Organizzazione fisica e gestione delle interrogazioni

Atzeni, Ceri, Paraboschi, Torlone
"**Basi di dati**
Architetture e linee di evoluzione"
McGraw-Hill Italia
Capitolo 1

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")

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 sui dispositivi solo la lettura e la scrittura di una **pagina**, cioè dei dati di un blocco (cioè di una stringa di byte);
- per comodità consideriamo **blocco** e **pagina** sinonimi

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 costo di un accesso a memoria secondaria è quattro o più ordini di grandezza maggiore di quello per operazioni in memoria centrale

Memoria principale e secondaria

- 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)

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

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)

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

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***.

Funzioni del buffer manager

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

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
- ***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)

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

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

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.

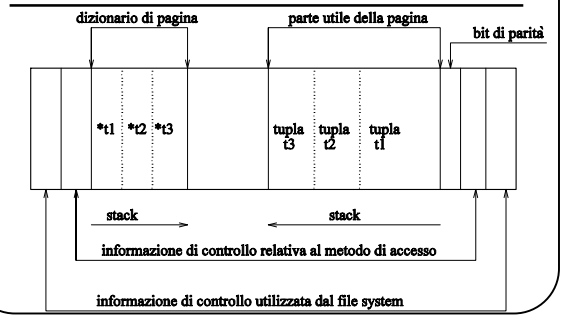
DBMS e file system

- 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'intero database)
- 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 tuple di un'unica relazione

Organizzazione delle tuple nelle pagine

- Ci sono varie alternative, anche legate ai metodi di accesso; vediamo una possibilità
- 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)

Organizzazione delle tuple nelle pagine



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
  - in coda (con riorganizzazioni periodiche)
  - al posto di record cancellati

Strutture ordinate

- Permettono ricerche binarie
- Nelle basi di dati relazionali si utilizzano quasi solo in combinazione con indici (file ISAM o file ordinati con indice primario)

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

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)

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 causale e uniforme, riducendo le probabilità di collisione (che si riduce aumentando lo spazio ridondante)

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

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

File hash

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

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
- file hash con fattore di blocco 10;
  5 blocchi con 10 posizioni ciascuno:
  - due soli overflow!

  numero medio di accessi: 1,05

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

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		

File hash, osservazioni

- 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)
- Le collisioni (overflow) sono di solito gestite con blocchi collegati
- Non è efficiente per ricerche basate su intervalli (né per ricerche basate su altri attributi)
- I file hash "degenerano" se si riduce lo spazio sovrabbondante: funzionano solo con file la cui dimensione non varia molto nel tempo

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

Tipi di indice

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

Tipi di indice

- indice denso:
  - contiene un record per ciascun record del file
- indice sparso:
  - contiene un numero di record inferiore rispetto a quelli del file

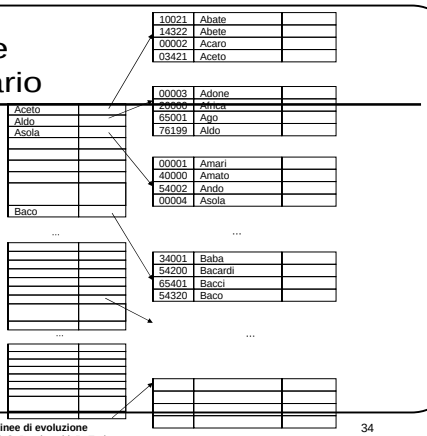
Tipi di indice, commenti

- Un indice primario può essere sparso, uno secondario deve essere denso
- Esempio, sempre rispetto ad un libro
  - indice generale
  - indice analitico

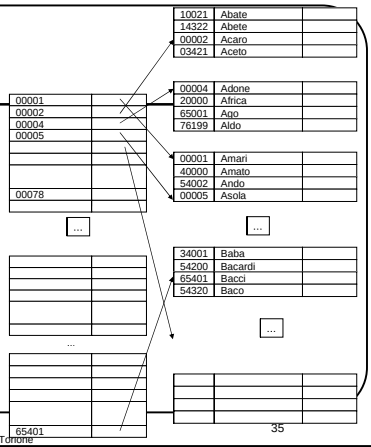
Tipi di indice, commenti

- 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

Indice Primario



Indice secondario



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

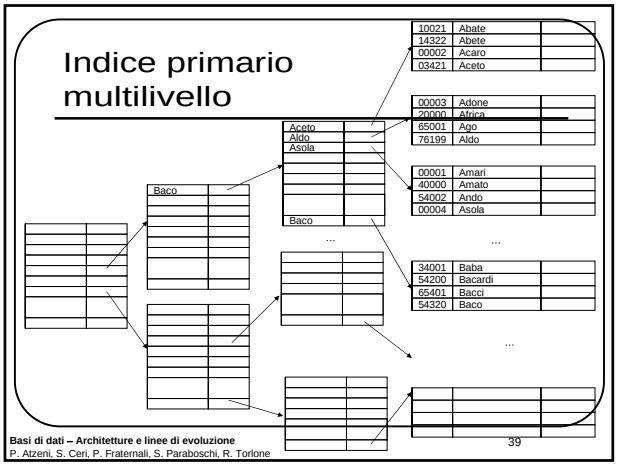
Indici secondari, due osservazioni

- 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)

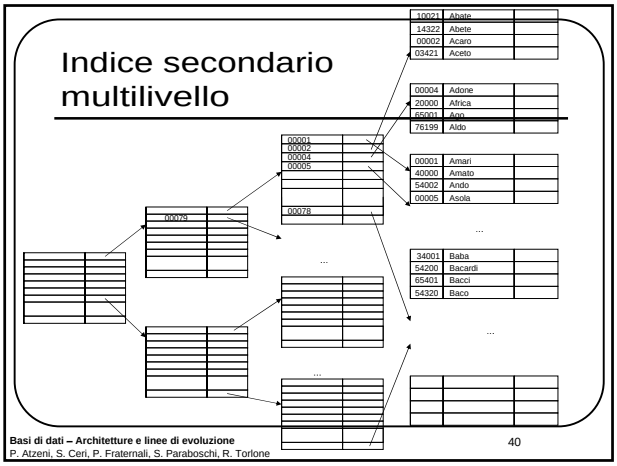
Indici multilivello

- 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

Indice primario multilivello



Indice secondario multilivello

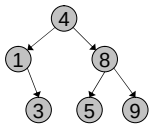


Indici, problemi

- 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

Albero binario di ricerca

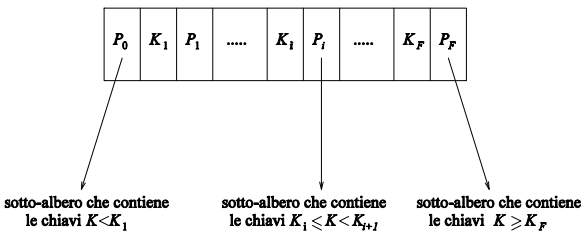
- Albero binario etichettato in cui per ogni nodo il sottoalbero sinistro contiene solo etichette minori di quella del nodo e il sottoalbero destro etichette maggiori
- tempo di ricerca (e inserimento), pari alla profondità:
  - logaritmico nel caso "medio" (assumendo un ordine di inserimento casuale)



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

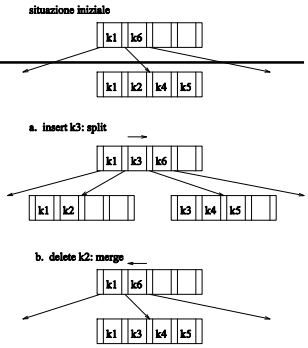
Organizzazione dei nodi del B-tree



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

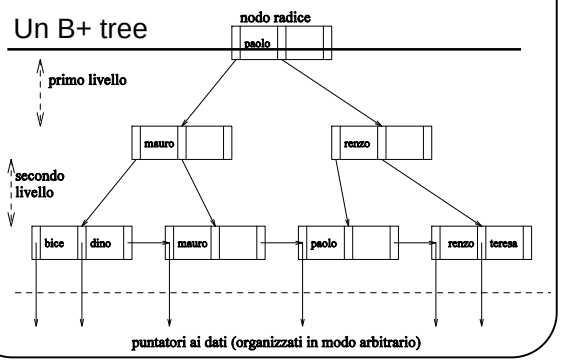
Split e merge



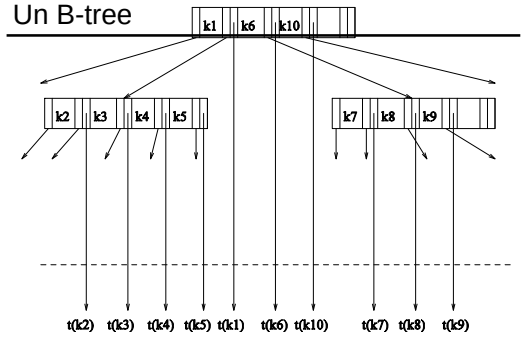
B tree e B+ tree

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

Un B+ tree



Un B-tree



Progettazione Fisica

- Riceve in ingresso il progetto logico, le caratteristiche del DBMS scelto ed il carico applicativo
- Produce
  - l'effettiva definizione delle tabelle (i create table con gli opportuni vincoli)
  - definisce le strutture fisiche di accesso

Creazione degli indici in SQL

- In molto sistemi (se non si specifica nulla) viene creato un indice primario sulla chiave primaria
- Indici secondari
  - CREATE INDEX [UNIQUE] index_name ON table_name (column_name)

Il processo di esecuzione delle interrogazioni