

**Basi di dati Vol.2**  
**Capitolo 1**  
**Organizzazione fisica e**  
**gestione delle interrogazioni**

03/05/2006

**Tecnologia delle BD: perché studiarla?**

- I DBMS offrono i loro servizi in modo "trasparente":
  - per questo abbiamo potuto finora ignorare molti aspetti realizzativi
- Abbiamo considerato il DBMS come una "scatola nera"
- Perché aprirla?
  - capire come funziona può essere utile per un migliore utilizzo
  - alcuni servizi sono offerti separatamente

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

2

**DataBase Management System — DBMS**

Sistema (prodotto software) in grado di gestire collezioni di dati che siano (anche):

- **grandi** (di dimensioni (molto) maggiori della memoria centrale dei sistemi di calcolo utilizzati)
- **persistenti** (con un periodo di vita indipendente dalle singole esecuzioni dei programmi che le utilizzano)
- **condivise** (utilizzate da applicazioni diverse)

garantendo **affidabilità** (resistenza a malfunzionamenti hardware e software) e **privatizza** (con una disciplina e un controllo degli accessi). Come ogni prodotto informatico, un DBMS deve essere **efficiente** (utilizzando al meglio le risorse di spazio e tempo del sistema) ed **efficace** (rendendo produttive le attività dei suoi utilizzatori).

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

3

**Le basi di dati sono grandi e persistenti**

- La persistenza richiede una gestione in memoria secondaria
- La grandezza richiede che tale gestione sia sofisticata (non possiamo caricare tutto in memoria principale e poi riscaricare)

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

4

**Le basi di dati vengono interrogate ...**

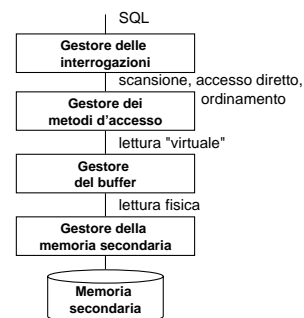
- Gli utenti vedono il modello logico (relazionale)
- I dati sono in memoria secondaria
- Le strutture logiche non sarebbero efficienti in memoria secondaria:
  - servono strutture fisiche opportune
- La memoria secondaria è molto più lenta della memoria principale:
  - serve un'interazione fra memoria principale e secondaria che limiti il più possibile gli accessi alla secondaria
- Esempio: una interrogazione con un join

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

5

**Gestore degli accessi e delle interrogazioni**



03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

6

### Le basi di dati sono affidabili

- Le basi di dati sono una risorsa per chi le possiede, e debbono essere conservate anche in presenza di malfunzionamenti
- Esempio:
  - un trasferimento di fondi da un conto corrente bancario ad un altro, con guasto del sistema a metà
- Le **transazioni** debbono essere
  - atomiche (o tutto o niente)
  - definitive: dopo la conclusione, non si dimenticano

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

7

### Le basi di dati vengono aggiornate ...

- L'**affidabilità** è impegnativa per via degli aggiornamenti frequenti e della necessità di gestire il buffer

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

8

### Le basi di dati sono condivise

- Una base di dati è una risorsa **integrata, condivisa** fra le varie applicazioni
- conseguenze
  - Attività diverse su dati in parte condivisi:
    - meccanismi di **autorizzazione**
  - Attività multi-utente su dati condivisi:
    - controllo della **concorrenza**

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

9

### Aggiornamenti su basi di dati condivise ...

- Esempi:
  - due prelevamenti (quasi) contemporanei sullo stesso conto corrente
  - due prenotazioni (quasi) contemporanee sullo posto
- Intuitivamente, le transazioni sono corrette se **seriali** (prima una e poi l'altra)
- Ma in molti sistemi reali l'efficienza sarebbe penalizzata troppo se le transazioni fossero seriali:
  - il **controllo della concorrenza** permette un ragionevole compromesso

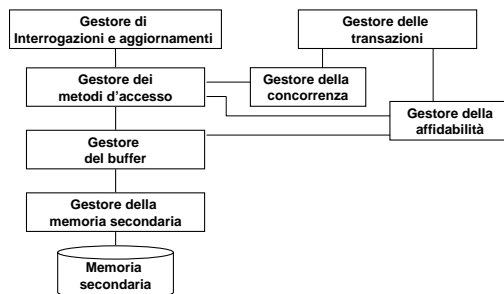
03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

10

### Gestore degli accessi e delle interrogazioni

### Gestore delle transazioni



03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

11

### Tecnologia delle basi di dati, argomenti

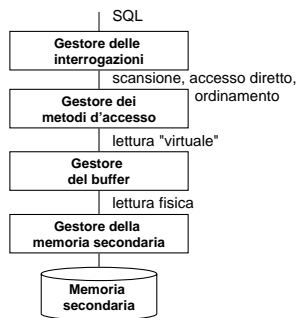
- Gestione della memoria secondaria e del buffer
- Organizzazione fisica dei dati
- Gestione ("ottimizzazione") delle interrogazioni
- Controllo della affidabilità
- Controllo della concorrenza
  
- Architetture distribuite

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

12

## Gestore degli accessi e delle interrogazioni

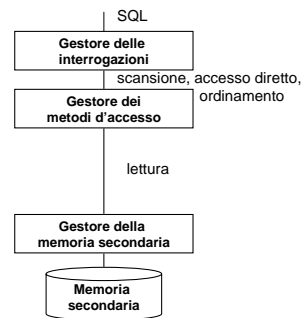


03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

13

## Gestore degli accessi e delle interrogazioni, semplifichiamo, per ora



03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

14

## Memoria principale e secondaria

- I programmi possono fare riferimento solo a dati in memoria principale
- Le basi di dati debbono essere (sostanzialmente) in memoria secondaria per due motivi:
  - dimensioni
  - persistenza
- I dati in memoria secondaria possono essere utilizzati solo se prima trasferiti in memoria principale (questo spiega i termini "principale" e "secondaria")

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

15

## Memoria principale e secondaria, 2

- I dispositivi di memoria secondaria sono organizzati in **blocchi** di lunghezza (di solito) **fissa** (ordine di grandezza: alcuni KB)
- Le uniche operazioni sui dispositivi sono la lettura e la scrittura di di una **pagina**, cioè dei dati di un blocco (cioè di una stringa di byte);
- per comodità consideriamo **blocco** e **pagina** sinonimi

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

16

## Memoria principale e secondaria, 3

- Accesso a memoria secondaria (dal sito della Seagate, 2005):
  - tempo di **posizionamento della testina (seek time)**: in media 3-15ms (a seconda del tipo di disco), migliora del 7-10% all'anno
  - tempo di **latenza (rotational delay)**: 2-6ms (conseguenza della velocità di rotazione, 4-15K giri al minuto, migliora del 7-10% all'anno)
  - tempo di **trasferimento** di un blocco: frazioni di ms (conseguenza della velocità di trasferimento, 100-300MB/s, migliora del 40-50% all'anno)
 in media non meno di qualche ms
- Il costo di un accesso a memoria secondaria è quattro o più ordini di grandezza maggiore di quello per operazioni in memoria centrale
- Perciò, nelle applicazioni "I/O bound" (cioè con molti accessi a memoria secondaria e relativamente poche operazioni) il costo dipende esclusivamente dal numero di accessi a memoria secondaria
- Inoltre, accessi a blocchi "vicini" costano meno (**contiguità**)

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

17

## DBMS e file system

- Il file system è il componente del sistema operativo che gestisce la memoria secondaria
- I DBMS ne utilizzano le funzionalità, ma in misura limitata, per creare ed eliminare file e per leggere e scrivere singoli blocchi o sequenze di blocchi contigui.
- L'organizzazione dei file, sia in termini di distribuzione dei record nei blocchi sia relativamente alla struttura all'interno dei singoli blocchi è gestita direttamente dal DBMS.

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

18

## DBMS e file system, 2

- Il DBMS gestisce i blocchi dei file allocati come se fossero un unico grande spazio di memoria secondaria e costruisce, in tale spazio, le strutture fisiche con cui implementa le relazioni.
- Il DBMS crea file di grandi dimensioni che utilizza per memorizzare diverse relazioni (al limite, l'intera base di dati)
- Talvolta, vengono creati file in tempi successivi:
  - è possibile che un file contenga i dati di più relazioni e che le varie tuple di una relazione siano in file diversi.
- Spesso, ma non sempre, ogni blocco è dedicato a ennuple di un'unica relazione

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

19

## Blocchi e record

- I blocchi (componenti "fisici" di un file) e i record (componenti "logici") hanno dimensioni in generale diverse:
  - la dimensione del blocco dipende dal file system
  - la dimensione del record (semplificando un po') dipende dalle esigenze dell'applicazione, e può anche variare nell'ambito di un file

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

20

## Fattore di blocco

- numero di record in un blocco
  - $L_R$ : dimensione di un record (per semplicità costante nel file: "record a lunghezza fissa")
  - $L_B$ : dimensione di un blocco
  - se  $L_B > L_R$ , possiamo avere più record in un blocco:
 
$$\lfloor L_B / L_R \rfloor$$
- lo spazio residuo può essere
  - utilizzato (record "spanned" o impaccati)
  - non utilizzato ("unspanned")

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

21

## Organizzazione delle ennuple nelle pagine

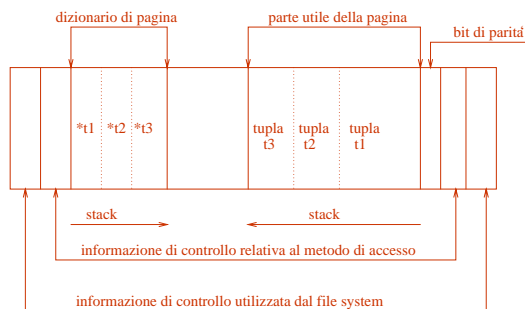
- Ci sono varie alternative, anche legate alle specifiche strutture fisiche; vediamo una possibilità
- Inoltre:
  - se la lunghezza delle ennuple è fissa, la struttura può essere semplificata
  - le ennuple (come visto, caso "spanned") possono essere su più pagine (necessario per ennuple grandi)

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

22

## Organizzazione delle ennuple nelle pagine



03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

23

## Strutture

- Sequenziali
- Calcolate ("Hash")
- Ad albero (di solito, indici)

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

24

## Strutture sequenziali

- Esiste un ordinamento fra le ennuple, che può essere rilevante ai fini della gestione
  - **seriale**: ordinamento fisico ma non logico
  - **array**: posizioni individuate attraverso indici
  - **ordinata**: l'ordinamento delle tuple coerente con quello di un campo

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

25

## Struttura seriale

- Chiamata anche:
  - "Entry sequenced"
  - file heap
  - file disordinato
- È molto diffusa nelle basi di dati relazionali, associata a indici secondari
- Gli inserimenti vengono effettuati (varianti)
  - in coda (con riorganizzazioni periodiche)
  - al posto di record cancellati
- La gestione è molto semplice da certi punti di vista, ma in molti casi inefficiente

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

26

## Array sequential structure

- Possible only when the tuples are of fixed length
- Made of  $n$  of adjacent blocks, each block with  $m$  of available slots for tuples
- Each tuple has a numeric index  $i$  and is placed in the  $i$ -th position of the array
- Primitives:
  - Accessed via `read-ind` (at a given index value).
  - Data loading happens at the end of the file (indices are obtained simply by increasing a counter)
  - Deletions create free slots
  - Updates are done on place

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

27

## Ordered sequential structure

- Each tuple has a position based on the value of a "key" (or "pseudo-key") field
- Historically, ordered sequential structures were used on sequential devices (tapes) by batch processes. Data were located into the *main file*, modifications were collected in *differential files*, and the files were *periodically merged*. This has fallen out of use
- The main problems: insertions or updates which increase the physical space - they require reorganizations
- Options to avoid global reorderings:
  - Leaving a certain number of slots free at the time of first loading. This is followed by 'local reordering' operations
  - Integrating the sequentially ordered files with an *overflow file*, where new tuples are inserted into blocks linked to form an *overflow chain*

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

28

## Strutture ordinate

- Permettono ricerche binarie, ma solo fino ad un certo punto (come troviamo la "metà" del file?)
- Nelle basi di dati relazionali si utilizzano quasi solo in combinazione con indici (file ISAM o file ordinati con indice primario)

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

29

## File hash

- Permettono un accesso diretto molto efficiente (da alcuni punti di vista)
- La tecnica si basa su quella utilizzata per le tavole hash in memoria centrale

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

30

## Tavola hash

- Obiettivo: accesso diretto ad un insieme di record sulla base del valore di un campo (detto **chiave**, che per semplicità supponiamo identificante, ma non è necessario)
- Se i possibili valori della chiave sono in numero paragonabile al numero di record (e corrispondono ad un "tipo indice") allora usiamo un array; ad esempio: università con 1000 studenti e numeri di matricola compresi fra 1 e 1000 o poco più e file con tutti gli studenti
- Se i possibili valori della chiave sono molti di più di quelli effettivamente utilizzati, non possiamo usare l'array (spreco); ad esempio:
  - 40 studenti e numero di matricola di 6 cifre (un milione di possibili chiavi)

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

31

## Tavola hash, 2

- Volendo continuare ad usare qualcosa di simile ad un array, ma senza sprecare spazio, possiamo pensare di trasformare i valori della chiave in possibili indici di un array:
  - **funzione hash:**
    - associa ad ogni valore della chiave un "indirizzo", in uno spazio di dimensione paragonabile (leggermente superiore) rispetto a quello strettamente necessario
    - poiché il numero di possibili chiavi è molto maggiore del numero di possibili indirizzi ("lo spazio delle chiavi è più grande dello spazio degli indirizzi"), la funzione non può essere iniettiva e quindi esiste la possibilità di collisioni (chiavi diverse che corrispondono allo stesso indirizzo)
    - le buone funzioni hash distribuiscono in modo casuale e uniforme, riducendo le probabilità di collisione (che si riduce aumentando lo spazio ridondante)

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

32

## Un esempio

- 40 record
- tavola hash con 50 posizioni:
  - 1 collisione a 4
  - 2 collisioni a 3
  - 5 collisioni a 2

M	M mod 50	M	M mod 50
60600	0	200268	18
66301	1	205619	19
205751	1	210522	22
205802	2	205724	24
200902	2	205977	27
116202	2	205478	28
200604	4	200430	30
66005	5	210533	33
116455	5	205887	37
200205	5	200138	38
201159	9	102338	38
205610	10	102690	40
201260	10	115541	41
102360	10	206092	42
205460	10	205693	43
205912	12	205845	45
205762	12	200296	46
200464	14	205796	46
205617	17	200498	48
205667	17	206049	49

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

33

## Tavola hash

0	60600			40	102690	1	205751
1	66301			41	115541	2	200902
2	205802	22	210522	42	206092	2	116202
				43	205693	5	116455
4	200604	24	205724			5	200205
5	66005			45	205845	10	201260
				46	205796	10	102360
						10	205460
		27	205977	48	200498	12	205762
9	201159	28	205478	49	206049	17	205667
10	205610	30	200430			38	200138
						46	200296
12	205912						
		33	210533				
14	200464						
17	205617	37	205887				
18	200268	38	102338				
19	205619						

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

34

## Tavola hash, collisioni

- Varie tecniche:
  - posizioni successive disponibili
  - tabella di overflow (gestita in forma collegata)
  - funzioni hash "alternative"
- Nota:
  - le collisioni ci sono (quasi) sempre
  - le collisioni multiple hanno probabilità che decresce al crescere della molteplicità
  - la molteplicità media delle collisioni è molto bassa

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

35

## File hash

- L'idea è la stessa della tavola hash, ma si basa sull'organizzazione in blocchi:
  - ogni blocco contiene più record
  - Lo spazio degli indirizzi è più piccolo
    - Nell'esempio, con fattore di blocco pari a 10, possiamo usare "mod 5" invece di "mod 50"

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

36

### Un esempio

- 40 record
- tavola hash con 50 posizioni:
  - 1 collisione a 4
  - 2 collisioni a 3
  - 5 collisioni a 2
- numero medio di accessi: 1,425

M	M mod 50	M	M mod 50
60600	0	200268	18
66301	1	205619	19
205751	1	210522	22
205802	2	205724	24
209902	2	205977	27
116202	2	205478	28
200604	4	200430	30
66005	5	210533	33
116455	5	205887	37
200205	5	200138	38
201159	9	102338	38
205610	10	102690	40
201260	10	115541	41
102360	10	206092	42
205460	10	205693	43
205912	12	205845	45
205762	12	200296	46
200464	14	205796	46
205617	17	200498	48
205667	17	206049	49

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

37

### Un file hash

0	1	2	3	4
60600	66301	205802	200268	200604
66005	205751	200902	205478	201159
116455	115541	116202	210533	200464
200205	200296	205912	200138	205619
205610	205796	205762	102338	205724
201260		205617	205693	206049
102360		205667	200498	
205460		210522		
200430		205977		
102690		205887		
205845		206092		

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

38

### Nell'esempio

- 40 record
- tavola hash con 50 posizioni:
  - 1 collisione a 4
  - 2 collisioni a 3
  - 5 collisioni a 2
- numero medio di accessi: 1,425
- file hash con fattore di blocco 10; 5 blocchi con 10 posizioni ciascuno:
  - due soli overflow (blocchi con più di 10 record)
- numero medio di accessi: 1,05
- Perché?

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

39

### Collisioni, stima

- Lunghezza media delle catene di overflow, al variare di
  - Fattore di blocco: F
  - Coefficiente di riempimento: T/(FxB)

	1	2	3	5	10	(F)
.5	0.5	0.177	0.087	0.031	0.005	
.6	0.75	0.293	0.158	0.066	0.015	
.7	1.167	0.494	0.286	0.136	0.042	
.8	2.0	0.903	0.554	0.289	0.110	
.9	4.495	2.146	1.377	0.777	0.345	
T/(FxB)						

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

40

### File hash, osservazioni

- Le collisioni (overflow) sono di solito gestite con blocchi collegati
- È l'organizzazione più efficiente per l'accesso diretto basato su valori della chiave con condizioni di uguaglianza (accesso puntuale):
  - costo medio di poco superiore all'unità (il caso peggiore è molto costoso ma talmente improbabile da poter essere ignorato)
- I file hash "degenerano" se si riduce lo spazio sovrabbondante: funzionano solo con file la cui dimensione non varia molto nel tempo
- Non è efficiente per ricerche basate su intervalli (né per ricerche basate su altri attributi)

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

41

### Indici di file

- Indice:
  - struttura ausiliaria per l'accesso (efficiente) ai record di un file sulla base dei valori di un campo (o di una "concatenazione di campi") detto chiave (o, meglio, pseudochiave, perché non è necessariamente identificante);
- Idea fondamentale: l'indice analitico di un libro: lista di coppie (termine, pagina), ordinata alfabeticamente sui termini, posta in fondo al libro e separabile da esso
- Un indice I di un file f è un altro file, con record a due campi: chiave e indirizzo (dei record di f o dei relativi blocchi), ordinato secondo i valori della chiave

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

42

## Tipi di indice

- indice primario:
  - su un campo sul cui ordinamento è basata la memorizzazione (detti anche indici di cluster, anche se talvolta si chiamano primari quelli su una chiave identificante e di cluster quelli su una pseudochiave non identificante)
- indice secondario
  - su un campo con ordinamento diverso da quello di memorizzazione

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

43

## Tipi di indice, commenti

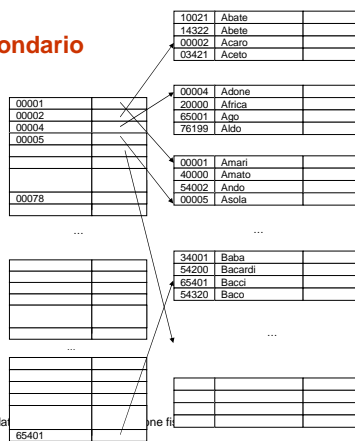
- Esempio, sempre rispetto ad un libro
  - indice generale
  - indice analitico
- I benefici legati alla presenza di indici secondari sono molto più sensibili
- Ogni file può avere al più un indice primario e un numero qualunque di indici secondari (su campi diversi). Esempio:
  - una guida turistica può avere l'indice dei luoghi e quello degli artisti
- Un file hash non può avere un indice primario

03/05/2006

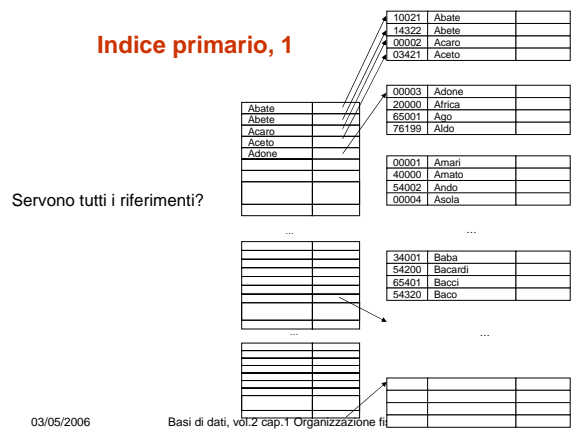
Basi di dati, vol.2 cap.1 Organizzazione fisica

44

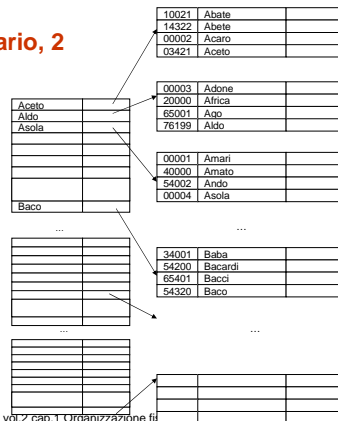
## Indice secondario



## Indice primario, 1



## Indice primario, 2



## Tipi di indice, ancora

- indice denso:
  - contiene tutti i valori della chiave e quindi un riferimento per ciascun record del file
- indice sparso:
  - contiene solo alcuni valori della chiave e quindi un numero di riferimenti inferiore rispetto ai record del file
- Un indice primario
  - di solito è sparso
  - denso permette di eseguire operazioni sugli indirizzi, senza accedere ai record
- Un indice secondario deve essere denso

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

48



## Dimensioni dell'indice

- L numero di record nel file
- B dimensione dei blocchi
- R lunghezza dei record (fissa)
- K lunghezza del campo chiave
- P lunghezza degli indirizzi (ai blocchi)

N. di blocchi per il file (circa):  $N_F = L / (B/R)$   
N. di blocchi per un indice denso:  $N_D = L / (B/(K+P))$   
N. di blocchi per un indice sparso:  $N_S = N_F / (B/(K+P))$

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

49

## Dimensioni dell'indice, esempio

- L numero di record nel file 1.000.000
- B dimensione dei blocchi 4KB
- R lunghezza dei record (fissa per semplicità) 100B
- K lunghezza del campo chiave 4B
- P lunghezza degli indirizzi (ai blocchi) 4B

$N_F = L / (B/R) \quad \approx 1.000.000 / (4.000/100) = 25.000$   
 $N_D = L / (B/(K+P)) \quad \approx 1.000.000 / (4.000/8) = 2.000$   
 $N_S = N_F / (B/(K+P)) \quad \approx 25.000 / (4.000/8) = 50$

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

50

## Caratteristiche degli indici

- Accesso diretto (sulla chiave) efficiente, sia puntuale sia per intervalli
- Scansione sequenziale ordinata efficiente
- Modifiche della chiave, inserimenti, eliminazioni inefficienti (come nei file ordinati)
  - tecniche per alleviare i problemi:
    - file o blocchi di overflow
    - marcatura per le eliminazioni
    - riempimento parziale
    - blocchi collegati (non contigui)
    - riorganizzazioni periodiche
    - ... (vedremo più avanti)

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

51

## Indici secondari, due osservazioni

- Si possono usare, come detto, puntatori ai blocchi oppure puntatori ai record
  - I puntatori ai blocchi sono più compatti
  - I puntatori ai record permettono di
    - semplificare alcune operazioni (effettuate solo sull'indice, senza accedere al file se non quando indispensabile)
- Tutti gli indici (in particolare quelli secondari) forniscono un **ordinamento logico** sui record del file; con numero di accessi pari al numero di record del file (a parte qualche beneficio dovuto alla bufferizzazione)

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

52

## Indici su campi non chiave

- Ci sono (in generale) più record per un valore della (pseudo)chiave
  - primario sparso, possibili semplificazioni:
    - puntatori solo a blocchi con valori "nuovi"
  - primario denso:
    - una coppia con valore della chiave e riferimento per ogni record (quindi i valori della chiave si ripetono)
    - valore della chiave seguito dalla lista di riferimenti ai record con quel valore
    - valore della chiave seguito dal riferimento al primo record con quel valore (perde i benefici dell'indice primario denso legati alla possibilità di lavorare sui puntatori)
  - secondario (denso):
    - una coppia con valore della chiave e riferimento per ogni record (quindi i valori della chiave si ripetono)
    - un livello (di "indirizione") in più: per ogni valore della chiave l'indice contiene un record con riferimento al blocco di una struttura intermedia che contiene riferimenti ai record

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

53

## Tipi di indice per indici su campi non chiave

- indice denso:
  - contiene tutti i valori della chiave
- indice sparso:
  - contiene solo alcuni valori della chiave

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

54

## Indici multilivello

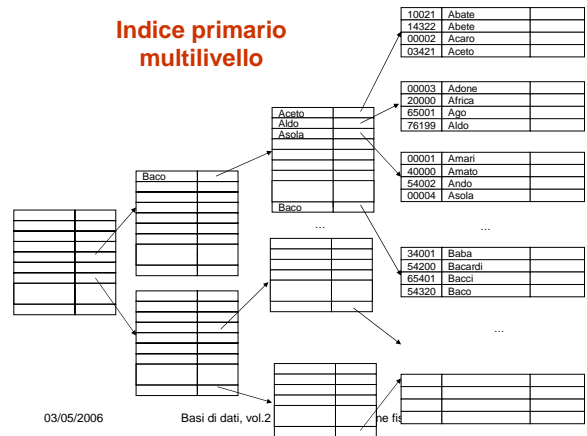
- Gli indici sono file essi stessi e quindi ha senso costruire indici sugli indici, per evitare di fare ricerche fra blocchi diversi (che potrebbero richiedere scansioni sequenziali)
- L'indice è ordinato e quindi l'indice sull'indice è primario (e sparso)
- Il tutto a più livelli, fino ad avere un livello con un solo blocco

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

55

## Indice primario multilivello

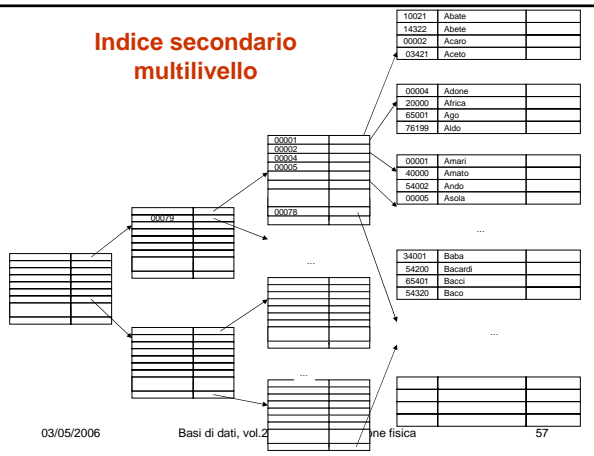


03/05/2006

Basi di dati, vol.2

Indice fis

## Indice secondario multilivello



03/05/2006

Basi di dati, vol.2

Organizzazione fisica

57

## Indici multilivello

- I livelli sono di solito abbastanza pochi, perché
  - l'indice è ordinato, quindi l'indice sull'indice è sparso
  - i record dell'indice sono piccoli
- $N_j$  numero di blocchi al livello  $j$  dell'indice (circa):
  - $N_j = N_{j-1} / (B/(K+P))$
- Negli esempi numerici  $B/(K+P) = 4.000/8 = 500$ 
  - Denso:  $N_1 = 2.000, N_2 = 4, N_3 = 1$
  - Sparso:  $N_1 = 50, N_2 = 1$

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

58

## Indici, problemi

- Tutte le strutture di indice viste finora sono basate su strutture ordinate e quindi sono poco flessibili in presenza di elevata dinamicità
- Gli indici utilizzati dai DBMS sono più sofisticati:
  - indici dinamici multilivello: B-tree (intuitivamente: alberi di ricerca bilanciati)
    - Arriviamo ai B-tree per gradi
      - Alberi binari di ricerca
      - Alberi n-ari di ricerca
      - Alberi n-ari di ricerca bilanciati

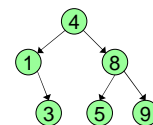
03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

59

## Albero binario di ricerca

- Albero binario etichettato in cui per ogni nodo il sottoalbero sinistro contiene solo etichette minori di quella del nodo e il sottoalbero destro etichette maggiori
- tempo di ricerca (e inserimento), pari alla profondità:
  - logaritmico nel caso "medio" (assumendo un ordine di inserimento casuale)



03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

60

## Albero di ricerca di ordine P

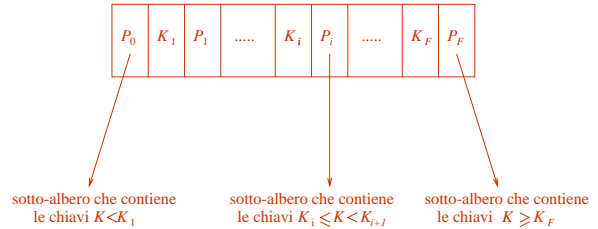
- Ogni nodo ha (fino a) P figli e (fino a) P-1 etichette, ordinate
- Nell'i-esimo sottoalbero abbiamo tutte etichette maggiori della (i-1)-esima etichetta e minori della i-esima
- Ogni ricerca o modifica comporta la visita di un cammino radice foglia
- In strutture fisiche, un nodo corrisponde di solito ad un blocco e quindi ogni nodo intermedio ha molti figli (un "fan-out" molto grande, pari al fattore di blocco dell'indice)
- All'interno di un nodo, la ricerca è sequenziale (ma in memoria centrale!)
- La struttura è ancora (potenzialmente) rigida

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

61

## Nodi in un albero di ricerca di ordine F+1



03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

62

## B-tree

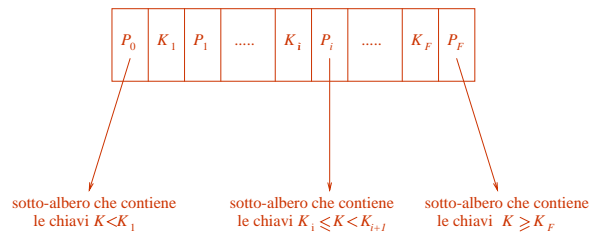
- Albero di ricerca in cui ogni nodo corrisponde ad un blocco, viene mantenuto bilanciato, grazie a:
  - Riempimento parziale (mediamente 70%)
  - Riorganizzazioni (locali) in caso di sbilanciamento

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

63

## Organizzazione dei nodi del B-tree



03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

64

## Split e merge

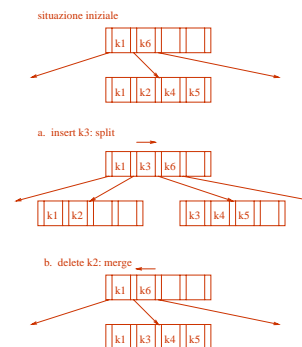
- Inserimenti ed eliminazioni sono precedute da una ricerca fino ad una foglia
- Per gli inserimenti, se c'è posto nella foglia, ok, altrimenti il nodo va suddiviso, con necessità di un puntatore in più per il nodo genitore; se non c'è posto, si sale ancora, eventualmente fino alla radice. Il riempimento rimane sempre superiore al 50%
- Dualmente, le eliminazioni possono portare a riduzioni di nodi
- Modifiche del campo chiave vanno trattate come eliminazioni seguite da inserimenti

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

65

## Split and merge operations



03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

66

## B tree e B+ tree

- B+ tree:
  - le chiavi compaiono tutte nelle foglie (e quindi quelle nei nodi intermedi sono comunque ripetute nelle foglie)
  - le foglie sono collegate in una lista
  - ottimi per le ricerche su intervalli
  - molto usati nei DBMS
- B tree:
  - Le chiavi che compaiono nei nodi intermedi non sono ripetute nelle foglie

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

67

## B tree e B+ tree, primari e secondari

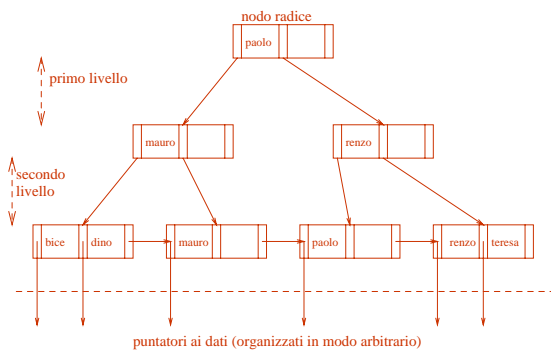
- In un B+-tree
  - primario, le ennuple possono essere contenute nelle foglie
  - secondario, le foglie contengono puntatori alle ennuple
- In un B-tree
  - anche i nodi intermedi contengono ennuple (se primari) o puntatori (se secondari)

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

68

## Un B+ tree

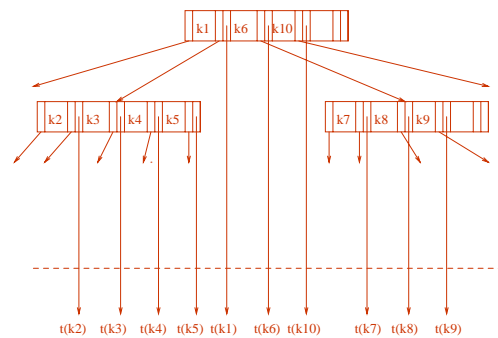


03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

69

## Un B-tree



03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

70

## Indici hash

- Strutture secondarie costituite da un file hash con record contenenti
  - pseudochiavi
  - puntatori ai record
- Costo della ricerca:
  - poco più di due accessi: uno (di solito, salvo overflow) all'indice e l'altro al file

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

71

## Esecuzione e ottimizzazione delle interrogazioni

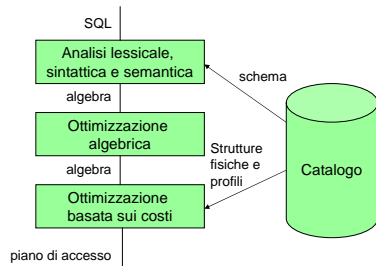
- **Query processor** (o **Ottimizzatore**): un modulo del DBMS
- Più importante nei sistemi attuali che in quelli "vecchi" (gerarchici e reticolari):
  - le interrogazioni sono espresse ad alto livello (ricordare il concetto di **indipendenza dei dati**):
    - insiemi di ennuple
    - poca proceduralità
  - l'ottimizzatore sceglie la strategia realizzativa (di solito fra diverse alternative), a partire dall'istruzione SQL

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

72

## Il processo di esecuzione delle interrogazioni



03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

73

## "Profili" delle relazioni

- Informazioni quantitative:
  - cardinalità di ciascuna relazione
  - dimensioni delle tuple
  - dimensioni dei valori
  - numero di valori distinti degli attributi
  - valore minimo e massimo di ciascun attributo
- Sono memorizzate nel "catalogo" e aggiornate con comandi del tipo `update statistics`
- Utilizzate nella fase finale dell'ottimizzazione, per stimare le dimensioni dei risultati intermedi

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

74

## Approaches to query compilation

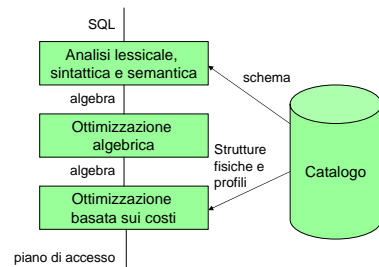
- *Compile and store*: the query is compiled once and carried out many times
  - The internal code is stored in the database, together with an indication of the dependencies of the code on the particular versions of tables and indexes of the database
  - On changes, the compilation of the query is invalidated and repeated
- *Compile and go*: immediate execution, no storage

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

75

## Il processo di esecuzione delle interrogazioni



03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

76

## Da SQL all'algebra

- (Semplificando)
  - prodotto cartesiano (`FROM`)
  - selezione (`WHERE`)
  - proiezione (`SELECT`)

```
SELECT  A , E
FROM    R1, R2, R3
WHERE   C=D AND B>100 AND F=G AND H=7 AND I>2
```

```
PROJAE (SELC=D AND B>100 AND F=G AND H=7 AND I>2 (
    (R1 JOIN R2) JOIN R3))
```

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

77

## Rappresentazione ad albero

- Alberi:
  - foglie: dati (relazioni, file)
  - nodi intermedi: operatori (operatori algebrici, poi effettivi operatori di accesso)

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

78

## Alberi per la rappresentazione di interrogazioni

- $SEL_{A=10}(R_1 JOIN R_2)$
- $R_1 JOIN SEL_{A=10}(R_2)$



03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

79

## Ottimizzazione algebrica

- Il termine *ottimizzazione* è improprio (anche se efficace) perché il processo utilizza euristiche
- Si basa sulla nozione di equivalenza:
  - Due espressioni sono *equivalenti* se producono lo stesso risultato qualunque sia l'istanza attuale della base di dati
- I DBMS cercano di eseguire espressioni equivalenti a quelle date, ma meno "costose"
- Euristiche fondamentali:
  - selezioni e proiezioni il più presto possibile (per ridurre le dimensioni dei risultati intermedi):
    - "push selections down"
    - "push projections down"

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

80

## "Push selections"

- Assumiamo A attributo di  $R_2$   
 $SEL_{A=10}(R_1 JOIN R_2) = R_1 JOIN SEL_{A=10}(R_2)$
- Riduce in modo significativo la dimensione del risultato intermedio (e quindi il costo dell'operazione)

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

81

## Una procedura euristica di ottimizzazione

- Decomporre le selezioni congiuntive in successive selezioni atomiche
- Anticipare il più possibile le selezioni
- In una sequenza di selezioni, anticipare le più selettive
- Combinare prodotti cartesiani e selezioni per formare join (eventualmente riordinando gli argomenti)
- Anticipare il più possibile le proiezioni (anche introducendone di nuove)

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

82

## Esempio

R1(ABC), R2(DEF), R3(GHI)

```
SELECT A, E
FROM R1, R2, R3
WHERE C=D AND B>100 AND F=G AND H=7 AND I>2
```

$$PROJ_{AE}(SEL_{C=D \wedge B>100 \wedge F=G \wedge H=7 \wedge I>2}(R1 JOIN R2 JOIN R3))$$

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

83

## Esempio, continua

$$PROJ_{AE}(SEL_{C=D \wedge B>100 \wedge F=G \wedge H=7 \wedge I>2}(R1 JOIN R2 JOIN R3))$$

- diventa qualcosa del tipo

$$PROJ_{AE}(SEL_{B>100}(R1) JOIN_{C=D} R2) JOIN_{F=G} SEL_{I>2}(SEL_{H=7}(R3)))$$

- oppure

$$PROJ_{AE}((PROJ_{AC}(SEL_{B>100}(R1))) JOIN_{C=D} R2) JOIN_{F=G} PROJ_G(SEL_{I>2}(SEL_{H=7}(R3))))$$

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

84

## Esecuzione delle operazioni

- I DBMS implementano gli operatori dell'algebra relazionale (o meglio, loro combinazioni) per mezzo di operazioni di livello abbastanza basso, che però possono implementare vari operatori "in un colpo solo"
- Operatori fondamentali:
  - scansione
  - accesso diretto
- A livello più alto:
  - ordinamento
- Ancora più alto
  - join

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

85

## Scan operation

- Performs a sequential access to all the tuples of a table, at the same time executing various operations of an algebraic or extra-algebraic nature:
  - Projection of a set of attributes (no duplicate elimination)
  - Selection on a local predicate (of type:  $A_i = v \dots$ )
  - Insertions, deletions, and modifications of the tuples currently accessed during the scan
- Primitives:  
open, next, read, modify, insert, delete, close

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

86

## Ordinamento

- Importante, per
  - Eliminazione duplicati
  - Produrre risultati ordinati
  - Preparare i join
- Utilizza significativamente i buffer

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

87

## Accesso diretto

- Può essere eseguito solo se le strutture fisiche lo permettono
  - indici
  - strutture hash

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

88

## Accesso diretto basato su indice

- Efficace per interrogazioni (sulla "chiave" dell'indice)
  - "puntuali" ( $A_i = v$ )
  - su intervallo ( $v_1 \leq A_i \leq v_2$ )
- Per predicati congiuntivi
  - si sceglie il più selettivo per l'accesso diretto e si verifica poi sugli altri dopo la lettura (e quindi in memoria centrale)
  - Oppure intersezioni sui riferimenti
- Per predicati disgiuntivi:
  - servono indici su tutti, ma conviene usarli se molto selettivi e facendo attenzione ai duplicati

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

89

## Accesso diretto basato su hash

- Efficace per interrogazioni (sulla "chiave" dell'indice)
  - "puntuali" ( $A_i = v$ )
  - NON su intervallo ( $v_1 \leq A_i \leq v_2$ )
- Per predicati congiuntivi e disgiuntivi, vale lo stesso discorso fatto per gli indici

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

90

## Indici e hash su più campi

- Indice su cognome e nome
  - funziona per accesso diretto su cognome?
  - funziona per accesso diretto su nome?
- Hash su cognome e nome
  - funziona per accesso diretto su cognome?
  - funziona per accesso diretto su nome?

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

91

## Join

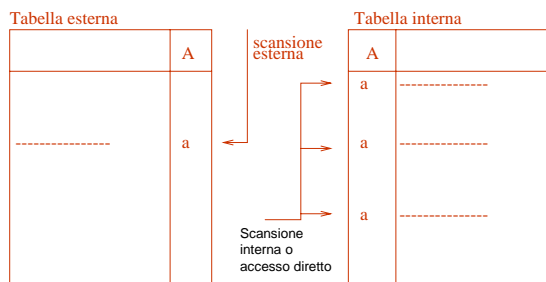
- L'operazione più costosa
- Vari metodi; i più noti:
  - *nested-loop*, *merge-scan* and *hash-based*

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

92

## Nested-loop

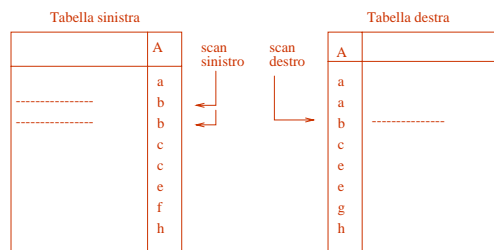


03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

93

## Merge-scan

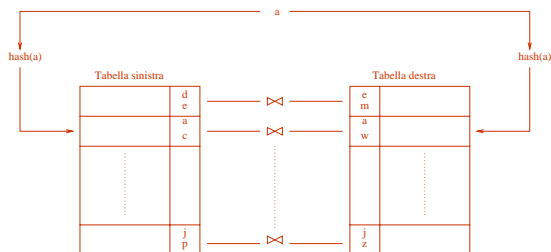


03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

94

## Hash join



03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

95

## Ottimizzazione basata sui costi

- Un problema articolato, con scelte relative a:
  - operazioni da eseguire (es.: scansione o accesso diretto?)
  - ordine delle operazioni (es. join di tre relazioni; ordine?)
  - i dettagli del metodo (es.: quale metodo di join)
- Architetture parallele e distribuite aprono ulteriori gradi di libertà

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

96



## Il processo di ottimizzazione

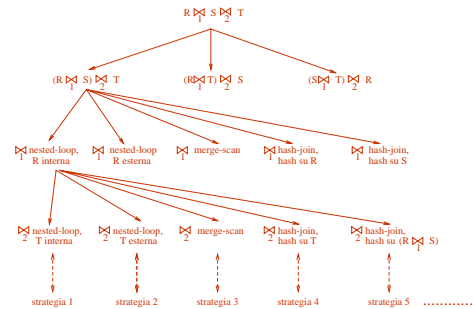
- Si costruisce un albero di decisione con le varie alternative ("piani di esecuzione")
- Si valuta il costo di ciascun piano
- Si sceglie il piano di costo minore
  
- L'ottimizzatore trova di solito una "buona" soluzione, non necessariamente quella "ottima"

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

97

## Un albero di decisione



03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

98

## Gestione dei buffer ("buffer management")

- **Buffer:**
  - area di memoria centrale, gestita dal DBMS (preallocata) e condivisa fra le transazioni
  - organizzato in **pagine** di dimensioni pari o multiple di quelle dei blocchi di memoria secondaria (1KB-100KB)
  - è importantissimo per via della grande differenza di tempo di accesso fra memoria centrale e memoria secondaria

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

99

## Scopo della gestione del buffer

- Ridurre il numero di accessi alla memoria secondaria
  - In caso di lettura, se la pagina è già presente nel buffer, non è necessario accedere alla memoria secondaria
  - In caso di scrittura, il gestore del buffer può decidere di differire la scrittura fisica (ammesso che ciò sia compatibile con la gestione dell'affidabilità – vedremo più avanti)
- La gestione dei buffer e la differenza di costi fra memoria principale e secondaria possono suggerire algoritmi innovativi.

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

100

## Buffer: DBMS e sistema operativo

- Due possibilità:
  - Il buffer controlla direttamente ("fisicamente") la memoria centrale
  - Il buffer lavora sulla memoria virtuale, lasciando poi al sistema operativo la decisione su quali pagine sono in memoria centrale e quali appoggiate su disco

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

101

## Buffer management in DBMSs vs. OSs

- Buffer management for a DBMS curiously "tastes" like the virtual memory concept of modern operating systems.
- Both techniques provide access to more data than will fit into primary memory.
- Why, then, don't we use OS virtual memory facilities to implement DBMSs?

(From material published by T Grust)

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

102

## Buffer management in DBMSs vs. OSs

- A DBMS can predict certain reference patterns for pages in a buffer a lot better than a general purpose OS.
- This is mainly because page references in a DBMS are initiated by higher-level operations (sequential scans, relational operators) the DBMS itself knows about.
- Reference pattern examples in a DBMS
  - Sequential scans call for prefetching.
    - Nested-loop joins call for page fixing and hating.
- Finally, concurrency control is based on protocols which prescribe the order in which pages have to be written back to disk. Operating systems usually do not provide hooks for that.

(From material published by T. Grust)

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

103

## Dati gestiti dal buffer manager

- Il buffer
- Un direttorio che per ogni pagina mantiene (ad esempio)
  - il file fisico e il numero del blocco
  - due variabili di stato:
    - un contatore che indica quanti programmi utilizzano la pagina
    - un bit che indica se la pagina è "sporca", cioè se è stata modificata

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

104

## Funzioni del buffer manager

- Intuitivamente:
  - riceve richieste di lettura e scrittura (di pagine)
  - le esegue accedendo alla memoria secondaria solo quando indispensabile e utilizzando invece il buffer quando possibile
  - esegue le primitive
    - *fix*, *unfix*, *setDirty*, *force*.
- Le politiche sono simili a quelle relative alla gestione della memoria da parte dei sistemi operativi; principi
  - "località dei dati": è alta la probabilità di dover riutilizzare i dati attualmente in uso
  - "legge 80-20" l'80% delle operazioni utilizza sempre lo stesso 20% dei dati

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

105

## Interfaccia offerta dal buffer manager

- *fix*: richiesta di una pagina; richiede una lettura solo se la pagina non è nel buffer (incrementa il contatore associato alla pagina)
- *setDirty*: comunica al buffer manager che la pagina è stata modificata e non ancora salvata in memoria secondaria
- *unfix*: indica che la transazione ha concluso l'utilizzo della pagina (decrementa il contatore associato alla pagina)
- *force*: trasferisce in modo sincrono una pagina in memoria secondaria (su richiesta del gestore dell'affidabilità, non del gestore degli accessi)

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

106

## Esecuzione della fix

- Cerca la pagina nel buffer;
  - se c'è, restituisce l'indirizzo
  - altrimenti, cerca una pagina libera nel buffer (contatore a zero);
    - se la trova, legge il blocco di interesse dalla memoria secondaria e restituisce l'indirizzo della pagina
    - altrimenti, due alternative
      - "*steal*": seleziona una "vittima", pagina occupata del buffer; scrive i dati della vittima in memoria secondaria (se "dirty"); legge il blocco di interesse dalla memoria secondaria e restituisce l'indirizzo
      - "*no-steal*": pone l'operazione in attesa

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

107

## Commenti

- Il buffer manager richiede scritture in due contesti diversi:
  - in modo **sincrono** quando è richiesto esplicitamente con una *force*
  - in modo **asincrono** quando lo ritiene opportuno (o necessario); in particolare, può decidere di anticipare o posticipare scritture per coordinarle e/o sfruttare la disponibilità dei dispositivi

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

108

### Algoritmi sui buffer, esempio 1

- Join, con il metodo nested loop:
  - R1 100 blocchi
  - R2 50 blocchi
  - 6 pagine a disposizione nel buffer

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

109

### Algoritmi sui buffer, esempio 2

- Ordinamento:
  - File di 1.000.000.000 di record di 100 byte ciascuno (100GB)
  - Blocchi di 10KB
  - Buffer disponibile di 100MBCome possiamo procedere?
  - Merge-sort ...

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

110

### Progettazione fisica

- La fase finale del processo di progettazione di basi di dati
- input
  - lo schema logico e informazioni sul carico applicativo
- output
  - schema fisico, costituito dalle definizioni delle relazioni con le relative strutture fisiche (e molti parametri, spesso legati allo specifico DBMS)

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

111

### Strutture fisiche nei DBMS relazionali

- Struttura primaria:
  - disordinata (heap, "unclustered")
  - ordinata ("clustered"), anche su una pseudochiave
  - hash ("clustered"), anche su una pseudochiave, senza ordinamento
  - clustering di più relazioni
- Indici (densi/sparsi, semplici/composti):
  - ISAM (statico), di solito su struttura ordinata
  - B-tree (dinamico)
  - Indici hash (secondario, poco dinamico)

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

112

### Strutture fisiche in alcuni DBMS

- Oracle:
  - struttura primaria
    - file heap
    - "hash cluster" (cioè struttura hash)
    - cluster (anche plurirelazionali) anche ordinati (con B-tree denso)
  - indici secondari di vario tipo (B-tree, bit-map, funzioni)
- DB2:
  - primaria: heap o ordinata con B-tree denso
  - indice sulla chiave primaria (automaticamente)
  - indici secondari B-tree densi
- SQL Server:
  - primaria: heap o ordinata con indice B-tree sparso
  - indici secondari B-tree densi

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

113

### Strutture fisiche in alcuni DBMS, 2

- Ingres (anni fa):
  - file heap, hash, ISAM (ciascuno anche compresso)
  - indici secondari
- Informix (per DOS, 1994):
  - file heap
  - indici secondari (e primari [cluster] ma non mantenuti)

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

114

## Definizione degli indici SQL

- Non è standard, ma presente in forma simile nei vari DBMS
  - `create [unique] index IndexName on TableName(AttributeList)`
  - `drop index IndexName`

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

115

## Progettazione fisica nel modello relazionale

- La caratteristica comune dei DBMS relazionali è la disponibilità degli indici:
  - la progettazione fisica spesso coincide con la scelta degli indici (oltre ai parametri strettamente dipendenti dal DBMS)
- Le chiavi (primarie) delle relazioni sono di solito coinvolte in selezioni e join: molti sistemi prevedono (oppure suggeriscono) di definire indici sulle chiavi primarie
- Altri indici vengono definiti con riferimento ad altre selezioni o join "importanti"
- Se le prestazioni sono insoddisfacenti, si "tara" il sistema aggiungendo o eliminando indici
- È utile verificare se e come gli indici sono utilizzati con il comando SQL `show plan` oppure `explain`

03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

116

## Progettazione fisica: euristiche suggerite da Informix

- Non creare indici su relazioni piccole (<200 ennuple)
- non creare indici su campi con pochi valori (se proprio servono, che siano primari)
- creare indici su campi con selezioni
- per i join: creare indici sulla relazione più grande

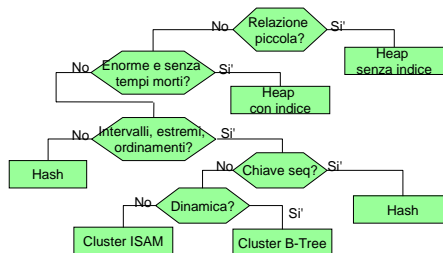
03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

117

## Sceita della struttura secondo Shasha

- D. Shasha. Database Tuning: a principled approach. Prentice-Hall, 1992



03/05/2006

Basi di dati, vol.2 cap.1 Organizzazione fisica

118