leq 2 + \left\lceil \log_{g+1} \frac{NL}{2} \right\rceil$$

- Dunque l'altezza di un primary B⁺-tree di ordine g è:

$$1 + \lceil \log_{2g+1} NL \rceil \leq h \leq 2 + \left\lceil \log_{g+1} \frac{NL}{2} \right\rceil$$

Esempi calcolo altezze

- La dimensione dei nodi è pari a $D = 1 \text{ KB}$, e tutti i puntatori sono lunghi 4 byte.

$\log_2 NR$	len(k) = 10 byte g = 36		len(k) = 20 byte g = 21	
	h_{\min}	h_{\max}	h_{\min}	h_{\max}
1	1	1	1	1
4	1	1	1	1
5	1	1	2	2
10	2	2	2	2
11	2	2	3	3
12	3	3	3	3
14	3	3	3	3
15	3	3	3	4
16	3	3	4	4
17	3	3	4	4
18	3	4	4	4
19	4	4	4	4
20	4	4	4	5

Con $D = 4 \text{ KB}$, in entrambi i casi l'altezza massima per $NR = 2^{20}$ è pari a 3



Secondary B⁺-tree


■ B⁺-tree a TID

- In un B⁺-tree per chiavi secondarie, per ogni valore di chiave si ha nelle foglie una lista di puntatori (TID: **Tuple Identifier**) ai record con quel valore (questa soluzione è anche nota come **inverted index**).
- La lista di TID è normalmente mantenuta ordinata per valori crescenti, sia nel caso di indice clustered sia nel caso di indice unclustered.

■ B⁺-tree a PID

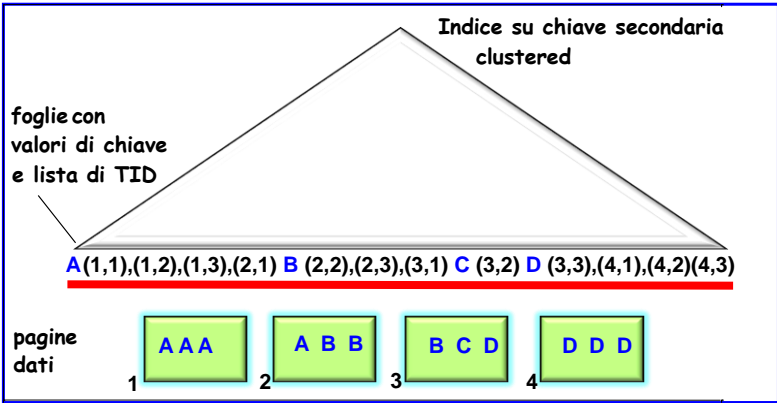
- Anziché fare uso di TID, si impiegano identificatori di pagina (PID: **Page Identifier**).
- Per un valore di chiave, la lista consiste di tanti PID quante sono le pagine del file dati che contengono almeno un record con quel valore di chiave.
- È pertanto una soluzione particolarmente conveniente nel caso in cui molti record con lo stesso valore di chiave si trovino nella stessa pagina (es. indice clustered).






Secondary clustered B⁺-tree (1)

■ B⁺-tree a TID



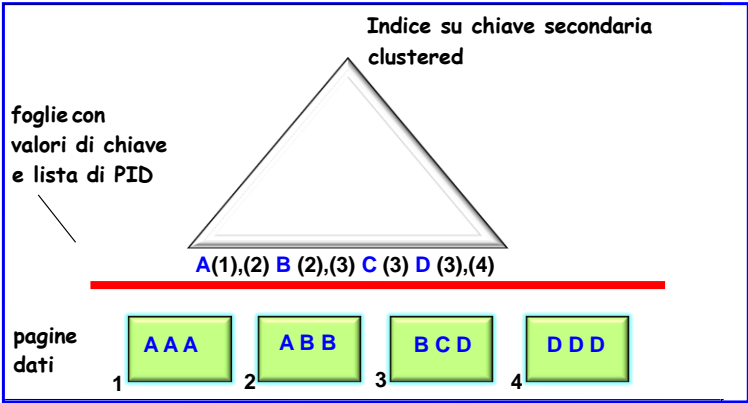
Organizzazioni dei dati- parte III

13



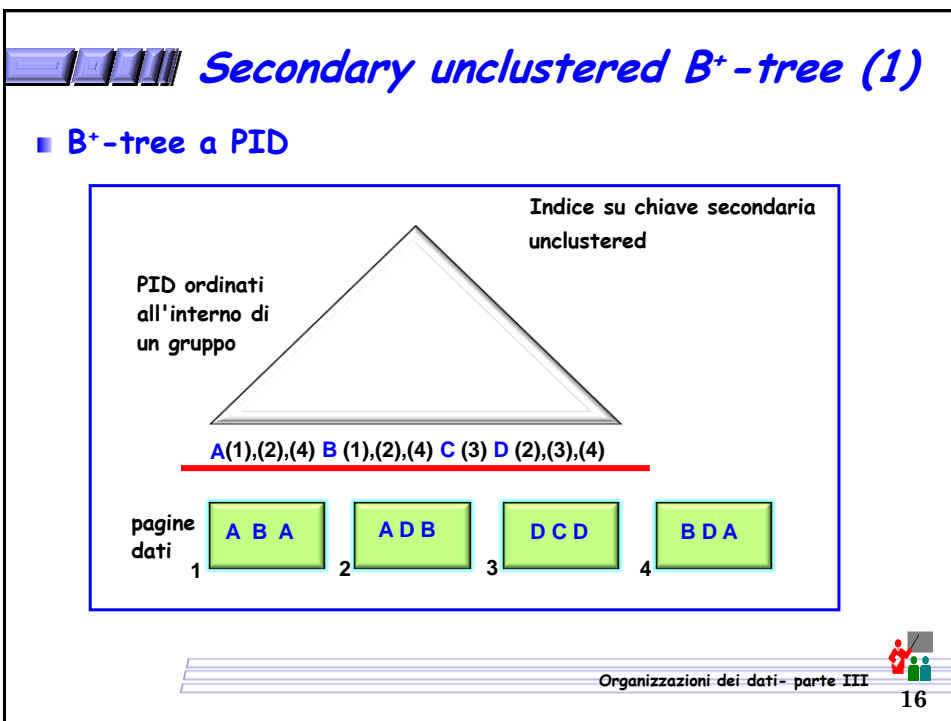
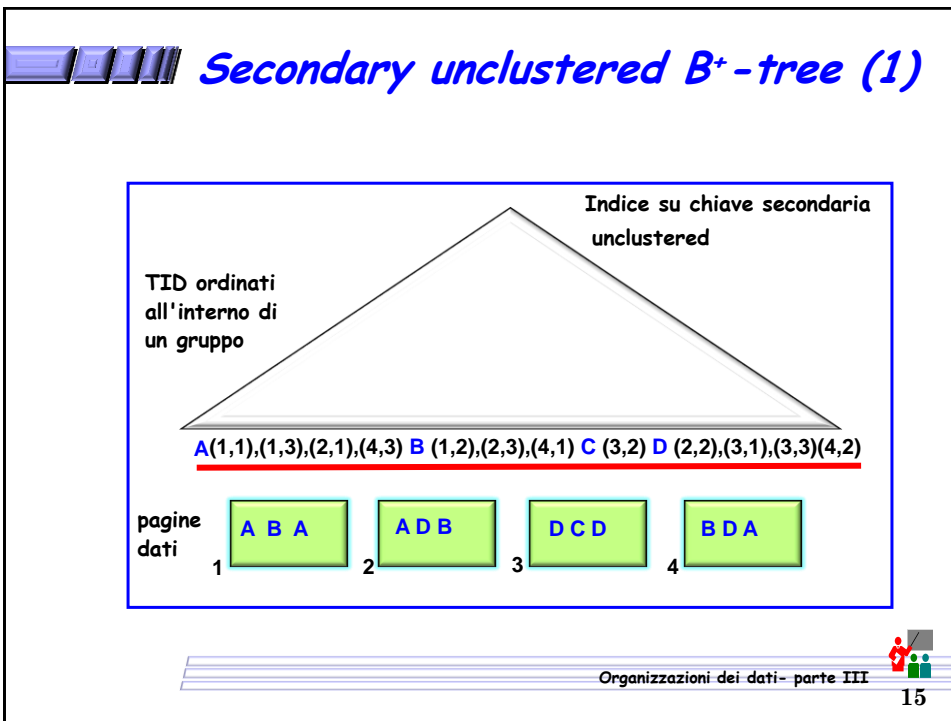
Secondary clustered B⁺-tree (2)

■ B⁺-tree a PID



Organizzazioni dei dati- parte III

14



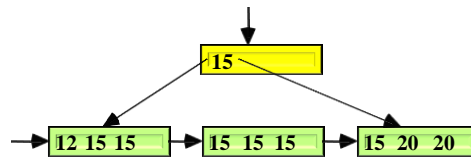


Secondary B⁺-tree: num. foglie

- (B⁺-tree a TID) Il numero delle foglie dipende ora anche dal numero di valori di chiave distinti, NK , in quanto:

$$NL = \left\lceil \frac{NK \times \text{len}(k) + NR \times \text{len}(p)}{D \times u} \right\rceil$$

- ✚ È importante considerare che una foglia deve essere indirizzata dal livello superiore solo se contiene TID di un "nuovo" valore di chiave. Per calcolare l'altezza è quindi necessario usare $\min\{NK, NL\}$ in luogo di NL .



Secondary B⁺-tree: altezza

- (B⁺-tree a TID): $D = 1024$ byte, $\text{len}(p) = 4$ byte, $NR = 10^6$

	len(k)=10 byte			len(k)=20 byte		
$\log_{10} NK$	NL	h_{\min}	h_{\max}	NL	h_{\min}	h_{\max}
1	5636	2	2	5636	2	2
2	5637	3	3	5639	3	3
3	5650	3	3	5664	3	4
4	5770	4	4	5918	4	4
5	7045	4	4	8454	4	4
6	19725	4	4	33814	4	5



Secondary B⁺-tree: posting file

- Al fine di mantenere un formato a lunghezza fissa per i nodi foglia e facilitare la gestione dell'evoluzione dell'albero, nelle foglie le liste dei TID (o dei PID) sono memorizzate in un'area separata detta "**posting file**". Le foglie del B⁺-tree contengono, per ciascun valore di chiave distinto, un puntatore alla testa della relativa lista.

$\log_{10} NK$	len(k) = 10 byte		len(k) = 20 byte	
	NL	h_{\max}	NL	h_{\max}
1	1	1	1	1
2	2	2	4	2
3	20	2	34	2
4	198	3	339	3
5	1973	3	3382	4
6	19725	4	33814	5

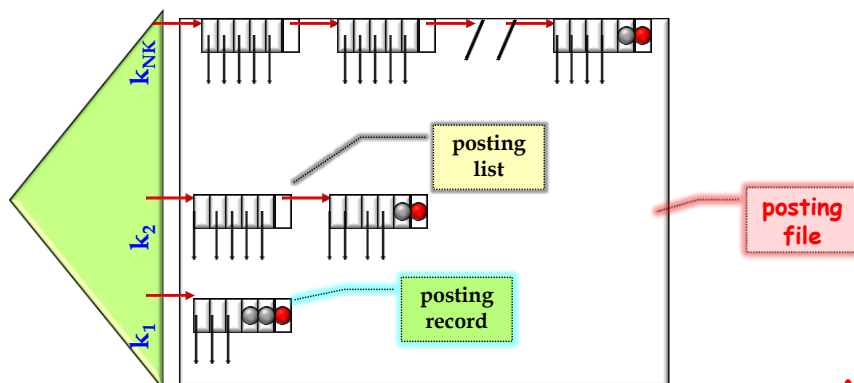
D= 1024 byte
len(p)= 4 byte
NR=10⁶

- Una soluzione intermedia prevede di mantenere nel B⁺-tree solo alcuni elementi di ogni lista, e i restanti nel posting file.



Gestione di posting file (1)

- Al momento dell'inserimento di un nuovo valore di chiave viene allocato un **posting record** di dimensione d_{\min} contenente il puntatore al record dati associato al valore di chiave. Questo record sarà poi riempito con i successivi puntatori per valori ripetuti della chiave in questione; a spazio esaurito sarà concatenato in lista a un altro nuovo posting record, e così via.





Gestione di posting file (2)

- Una possibile strategia per distribuzioni sbilanciate:

siano:

npr_{max}	numero massimo di posting record in una lista
npr	numero attuale di posting record in una lista
d_{min}	dimensione iniziale di un posting record
d	dimensione attuale di un posting record
f	fattore di crescita
f_c	fattore di crescita al momento della copia

- quando lo spazio disponibile in un posting record viene esaurito si sceglie fra due alternative:

1. se $npr_{max} > npr$ allora viene allocato un nuovo posting record di dimensione $d = d \times f$ da concatenare al precedente.

2. altrimenti viene allocato un nuovo posting record di dimensione $d = d_{min} \times F \times f_c$

in cui copiare tutti i valori finora inseriti, essendo

$$F = \frac{f^{npr_{max}} - 1}{f - 1}$$



Organizzazioni primarie basate su B- e B*-tree

- Storicamente i B-tree furono introdotti come **organizzazioni primarie, con i record dati inseriti direttamente nei nodi dell'albero**. Il vantaggio di usare i B- (e i B*-) tree come organizzazioni primarie è principalmente legato a due aspetti:

- Clustering**: il file è di fatto mantenuto ordinato sul valore di una chiave (tipicamente primaria nel caso dei B-tree). Ciò facilita ulteriormente l'elaborazione dei dati secondo valori crescenti di chiave.

- Dinamicità**: quando applicate al file dati, le procedure di gestione dei B- (B*-) tree rappresentano un modo elegante di mantenere l'ordinamento a fronte di inserimenti e cancellazioni, evitando quindi i problemi di riorganizzazione tipici di ISAM.

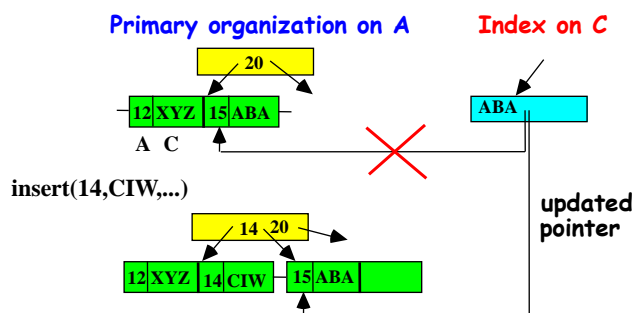
- Lo svantaggio principale, rispetto a un'organizzazione a heap, è a sua volta dovuto alla dinamicità.





Organizzazione primaria con B⁺

- Si abbia un file dati organizzato come un B⁺-tree, con i record memorizzati nelle foglie per valori crescenti del campo **A**. Si abbia inoltre un indice, organizzato sempre come B⁺-tree, sul campo **C**. Poiché il file dati evolve dinamicamente, a fronte di nuovi inserimenti che provocano split delle foglie gli indirizzi dei record cambiano. **È pertanto necessario provvedere a modificare i relativi puntatori nell'indice sul campo C.**



Organizzazioni dei dati- parte III



23



VSAM

- VSAM** (Virtual Storage Access Method), che rappresenta il risultato di un'analisi critica delle prestazioni della struttura **ISAM**, è un'organizzazione di tipo dinamico, utilizzata nei sistemi IBM OS/VS.
- VSAM si compone di un file ordinato sul valore di chiave e di un indice **non denso**, **organizzato come un B⁺-tree distribuito nel file stesso**.
- Le pagine dati sono gestite come le foglie di un B⁺-tree**, mentre nelle pagine indice sono riportate solo le coppie (valore di chiave più alto nella pagina dati, indirizzo della pagina dati).
- Secondo la terminologia VSAM un file è diviso in **regioni**, costituite da un insieme di tracce su uno o più cilindri contigui. Ogni regione è divisa in **intervalli**, consistenti di una parte di traccia o di più tracce contigue a cui si può accedere con una sola operazione di I/O. Quando un intervallo risulta pieno, viene suddiviso in due intervalli. Lo stesso accade per le regioni.

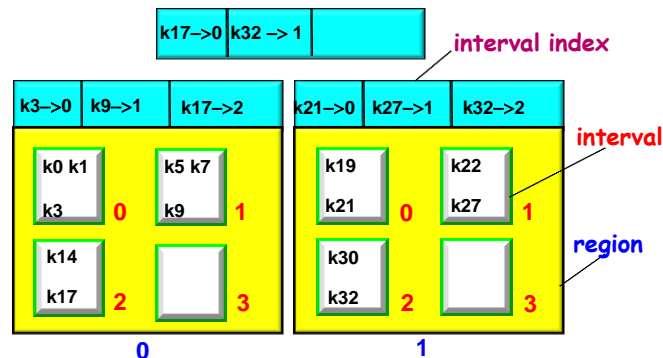
Organizzazioni dei dati- parte III



24

Esempio di organizzazione VSAM

- VSAM permette di usare un unico formato all'interno del file system. Ad esempio un VSAM file (**master catalog**) può essere usato per mantenere informazioni sul direttorio di tutti i VSAM file.



Convenienza d'uso di un indice (1)

- Benché un B⁺-tree consenta di eseguire efficacemente ricerche su valori di chiave compresi in un intervallo (**range query**), sono possibili casi in cui è da preferire la scansione sequenziale.
- Decidere se fare uso di scansione sequenziale o se accedere tramite indice è un'istanza particolare del problema dell'**ottimizzazione delle query**, il cui scopo è determinare la **strategia di esecuzione** che sia "costo" minimo (ovvero il miglior **piano di accesso**).
- Esistono diverse **metriche** per valutare la bontà di una strategia di esecuzione. In ambito DB è generalmente accettato come criterio primario (ma non sempre!) quello della **minimizzazione del numero di operazioni di I/O**.
- Avendo scelto una metrica, restano comunque da precisare alcuni aspetti importanti (presentazione del risultato di un'interrogazione, distribuzioni dei dati, ...).

Convenienza d'uso di un indice (2)

Esempio:

1. ordine dei dati in uscita (interessa ?)
2. ordinamento preventivo dei TID (viene eseguito ?)
 - 2.1 di un valore di chiave
 - 2.2 di più valori di chiave
3. distribuzione dei valori di chiave (è uniforme ?)
4. distribuzione dei record con un certo valore di chiave sulle pagine del file dati (è casuale ?)

- ✦ I punti 3. e 4. evidenziano chiaramente che, a causa della visione limitata che necessariamente si ha sul contenuto del file dati, occorre dotarsi di **modelli per la stima dei costi**.
- ✦ Se i modelli adottati ben riflettono la situazione reale, si può fare affidamento su di essi per determinare il migliore piano di accesso.
- ✦ In caso contrario, è possibile che il piano di accesso **stimato** ottimale comporti, viceversa, costi **reali** di esecuzione non minimi.



TID B⁺-tree vs sequential scan

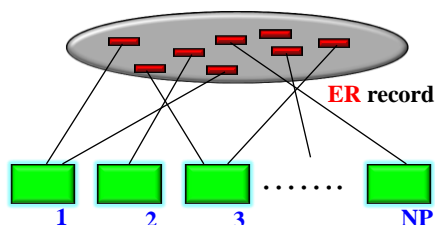
Ipotesi:

- ✦ 1. Non è richiesto nessun ordinamento dei dati in output.
 - ✦ Non devono essere considerati eventuali costi di ordinamento.
- ✦ 2.1 La lista di TID di un valore di chiave è ordinata...
 - ✦ Per reperire, via indice, tutti i record con un valore di chiave, non si accede mai più di una volta alla stessa pagina dati...
- ✦ 2.2 ... ma non si esegue il merge di più liste di TID.
 - ✦ ... ma ciò è possibile nel caso di record con valori di chiave diversi.
- ✦ 3. La distribuzione dei valori di chiave sui record è uniforme.
 - ✦ Ogni valore di chiave è ripetuto, in media, NR/NK volte.
- ✦ 4. Record con uno stesso valore di chiave sono distribuiti uniformemente (allocati casualmente) sulle pagine del file dati.
- ✦ Da 2.1 e 3. si deriva che il problema è stimare il numero (medio) di pagine che contengono almeno uno degli NR/NK record aventi un certo valore di chiave.
- ✦ L'ipotesi 4. è (molto) più forte rispetto a dire semplicemente che il file non è ordinato secondo i valori di chiavi di un certo campo. Di fatto, corrisponde ad affermare che non esiste, rispetto a tali valori, nessuna forma particolare di "clustering" dei record. Da un punto di vista probabilistico, equivale a dire che il campo in esame e quello di ordinamento sono tra loro indipendenti.



Formula di Cardenas (1)

- La formula di **Cardenas** fornisce una stima del numero medio di pagine, su un totale di **NP**, che contengono almeno uno degli **ER** record che si devono reperire.



- Il modello trova diverse applicazioni in ambito DB. Il problema astratto si può formulare come segue ($c=NP$, $e=ER$):
- Data un'urna con infinite biglie di c colori, ogni colore ripetuto un infinito numero di volte, quanti colori distinti risultano, in media, dall'estrazione di e biglie?*

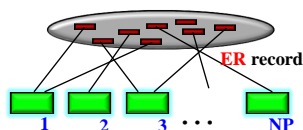
Formula di Cardenas (2)

- La formula di Cardenas si ricava considerando che:

- $1/NP$ è la probabilità che una pagina contenga uno degli **ER** record,
- $1-1/NP$ è la probabilità che non lo contenga,
- $(1-1/NP)^{ER}$ è la probabilità che non contenga nessuno degli **ER** record,
- $1-(1-1/NP)^{ER}$ è la probabilità che ne contenga almeno uno.

- Moltiplicando per il numero delle pagine si ottiene:

$$\Phi(ER, NP) = NP \times \left(1 - \left(1 - 1/NP\right)^{ER}\right) \leq \min \{ER, NP\}$$



Valutazione dei costi di accesso


- Nel problema specifico in esame, il numero di pagine che contengono almeno uno dei record con un dato valore di chiave è esprimibile come:

$$\Phi(NR/NK, NP)$$
- Se si devono reperire **EK** valori di chiave ($EK < NK$) si valutano come segue i costi di accesso (n. di operazioni di I/O):
- Scansione sequenziale (seq)** : il costo è pari al numero delle pagine del file dati (non si fa nessuna ipotesi di contiguità nell'allocazione):

$$C_a(seq) = NP$$
- Accesso con indice unclustered (uncl)**: il costo si valuta come la somma di due componenti:
- C_I costo di accesso all'indice; nell'ipotesi di ricerca di un singolo valore di chiave o di range query, è dato da una costante (numero di livelli - 1) più il numero di foglie da leggere:

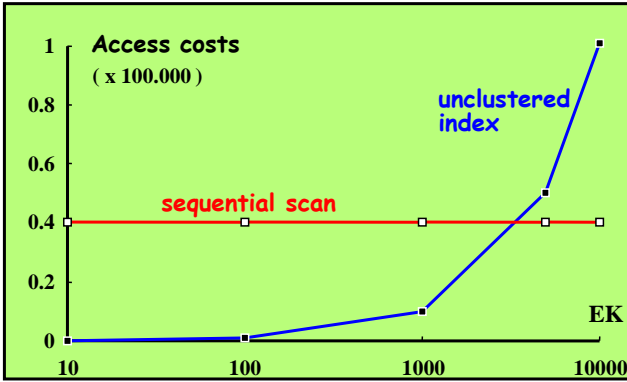
$$C_I = h - 1 + \left\lceil \frac{EK}{NK} NL \right\rceil$$
- C_D costo di accesso alle pagine dati, dato da:

$$C_D = EK \times \Phi(NR/NK, NP)$$
- Il costo complessivo è pertanto:** $C_a(uncl) = C_I + C_D$



 Organizzazioni dei dati- parte III
 31

Esempio costi di accesso

- Si abbia un file con **NR** = 10^6 record, allocati su **NP** = 40.000 pagine, e un indice unclustered su un campo con **NK** = 10^5 valori distinti. L'indice consiste di **NL** = 7045 foglie, e ha altezza **h** = 4.



EK	Sequential scan	Unclustered index
10	0.4	0.0
100	0.4	0.0
1000	0.4	0.1
10000	0.4	1.0


 Organizzazioni dei dati- parte III
 32



Osservazioni

- ✚ L'altezza effettiva dell'albero è, in analisi di questo tipo, del tutto ininfluyente. È usuale porre il termine costante pari a 2, considerando cioè $h = 4$ e assumendo la radice in memoria.
- ✚ In molti testi, lavori, ecc., è possibile trovare arrotondamenti all'intero superiore dei valori forniti dalla formula di Cardenas. Benché un numero non intero di accessi non abbia significato per la singola operazione, **non bisogna dimenticare che la formula fornisce un valor medio**, e quindi un valore non intero è del tutto lecito.
- ✚ Viceversa, non arrotondare il valore $NL \times EK/NK$ è concettualmente un errore, in quanto, con le ipotesi fatte, il valor medio del numero di foglie cui bisogna accedere si ricava proprio arrotondando.

$NK=3$
 $NL=4$

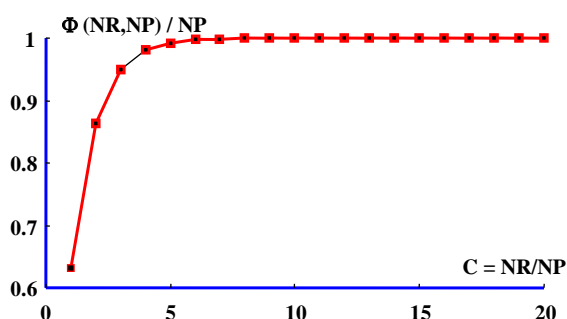
Con $EK=2$ si accede sempre a $3 = \lceil 4 \times 2 / 3 \rceil$ foglie.



Limiti del modello di Cardenas

- Il modello di Cardenas assume pagine di **capacità infinita** (si noti che NR non figura come argomento) e porta a **sottostimare** apprezzabilmente il valore corretto **nel caso di pagine con meno di circa 10 record**. Quando $ER = NR$, la formula restituisce **un valore minore di NP** , pari a:

$$\Phi(NR, NP) = NP \left(1 - \left(1 - 1/NP \right)^{NR} \right) \approx NP \left(1 - e^{-NR/NP} \right)$$



l'approssimazione
vale per NP grande

Utilizzo di più indici (1)

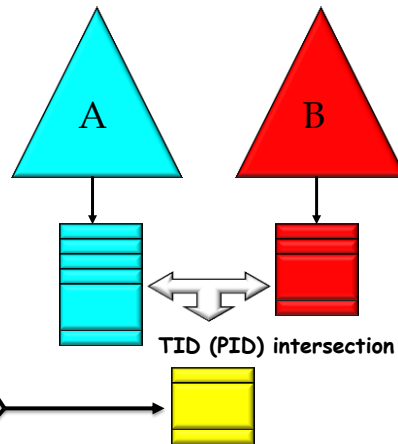
- Per risolvere interrogazioni complesse in alcuni RDBMS è possibile utilizzare più indici, tramite algoritmi di **intersezione** e **unione** di TID (PID).



```
select <...> from R
where <pred. on A>
and <pred on B>
```

TID: lista di puntatori a record che soddisfano entrambi i predicati

PID: lista di puntatori a blocchi che contengono almeno un record che soddisfa il predicato su A e almeno un record che soddisfa il predicato su B



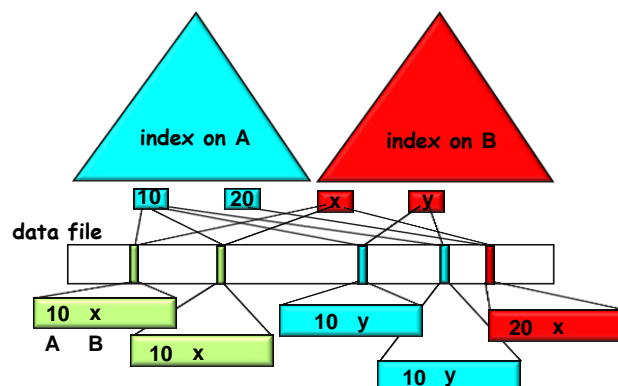
Organizzazioni dei dati- parte III



37

Utilizzo di più indici (2)

- Esempio: intersezione di TID con predicato **A=10** and **B=x**



Organizzazioni dei dati- parte III



38

Utilizzo di più indici (3)

- Se gli indici sono a PID l'intersezione dei PID può dar luogo a "**false drop**", ovvero accesso a pagine che non contengono record che soddisfano l'intero predicato.
- L'utilizzo contemporaneo di più indici per accedere ai dati può risultare conveniente rispetto all'utilizzo di un solo indice, ma occorre considerare, tra le altre cose:
 - ✚ il risparmio di accessi al file dati;
 - ✚ il costo aggiuntivo di accesso agli indici;
 - ✚ il costo per calcolare l'unione/intersezione dei TID.
- Un approccio meno "flessibile" per risolvere interrogazioni su più campi consiste nel definire **indici su combinazioni di attributi**.

Indici su combinazioni di attributi (1)

- Un indice sulla combinazione ordinata di attributi (A_1, A_2, \dots, A_n) , è un tipo particolare di organizzazione secondaria **multi-attributo** che memorizza come valori di chiave tutte le combinazioni distinte di valori di tali attributi presenti nel file dati.
- Combinando **n** attributi, sono possibili **n!** distinti indici, che si differenziano per l'ordine in cui gli attributi sono considerati.

Esempio: dato il file

A	B				
7	abc	7	fgh	2	cde
5	cde	3	fgh	5	cde
				2	abc
				7	fgh

un indice sulla combinazione di attributi **A** e **B** contiene come valori di chiave 6 coppie; i due possibili indici sono:

- ❖ **(A,B):** (2,abc),(2,cde),(3,fgh),(5,cde),(7,abc),(7,fgh)
- ❖ **(B,A):** (abc,2),(abc,7),(cde,2),(cde,5),(fgh,3),(fgh,7)

Indici su combinazioni di attributi (2)

- I principali vantaggi di questo tipo di organizzazione, rispetto a **n** indici su singolo attributo, sono:
 - riduzione del numero di TID di un fattore **n**;
 - efficienza di elaborazione di interrogazioni che specificano condizioni sui **primi** j attributi ($1 \leq j \leq n$);
 - efficienza nell'aggiornamento.
- Il problema principale, che spesso vanifica tutti i vantaggi elencati, è dato dall'inefficienza nell'elaborazione di interrogazioni che specificano condizioni sugli **ultimi** k attributi ($1 \leq k \leq n$), a causa dell'assenza di contiguità dei valori di chiave, che può portare a dover leggere tutte le foglie.
- Esistono indici multi-attributo che non hanno questo problema, e trattano in maniera **simmetrica** gli attributi.

Organizzazioni dei dati - parte III

41

Bit-mapped indexing

- Metodo di accesso applicabile nel caso di attributi con valori ripetuti e in presenza di più indici (**usato in Teradata**). Ad esempio:

select * from R
where $A_i=1$
and $A_j=3$
and $A_k=7$

R

.....	A_i	A_j	A_k
.....	1	3	6
.....	1	5	6
.....	1	3	7
.....	4	3	6

TID

a
b
c
d

da ogni indice si
 ricava una bit map

$A_i=1$
 $A_j=3$
 $A_k=7$
 and

a	b	c	d
1	1	1	0
1	0	1	1
0	0	1	0
0	0	1	0

unica tupla che si
 qualifica

Organizzazioni dei dati - parte III

42