Corso di Laurea in Ingegneria Informatica
Fondamenti di Informatica II
Modulo "*Basi di dati*"
a.a. 2015-2016

Docente: Gigliola Vaglini
Docente laboratorio: Francesco Pistolesi

1

Lezione 11

Organizzazione fisica e gestione delle interrogazioni

2

Tecnologia delle BD

- Il DBMS è una "scatola nera"
- Perché aprirla?
  - capire come funziona può essere utile per un migliore utilizzo

3

DataBase Management System — DBMS

Sistema per gestire collezioni di dati:

- **grandi**
- **persistenti**
- **condivise**, garantendo **affidabilità** e privatezza .

- In più 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).

4

Le basi di dati sono grandi e persistenti

- La persistenza richiede la gestione della memoria secondaria
- La grandezza richiede che tale gestione sia sofisticata
- Gli utenti vedono il modello logico, ma le strutture logiche debbono essere gestite efficientemente in memoria secondaria:
  - servono strutture fisiche opportune

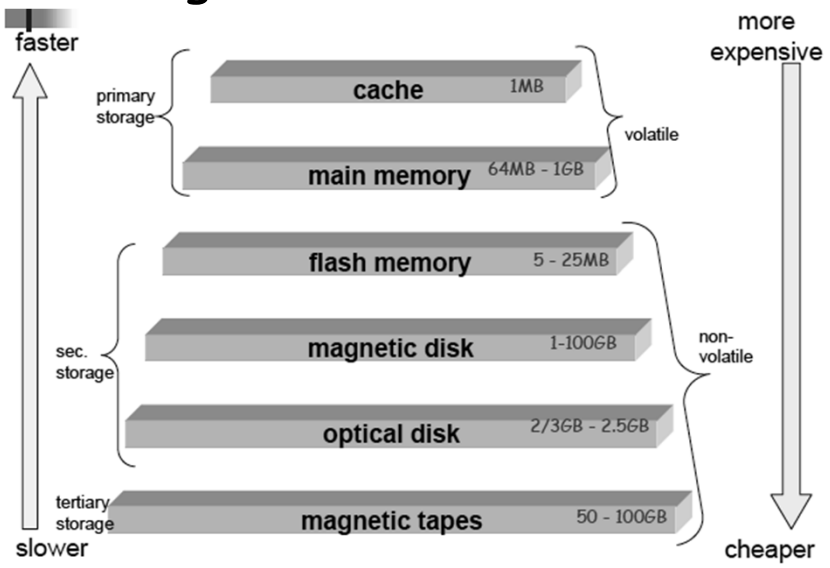
5

Le basi di dati vengono interrogate …

- I programmi possono fare riferimento solo a dati in memoria principale, quindi i dati in memoria secondaria possono essere utilizzati solo se prima trasferiti in memoria principale (questo spiega i termini "principale" e "secondaria) serve un'interazione fra memoria principale e secondaria che limiti il più possibile gli accessi alla secondaria

6

La gerarchia di memoria



7

Prestazioni di una memoria

- Dato un indirizzo di accesso, le prestazioni di memoria si misurano in termini della somma tra la latenza (tempo per accedere al primo byte) e il tempo di trasferimento (tempo necessario a muovere tutti i dati)

8

Memoria principale e secondaria

- Accesso a memoria secondaria:
  - tempo di posizionamento della testina (10-50ms)
  - tempo di latenza (5-10ms)
  - tempo di trasferimento (1-2ms)

  in media non meno di 10 ms
- Il tempo 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

9

Memoria principale e secondaria

- I dispositivi di memoria secondaria sono organizzati in **blocchi** di lunghezza (di solito) **fissa** (ordine di grandezza: alcuni KB)
- Le uniche operazioni sono la lettura e la scrittura dei dati di un blocco
- Accessi a blocchi "vicini" costano meno (contiguità)
- La memoria principale è organizzata in pagine

10

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)

11

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)

12

Le basi di dati sono affidabili e condivise

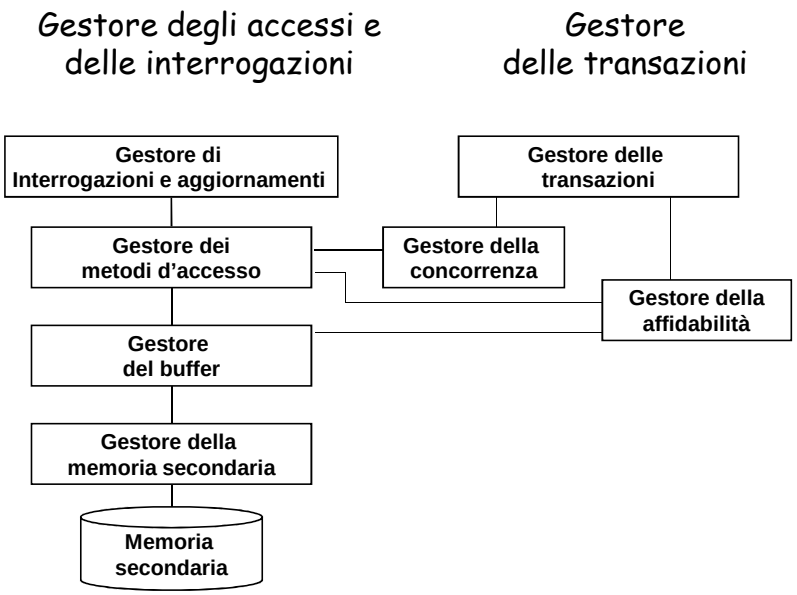
- Le basi di dati debbono essere preservate anche in presenza di malfunzionamenti
- L'affidabilità è impegnativa per via degli aggiornamenti frequenti e della necessità di gestire il buffer
- Una base di dati è una risorsa **condivisa** fra le varie applicazioni
  - Attività diverse su dati in parte condivisi:
    - meccanismi di autorizzazione
  - Attività multi-utente su dati condivisi:
    - controllo della **concorrenza**

13

Aggiornamenti su basi di dati condivise …

- Intutitivamente, 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

14



15

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

16

DBMS e file system

- Il file system è il componente del sistema operativo che gestisce la memoria secondaria
- I DBMS ne utilizzano le funzionalità per creare ed eliminare file e per leggere e scrivere singoli blocchi o sequenze di blocchi contigui.
- 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.

17

Tuple e blocchi

- Le tuple di una relazione stanno in blocchi contigui
- A volte in un blocco ci sono tuple di relazioni diverse ma correlate (i join sono favoriti)

18

Blocchi e record

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

19

Organizzazione delle tuple in pagine

- Assumiamo la pagina coincidente con il blocco (buffer)
- Ci sono varie alternative, anche legate ai metodi di accesso; sono sistemate sequenzialmente nei file
- Inoltre:
  - se la lunghezza delle tuple è fissa, la struttura può essere semplificata
  - alcuni sistemi possono spezzare le tuple su più pagine(necessario per tuple grandi)

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")

21

Esercizio

- Calcolare il fattore di blocco e il numero di blocchi occupati da una relazione contenente T = 500000 tuple di lunghezza fissa pari a L = 100 byte in un sistema con blocchi di dimensione pari a B = 1 kilobyte.
-

22

Soluzione

- $N_B = D_T/B$ $D_T = T*L$
- $N_B = 50000000/1024$
- $F_B = B/L$
  $F_B = 1024/100$

23

Dati gestiti dal buffer manager

- Il buffer
- Una directory 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

24

Il funzionamento del buffer manager

- Le politiche sono simili a quelle relative alla gestione della memoria da parte dei sistemi operativi;
  - "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

25

Funzioni del buffer manager

- riceve richieste di lettura e scrittura (di pagine)
- le esegue accedendo alla memoria secondaria solo quando indispensabile e utilizzando invece il buffer quando possibile

26

Interfaccia offerta dal buffer manager

- esegue le primitive
  - ***fix***: richiesta di una pagina; richiede una lettura solo se la pagina non è nel buffer (incrementa il contatore associato alla pagina)
  - ***setDirty***: comunica che la pagina è stata modificata
  - ***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)

27

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, vi inserisce i dati letti dalla memoria secondaria e ne restituisce l'indirizzo
    - altrimenti, due alternative
      - "**steal**": selezione di una "vittima", pagina occupata del buffer; i dati della vittima sono scritti in memoria secondaria; vengono letti i dati di interesse dalla memoria secondaria e si restituisce l'indirizzo
      - "**no-steal**": l'operazione viene posta in attesa

28

Commenti

- Il buffer manager fa partire 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

29

Strutture primarie e secondarie

- Le tuple devono essere organizzate all'interno dei blocchi dei file
- Strutture primarie sono quelle che contengono propriamente i dati
- Strutture secondarie sono quelle che favoriscono l'accesso ai dati senza contenerli

30

Strutture primarie

- L'organizzazione può essere
  - **seriale**: ordinamento fisico ma non logico
  - **ordinata**: l'ordinamento delle tuple coerente con quello di un campo
  - **array**: posizioni individuate attraverso indici
  - ad albero

31

Organizzazione seriale

- Chiamata anche:
  - "Entry sequenced"
  - file heap
  - file disordinato
- Gli inserimenti vengono effettuati
  - in coda (con riorganizzazioni periodiche)
  - al posto di record cancellati

32

Strutture ordinate e File hash

- Le strutture ordinate permettono ricerche binarie
- I file hash permettono un accesso diretto molto efficiente
  - La tecnica si basa su quella utilizzata per le tavole hash in memoria centrale

33

Remind: 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 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)

34

Tavola hash

- senza sprecare spazio
  - **funzione hash:**
    - associa ad ogni valore della chiave un "indirizzo", in uno spazio di dimensione leggermente superiore rispetto a quello strettamente necessario
    - poiché il numero di possibili chiavi è molto maggiore del numero di possibili 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 causale e uniforme, riducendo le probabilità di collisione (che si riduce aumentando lo spazio ridondante)

35

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
60600	0
66301	1
205751	1
205802	2
200902	2
116202	2
200604	4
66005	5
116455	5
200205	5
201159	9
205610	10
201260	10
102360	10
205460	10
205912	12
205762	12
200464	14
205617	17
205667	17

M	M mod 50
200268	18
205619	19
210522	22
205724	24
205977	27
205478	28
200430	30
210533	33
205887	37
200138	38
102338	38
102690	40
115541	41
206092	42
205693	43
205845	45
200296	46
205796	46
200498	48
206049	49

36

Risoluzione delle 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

37

File hash

- L'idea è la stessa, ma si basa sull'organizzazione in blocchi
- In questo modo si "ammortizzano" le probabilità di collisione

38

Un esempio

- 40 record
- tavola hash con 50 posizioni:
  - 1 collisione a 4
  - 2 collisioni a 3
  - 5 collisioni a 2
- file hash con fattore di blocco 10;
  5 blocchi con 10 posizioni ciascuno:
  - due soli overflow!

39

Un file hash

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		

40

Strutture ad albero

- Dette anche indici
- Strutture basate sull'uso di puntatori
- Si utilizzano sia come strutture primarie che secondarie

41

Strutture ad albero

- Indice:
  - struttura per l'accesso (efficiente) ai record sulla base dei valori di un campo (o di una "concatenazione di campi") detto chiave (o, meglio, pseudochiave, perché non è necessariamente identificante);
- 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
  - Ad es., l'indice analitico di un libro: lista di coppie (termine,pagina), ordinata alfabeticamente sui termini, posta in fondo al libro e separabile da esso

42

Tipi di indice

- indice primario:
  - su un campo sul cui ordinamento è basata la memorizzazione
- indice secondario
  - su un campo con ordinamento diverso da quello dell'ordinamento di memorizzazione
- indice denso
  - contiene un record per ciascun record del file
- indice sparso
  - contiene un numero di record inferiore rispetto a quelli del file

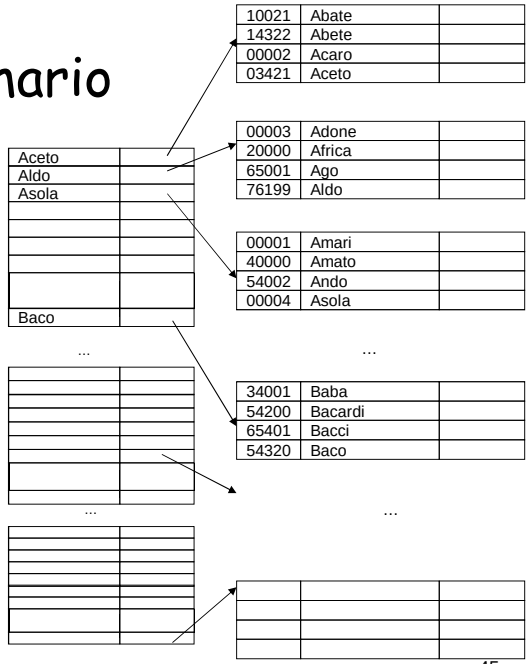
43

Tipi di indice, commenti

- Un indice primario può essere sparso (non tutti i valori della chiave compaiono nell'indice)
  - Esempio, sempre rispetto ad un libro
    - indice generale
- Gli indici secondari sono densi perché tutti i valori della chiave devono essere raggiungibili
- 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

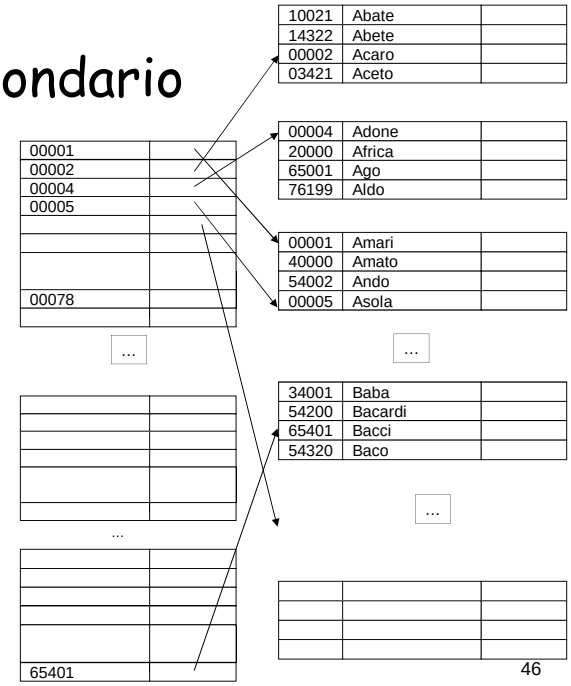
44

Indice primario



45

Indice secondario



46

Caratteristiche degli indici

- Accesso diretto (sulla chiave) efficiente
- 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

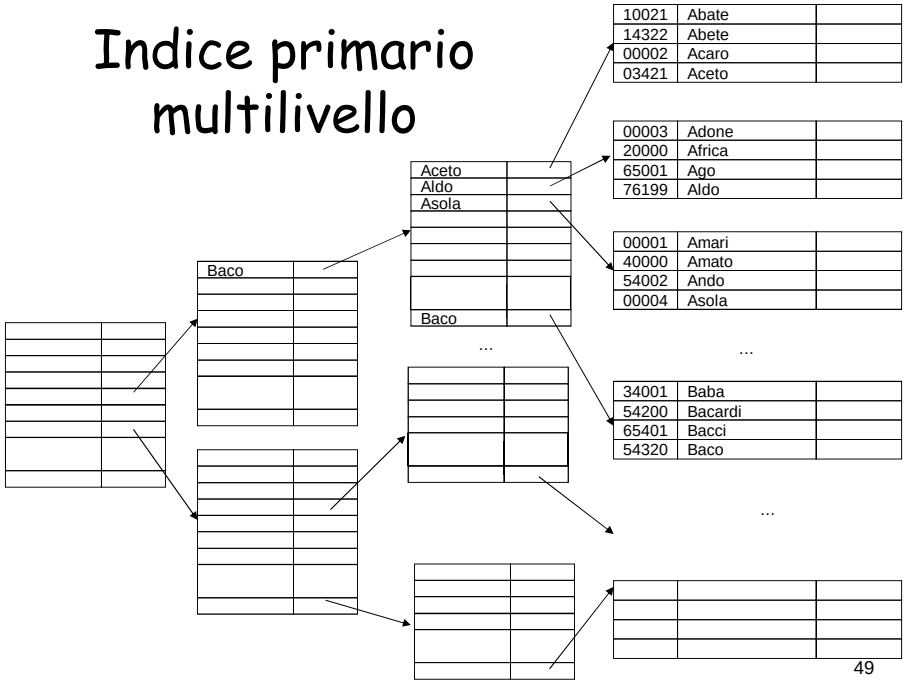
47

Indici multilivello

- Gli indici sono file essi stessi e quindi ha senso costruire indici sugli indici, per evitare di fare ricerche fra blocchi diversi
- Possono esistere più livelli fino ad avere il livello più alto con un solo blocco; 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

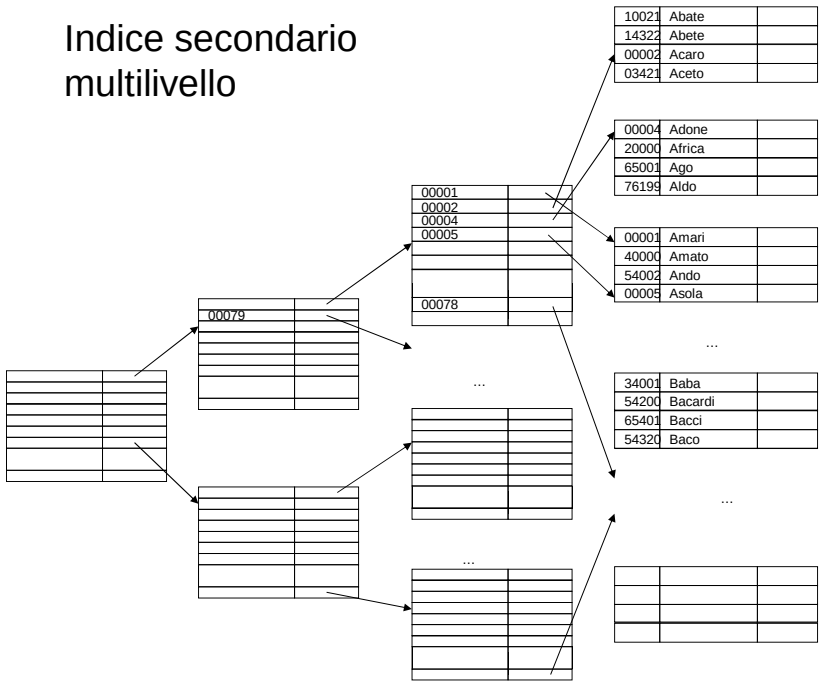
48

Indice primario multilivello



49

Indice secondario multilivello



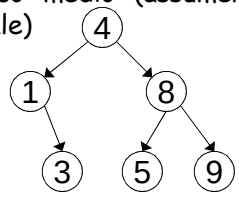
Indici, problemi

- Le strutture di indice basate su strutture ordinate sono poco flessibili in presenza di elevata dinamicità
- Gli indici utilizzati dai DBMS sono in generale
  - indici dinamici multilivello
  - Vengono memorizzati e gestiti come B-tree (intuitivamente: alberi di ricerca bilanciati)
    - Alberi binari di ricerca
    - Alberi n-ari di ricerca
    - Alberi n-ari di ricerca bilanciati

51

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)



52

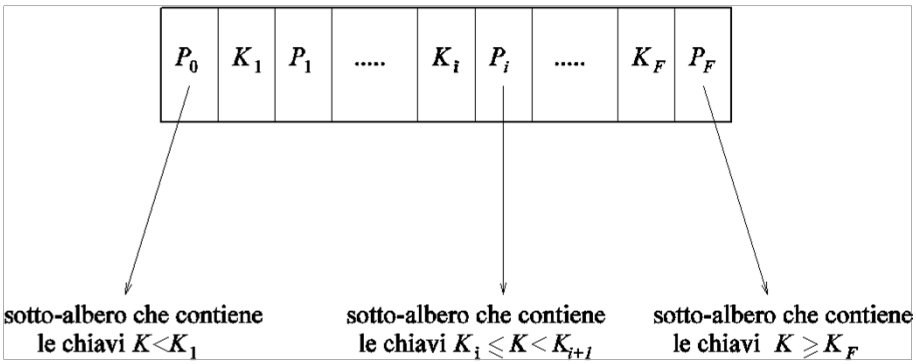
Albero di ricerca di ordine P

- 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 può corrispondere ad un blocco

- Un B-tree è un albero di ricerca che viene mantenuto bilanciato, grazie a:
  - Riempimento parziale (mediamente 70%)
  - Riorganizzazioni (locali) in caso di sbilanciamento

53

Organizzazione dei nodi del B-tree



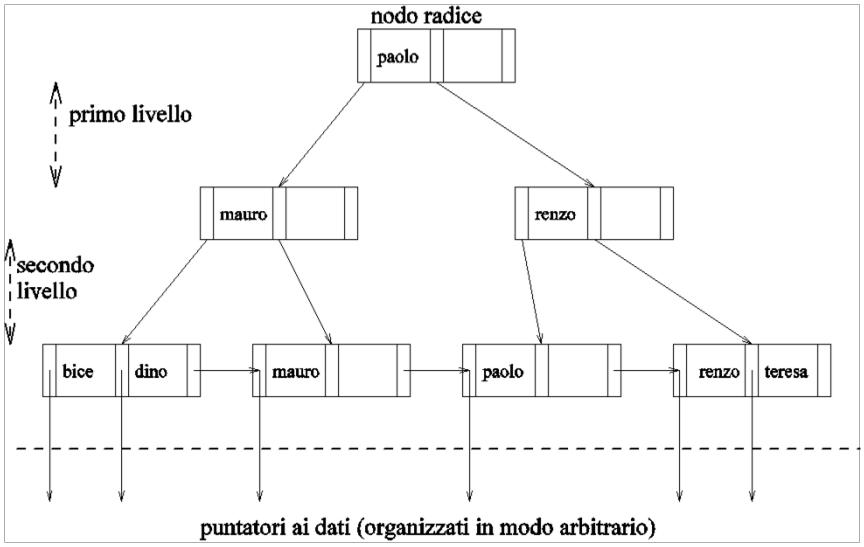
54

B tree e B+ tree

- B+ tree:
  - le foglie sono collegate in una lista
  - molto usati nei DBMS
- B tree:
  - I nodi intermedi possono avere puntatori direttamente ai dati

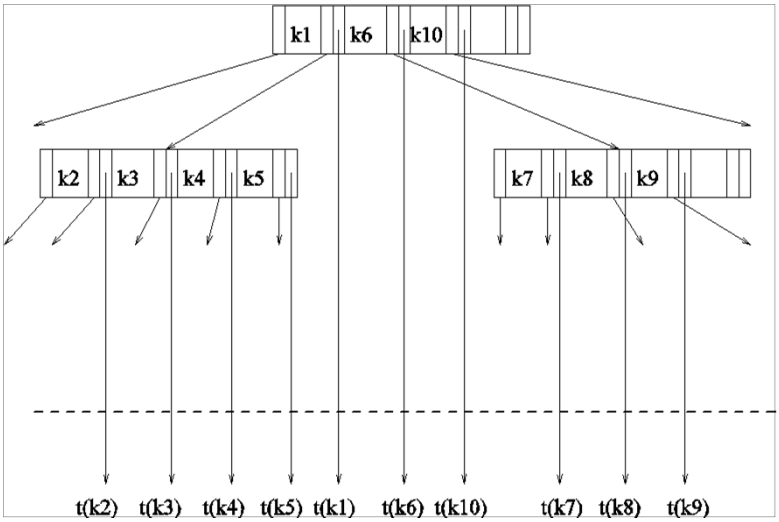
55

Un B+ tree



56

Un B-tree



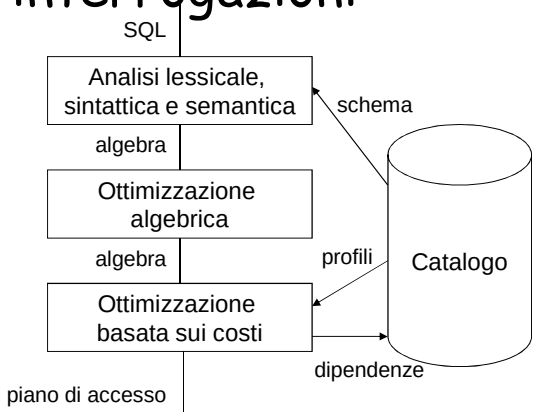
57

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 tuple
    - poca proceduralità
  - l'ottimizzatore sceglie la strategia realizzativa (di solito fra diverse alternative), a partire dall'istruzione SQL

58

Il processo di esecuzione delle interrogazioni



59

"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 possono essere aggiornate con comandi del tipo ***`update statistics`***
- Utilizzate nella fase finale dell'ottimizzazione, per stimare le dimensioni dei risultati intermedi

60

Ottimizzazione algebrica

- Il termine ottimizzazione è improprio 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
- Euristica fondamentale:
  - selezioni e proiezioni il più presto possibile (per ridurre le dimensioni dei risultati intermedi):
    - "push selections down"
    - "push projections down"

61

"Push selections"

- Assumiamo A attributo di R_2

$$\mathbf{SEL}_{A=10}(R_1 \ \mathbf{JOIN}\ R_2) = R_1 \ \mathbf{JOIN}\ \mathbf{SEL}_{A=10}(R_2)$$

- Riduce in modo significativo la dimensione del risultato intermedio (e quindi il costo dell'operazione)

62

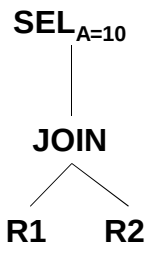
Rappresentazione interna delle interrogazioni

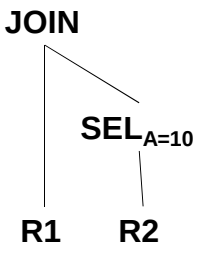
- Alberi:
  - foglie: dati (relazioni, file)
  - nodi intermedi: operatori (operatori algebrici, poi effettivi operatori di accesso)

63

Alberi per la rappresentazione di interrogazioni

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





64

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
- Anticipare il più possibile le proiezioni (anche introducendone di nuove)

65

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
```

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

PROJ $_{AE}$ **(SEL** $_{C=D \text{ AND } B>100 \text{ AND } F=G \text{ AND } H=7 \text{ AND } I>2}$ **(**
(R1 JOIN R2) JOIN R3))

66

Esempio, continua

$$\text{PROJ}_{AE}\ (\text{SEL}_{C=D\ \text{AND}\ B>100\ \text{AND}\ F=G\ \text{AND}\ H=7\ \text{AND}\ I>2}\ (\text{(R1 JOIN R2) JOIN R3))}$$

- diventa qualcosa del tipo

$$\text{PROJ}_{AE}\ (\text{SEL}_{B>100}\ (\text{R1})\ \text{JOIN}_{C=D}\ \text{R2})\ \text{JOIN}_{F=G}\ \text{SEL}_{I>2}(\text{SEL}_{H=7}(\text{R3})))$$

- oppure

$$\text{PROJ}_{AE}(\ \text{PROJ}_{AEF}((\text{PROJ}_{AC}(\text{SEL}_{B>100}\ (\text{R1})))\ \text{JOIN}_{C=D}\ \text{R2})\ \text{JOIN}_{F=G}\ \text{PROJ}_{G}\ (\text{SEL}_{I>2}(\text{SEL}_{H=7}(\text{R3}))))$$

67

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:
  - accesso diretto
  - scansione
- A livello più alto:
  - ordinamento
- Ancora più alto
  - Join, l'operazione più costosa

68

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

69

Metodi di join

- Nested loop
- Merge scan
- Hash-based

70

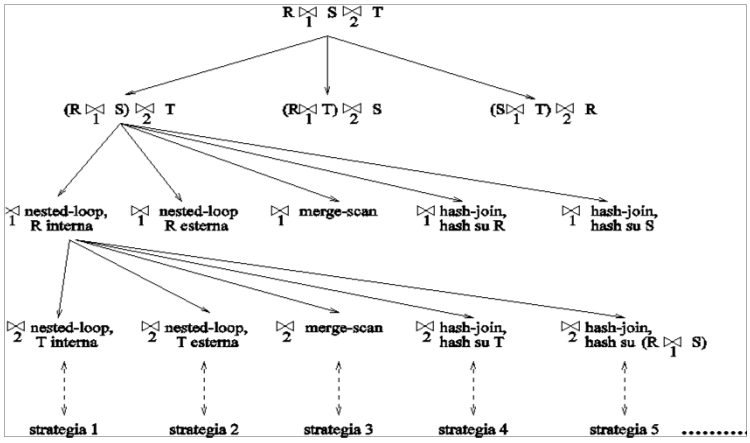
Il processo di ottimizzazione

- 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 l'"ottimo"

71

Un albero di decisione



72

Ordine del join

- Join associativo
- La scelta dell'ordine delle coppie a cui si applica prima l'operatore è rilevante
- Attributo di join chiave

73

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 definizione delle relazioni con le relative strutture fisiche (e molti parametri, spesso legati allo specifico DBMS)

74

Progettazione logica nel modello relazionale

- La caratteristica comune dei DBMS relazionali è la disponibilità degli indici:
  - la progettazione logica 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`

75

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*

76

Strutture fisiche nei DBMS relazionali

- Struttura primaria:
  - disordinata (heap, "unclustered")
  - ordinata ("clustered")
  - hash ("clustered")
- Indici (densi/sparsi, semplici/composti):
  - ISAM (statico), di solito su struttura ordinata
  - B-tree (dinamico)

77

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)

78

Strutture fisiche in alcuni DBMS

- 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

79