

La struttura interna dei DBMS



Caratteristiche dei DBMS

Devono **garantire**

- **Affidabilità**

(Non si perdono informazioni in caso di malfunzionamento)

- **Privatezza**

(Restrizione degli accessi)

- **Efficienza**

(IL sistema svolge inserimenti, modifiche, ricerche in un tempo ragionevole)

Gestiscono **collezioni di dati**:

- **Grandi**

- **Persistenti**

- **Condivise**



Accesso ai dati

Basi di dati **persistenti**

- I dati devono essere memorizzati in memoria secondaria

Basi di dati **grandi ed efficienza**

- L'accesso ai dati richiede opportune strutture in memoria secondaria (tali strutture non fanno parte del modello logico)
- Occorre un meccanismo di buffering fra memoria principale e secondaria (... la RAM non contiene le tabelle) per limitare accessi alla memoria secondaria
- Occorre tradurre efficientemente le interrogazioni dell'utente in accessi ai dati



Affidabilità

I dati devono essere sempre preservati

- Occorre un meccanismo per il **ripristino** in caso di errori nel software e malfunzionamenti hardware (Backup e log)
- Lo spostamento **da memoria principale a memoria secondaria** deve tenere in considerazione la consistenza dei dati
- Le transazioni devono possedere le **proprietà ACIDE**
 - **Atomicità**
 - **Consistenza**
 - **Isolamento**
 - **Persistenza**

Concorrenza e sicurezza

L'isolamento

- implica che le transazioni devono comportarsi come se fossero serializzate
- l'isolamento deve essere garantito anche in presenza di concorrenza

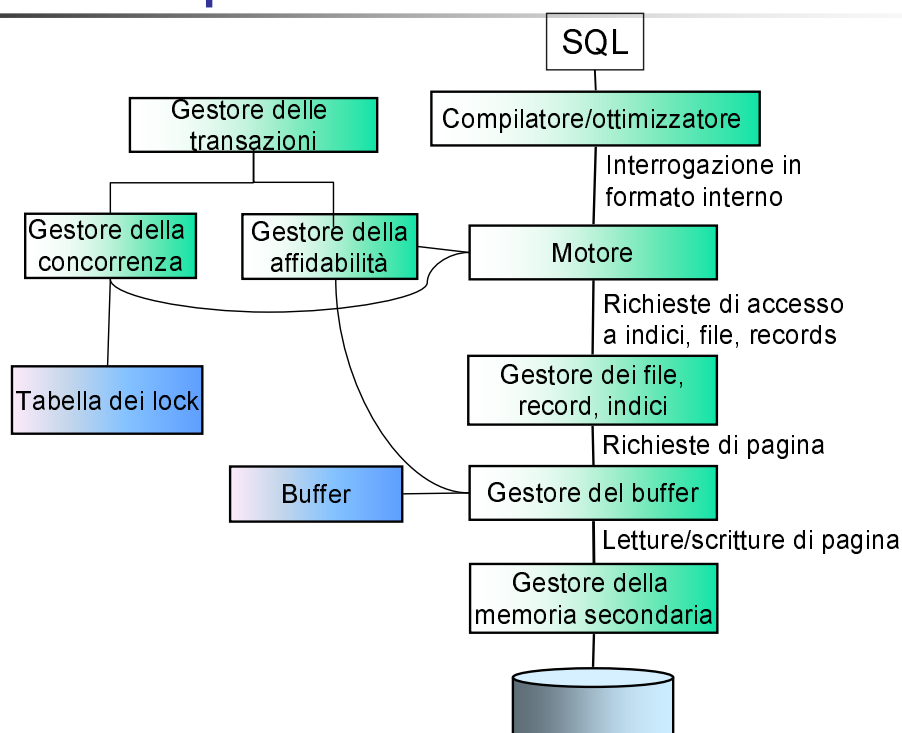
Gestione della concorrenza

- un apposito modulo del DBMS gestisce gli accessi in modo da garantire l'isolamento
- e' invisibile all'utente

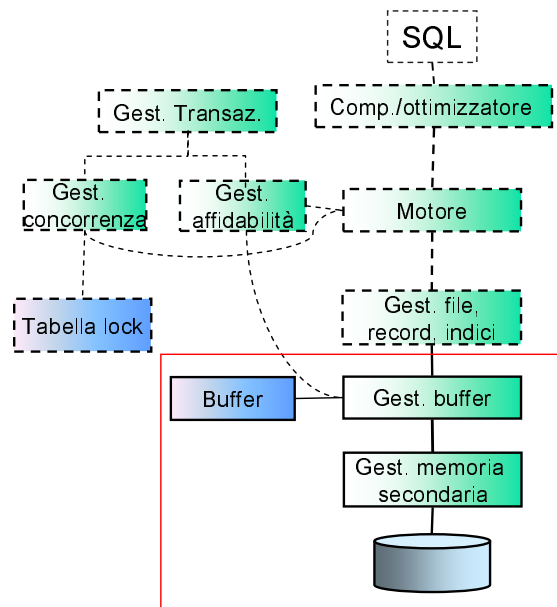
Gestione della sicurezza

- e' semplice grazie alla separazione fra modello logico e modello fisico
- il DBMS controlla ad ogni accesso che l'utente abbia i privilegi necessari

Le componenti di un DBMS



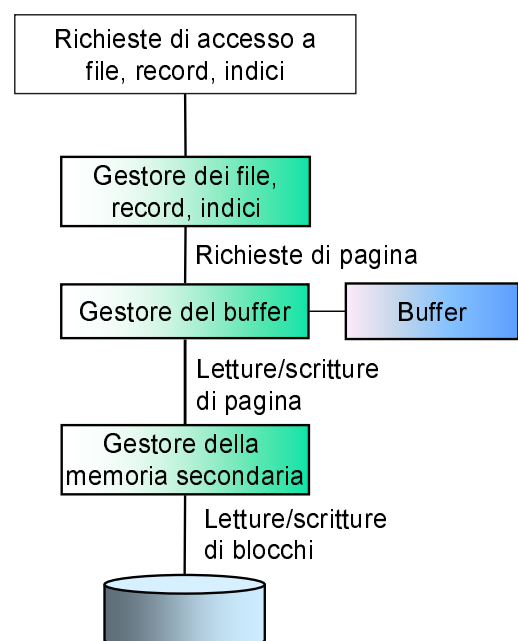
Gestione della memoria secondaria



Gestione della memoria secondaria

Le componenti coinvolte

- **Gestore dei file, record, indici**
Analizza le richieste di accesso a record, file e indici e individua le pagine da caricare
- **Gestore dei buffer**
Gestisce una cache dei record contenuti nella memoria secondaria
- **Gestore della memoria secondaria**
Scrive e legge effettivamente la memoria secondaria, individuandone la collocazione sui dischi





Memoria secondaria (dischi)

Alcuni fatti noti

- La memoria secondaria è organizzata in blocchi (pagine)
 - Un blocco è l'unità minima trasferibile
 - Un blocco va da pochi KByte a alcune decine di KByte
- L'accesso alla memoria secondaria
 - tempo di **posizionamento della testina** (5-50ms)
 - tempo di **latenza** (5-10ms)
 - tempo di **trasferimento** (0.5-2ms)
 - in media circa 10 ms



Memoria secondaria (dischi) II

L'accesso alla memoria secondaria

- È estremamente più costoso dell'accesso alla memoria primaria
- Con i DBMS, spesso il tempo speso in accesso alla memoria primaria è ininfluente:
 - la complessità delle operazioni si può misurare in termini di **numero di accessi alla memoria secondaria**
- Il tempo di accesso alla memoria secondaria dipende dall'ordine in cui i blocchi vengono letti/scritti
 - La lettura di blocchi contigui può costare **10/100 volte meno**

Un esempio: l'ordinamento

Cosa succede quando la memoria primaria non contiene i dati?

Per l'ordinamento si utilizza una nuova versione del merge-sort

- È come quello classico ma si suddivide i dati solo fino a quando una parte dei dati sta in memoria primaria
- si cerca di minimizzare gli accessi alla RAM

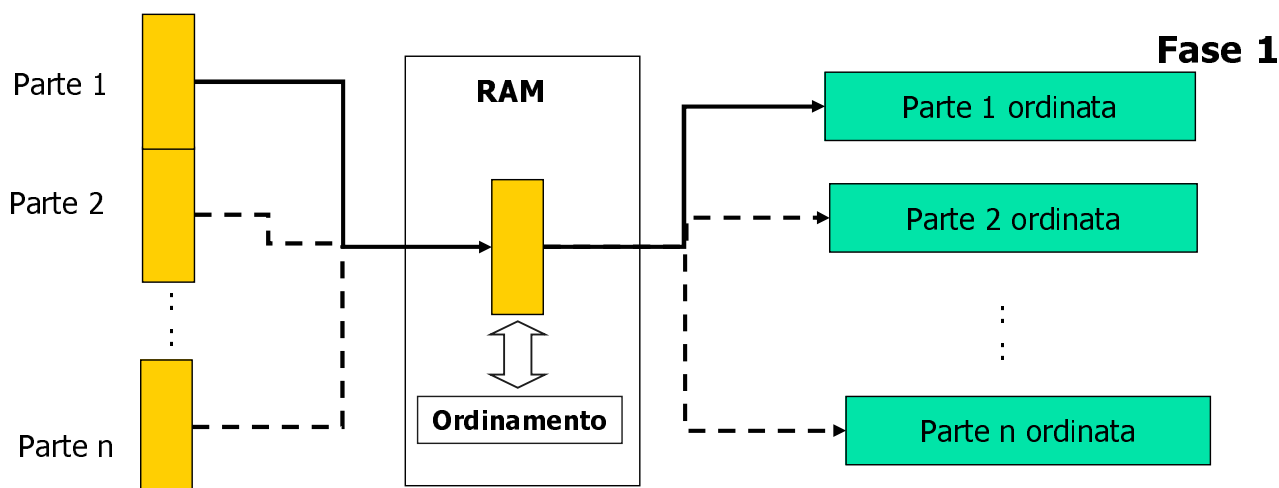
Two phase multiway merge-sort

- I dati sono divisi in parti della dimensione della RAM disponibile
- nella prima fase ogni parte è caricato in RAM, ordinata (ad. es. con quicksort) e riscritta in memoria secondaria
- nella seconda fase le parti sono unite insieme (merge) caricando i dati in RAM blocco per blocco

Un esempio: l'ordinamento II

Osservazioni

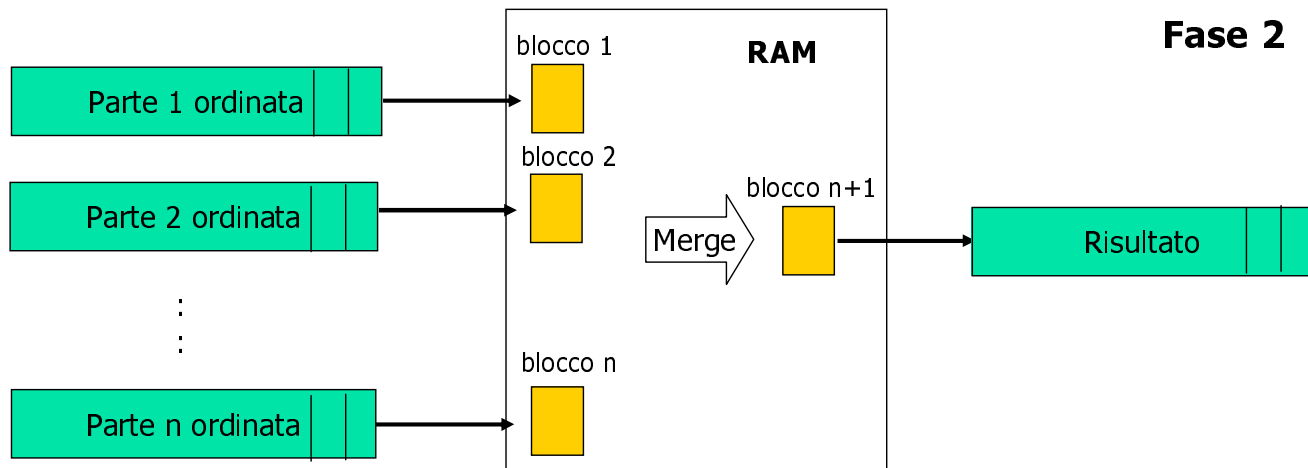
- Il file da ordinare si suddivide in parti della dimensione della RAM
- Si ordinano le parti una alla volta



Un esempio: l'ordinamento III

Osservazioni

- Si carica un blocco per ogni parte e si selezionano i record maggiori
- I ogni blocco viene caricato in memoria solo due volte



Scarselli Franco

Sistemi per basi di dati 2005-2006

13

Un esempio: l'ordinamento IV

La complessità del two phase multiway merge-sort

- Richiede $O(4 B(F))$ accessi alla memoria secondaria
 - $B(F)=n$ indica il numero dei blocchi del file
- Serve se $B(F) > M$
 - M indica i blocchi disponibili in memoria principale
- Funziona se $B(F) < M^2$
- Si ricorre ad algoritmi a più fasi se $B(F) > M^2$

Esempio

- Blocchi 64KB, RAM 1G
- Si può calcolare il numero dei blocchi della Memoria $M=2^{14}=16384$
- Se un file è più piccolo di 1G si applica algoritmi ad un passo
- Se un file è più grande di $M^2 = 2^{28}=268.435.456$ blocchi si usano algoritmi a 3 passi ($M^2 * 64K = 2^{44}=16Tera$)

Scarselli Franco

Sistemi per basi di dati 2005-2006

14



Migliorare l'accesso alla memoria secondaria

Soluzioni per migliorare l'accesso alla memoria secondaria

- Organizzare i dati sui dischi
 - dati correlati sono messi sullo stesso cilindro ed, eventualmente in blocchi contigui
 - vantaggioso se il modo in cui si accede ai dischi è prevedibile (es. fase 1 dell'ordinamento)
 - non utile in caso in cui il modo di accedere ai dischi sia imprevedibile (es. fase 2 dell'ordinamento o molti processi contemporanei)
- Prefetch
 - si cerca di prevedere i blocchi da caricare/scrivere e si mettono nel buffer
 - vantaggioso se sono prevedibili i blocchi richiesti ma non il momento in cui verranno richiesti (es. fase 2 dell'algoritmo)
 - richiede maggiore spazio per il buffer



Migliorare l'accesso alla memoria secondaria II

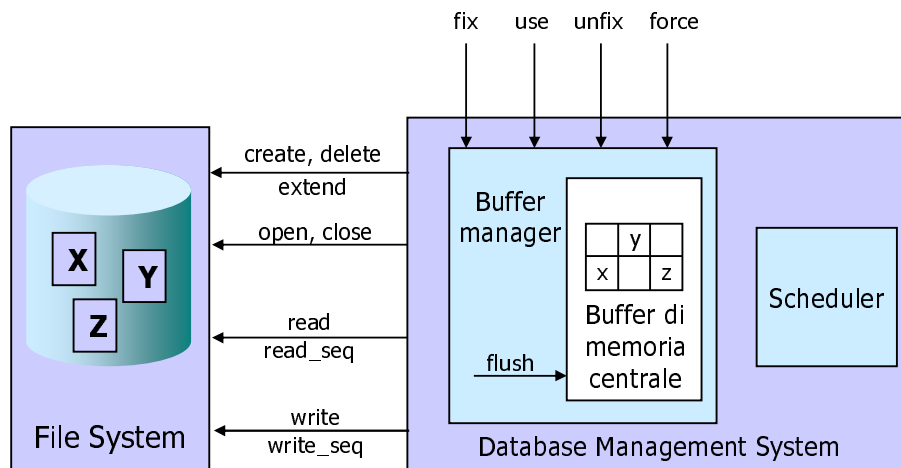
- Dischi multipli (es. RAID)
 - i dati sono distribuiti su n dischi
 - aumento di velocità di lettura e scrittura (per un fattore n), anche nel caso in cui i blocchi siano acceduti con un ordine imprevedibile
 - se blocchi che si vuole accedere sono sullo stesso disco non incrementa la velocità di accesso
 - il costo e i guasti aumentano con il numero dei dischi
- Mirroring
 - avere due o più dischi con lo stesso contenuto
 - accessi in lettura più veloci per qualsiasi applicazione
 - non ci sono problemi di blocchi che risiedono sullo stesso disco
 - costo dei dischi è maggiore

Migliorare l'accesso alla memoria secondaria III

- riordinare le richieste di lettura/scrittura
 - un algoritmo implementato dal controller o dal sistema operativo ordina le richieste di lettura/ scrittura (ad. es. l'algoritmo dell'ascensore)
 - riduce i costi di accesso per qualsiasi applicazione
 - valido se ci sono molte richieste di lettura scrittura per le quali i tempi di attesa sono lunghi

Gestione dei buffer

- **Buffer:** zona di memoria centrale preallocata e condivisa fra le transazioni





Organizzazione del buffer

- Il buffer è organizzato in pagine
 - La dimensione di una pagina è un multiplo della dimensione dei blocchi di ingresso-uscita utilizzati nelle letture/scritture sui dispositivi di memoria di massa
 - Dimensioni tipiche delle pagine variano fra 2Kb e 64Kb
 - La velocità di accesso ai record di pagine nel buffer è circa 10^6 volte maggiore
- Le politiche di gestione si basano sul principio di **località dei dati**
 - i dati referenziati di recente hanno maggiore probabilità di essere referenziati nel futuro
 - sono simili a quelle usate dai sistemi operativi per gestire la memoria secondaria



Organizzazione del buffer II

- Il **gestore del buffer** gestisce il trasferimento delle pagine fra la memoria principale e la memoria di massa
 - riceve richieste di lettura e scrittura che esegue accedendo alla memoria secondaria solo quando necessario
 - mette a disposizione alcune primitive: **fix, unfix, use, force, flush,...**
- Le **strutture dati del gestore del buffer** includono una tabella che per ogni pagina indica
 - il corrispondente blocco fisico
 - se la pagina è in uso o meno
 - se la pagina è stata modificata



Primitive

- **fix**

- si richiede l'accesso ad una pagina che viene caricata nel buffer
- restituisce il riferimento alla pagina nel buffer
- la pagina risulta **valida** e allocata alla transazione
- la lettura da disco è richiesta solo se la pagina non era già presente nel buffer

- **use**

- viene usata dalla transazione per accedere alla pagina caricata
- conferma l'allocazione della pagina nel buffer e lo stato di **valida**



Primitive

- **unfix**

- indica che la transazione ha terminato di usare la pagina
- la pagina passa nello stato di **non valida**

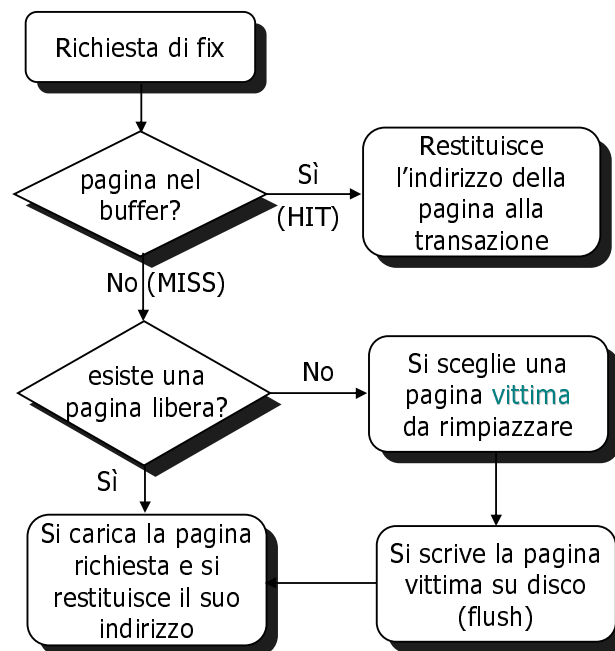
- **force**

- scrive una pagina in memoria di massa
- la scrittura è **sincrona** con la richiesta e la transazione si sospende fino a che non è terminata l'esecuzione della primitiva

- **flush**

- scrive su disco le pagine non più valide e inattive da più tempo
- la scrittura è **asincrona** e indipendente dalle transazioni
- rende **libere** le pagine del buffer salvate su disco
- può essere decisa dal gestore del buffer per **migliorare l'efficienza**

Esecuzione di fix



Politiche di gestione

- **Scelta della pagina vittima**
 - **steal**
Si possono selezionare anche le pagine attive di un'altra transazione
 - le pagine possono essere scritte prima del termine della transazione
 - può essere necessario recuperare il valore iniziale nel caso di un abort
 - **no-steal (*)**
- **Scrittura delle pagine**
 - **force**
Le pagine attive di una transazione sono scritte in sincrono col commit
 - **no-force (*)**
La scrittura dipende dalle decisioni del buffer manager (asincrona)
- **Pre-fetching/pre-flushing**



DBMS e file system

Il DBMS gestisce la memoria secondaria, ma il sistema operativo gestisce file system: come interagiscono ??

■ Soluzione 1

Il DBMS usa un file per ogni tabella e ogni indice

- Eventualmente, anche la gestione del buffer puo' essere lasciata al sistema operativo

■ Soluzione 2

Il DBMS usa alcuni file

- L'allocazione dei dati all'interno dei file e' gestita dal DBMS
 - il DBMS decide come allocare i blocchi del file e come allocare i record all'interno del file
 - spesso i DBMS usano un solo file
- La creazione e l'allocazione dei file sono gestite dal sistema operativo



DBMS e file system

■ Soluzione 3

Una zona del disco viene allocata al DBMS

- Il DBMS gestisce autonomamente la zona allocata
- il DBMS puo' allocare in modo contiguo dati che accede in maniera sequenziale,

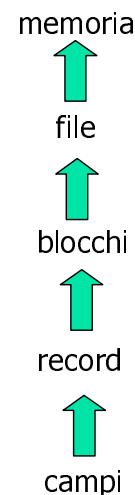
Vantaggi e svantaggi

- soluzione 3 implica **massima efficienza**, soluzione 1 minima efficienza
- soluzione 3 implica **massima affidabilita'**, soluzione 1 minima affidabilita'
- soluzione 1 implica **minimo costo di sviluppo** del DBMS
- la maggior parte dei DBMS usano soluzione 2, ma permettono anche soluzione 3
- man mano che i sistemi operativi diventano piu' efficienti e affidabili si va verso soluzione 1

Organizzazione dei dati

I dati sono organizzati in una gerarchia

- **campi**: contengono dati elementari (char, integer,...) rappresentati come noto
- **record**: insiemi di campi che costituiscono una riga di una tupla
- **blocchi**: unita' minima leggibile dalla memoria secondaria, assumeremo blocco=pagina
- **file**: **non corrisponde** necessariamente al file del file system. È un insieme di dati correlati



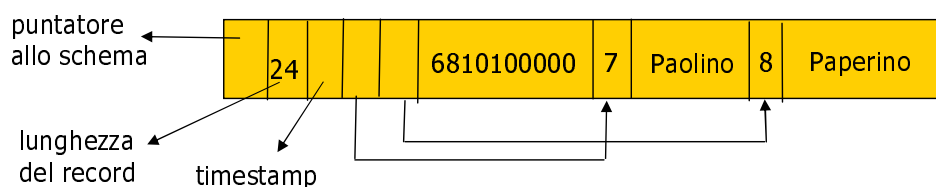
I record

Insieme ai dati del record si può memorizzare un **header**

- puntatore allo schema
- lunghezza del record
- timestamp
- altre informazioni

Osservazioni

- l'header **può essere omesso** se tutti i record di uno schema appartengono alla stessa tabella
- l'header contiene **puntatori all'inizio dei campi** a dimensione **variabile**



```
CREATE TABLE studenti(  
matricola CHAR(9),  
nome VARCHAR(15),  
cognome VARCHAR(15)  
)
```

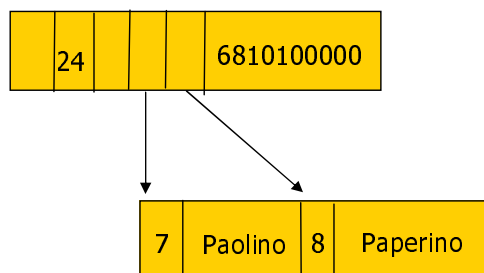
I record a dimensione variabile

I campi a dimensione variabile possono essere **memorizzati in un altro blocco**

- il record è piu' piccolo: ricerche piu' veloci
- accesso al campo variabile piu' lento
- compromesso: campi a dimensione variabile memorizzati in un altro blocco, ma sullo stesso cilindro

In SQL 3, record a dimensione variabile si incontrano anche a causa di

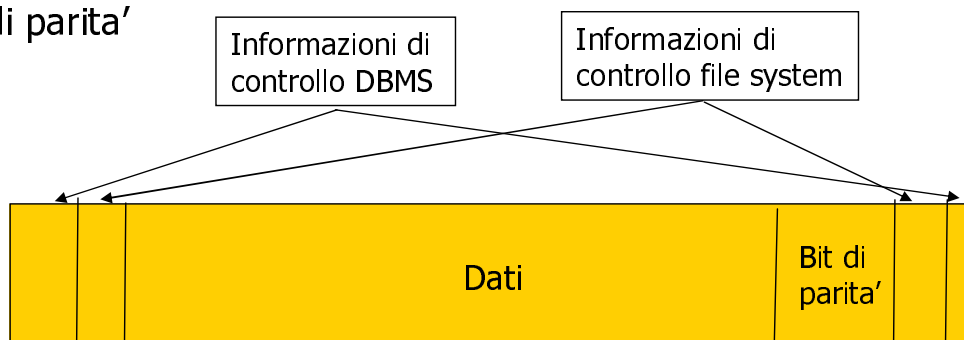
- campi multipli
- record a formato variabile
- BLOB



Organizzazione dei record nelle pagine

Tipicamente **una pagina contiene**

- informazioni di controllo del file system
- informazione di controllo del DBMS
 - dizionario di pagina, numero di record contenuti nella pagina, tipo di oggetto (tabella, indice, ...), spazio libero, puntatore all'oggetto successivo, ...
- dati
- bit di parità'



Organizzazione dei record nelle pagine II

Il **dizionario di pagina** contiene

- i puntatori ai record contenuti nella pagina
- i puntatori e i dati crescono in direzioni opposte
- altre informazioni sui record
- con il dizionario, i puntatori ai record sono più semplici e puntano al dizionario



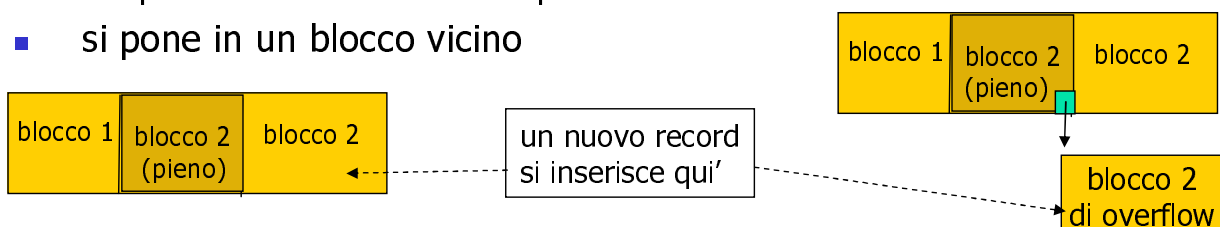
Inserimenti, cancellazioni e modifiche

Cancellare un record in un blocco

- marcare il record come eliminato
 - occorre un bit di validità per marcare l'eliminazione
 - lo spazio può essere (parzialmente) riutilizzato
- rimuovere il record e riorganizzare
 - occorre fare attenzione ai puntatori

Inserire un record in una pagina, se non esiste un posto libero

- si crea un blocco di overflow
 - può essere necessario un puntatore in avanti
- si pone in un blocco vicino





Organizzazione dei record nelle pagine III

Record e pagine: **osservazioni**

- se le tuple hanno lunghezza fissa, una pagina contiene $\lfloor L_p/L_t \rfloor$ record (L_p = lunghezza pagina, L_t = lunghezza tupla)
- in generale L_p/L_t non e' intero e in una pagina puo' rimanere spazio non occupato
- i DBMS possono permettere
 - che un record sia allocato su pagine differenti
 - il caso in cui $L_t > L_p$
(è possibile allocare record piu' grandi della dimensione della pagina)



Strutture per l'accesso ai dati

Le strutture dati usate nelle basi di dati si dividono in

- **primarie**
 - usate per le tabelle
 - contengono i dati veri e propri e strutture che ne facilitano l'accesso
- **secondarie**
 - usate per indici
 - non contengono i dati

Le strutture primarie

- **accesso sequenziale**
- **accesso calcolato**
- **ad albero**



Strutture ad accesso sequenziale

Nelle strutture ad accesso sequenziale:

- sono ordinate in sequenza secondo un qualche criterio
- si dividono in: **array, ad accesso seriale, ordinate**

array

- le posizioni sono individuate da indici
- utili se è possibile individuare un indice e le tuple hanno dimensione fissa
 - condizioni che si verificano raramente
- molto efficienti per la lettura/scrittura di una tupla



Strutture ad accesso sequenziale II

seriale

- ordinamento fisico che non dipende dal contenuto delle tuple
- gli inserimenti vengono effettuati
 - in coda (con riorganizzazioni periodiche)
 - al posto di record cancellati
- è molto diffusa per strutture primarie, associate a indici secondari
- è molto efficiente quando si vogliono leggere/modificare tutti gli elementi di una tabella



Strutture ad accesso sequenziale III

ordinate

- l'**ordinamento fisico** che dipende dal contenuto di **un campo delle tuple**
- l'accesso è efficiente
 - si usa una ricerca dicotomica
- l'inserimento di nuove tuple risulta problematico
 - lasciare degli spazi vuoti e riordinare localmente
 - inserire nuovi blocchi nel file
 - usare delle pagine di overflow
- vengono usate per implementare contemporaneamente una tabella e un indice (**ISAM** Index Sequential Access Method)



Strutture ad accesso calcolato (Tabelle hash)

tabelle hash e array

- è possibile usare un array per memorizzare un insieme di record se
 - esiste una chiave per la quale il numero dei valori possibili sia **paragonabile** al numero dei record
 - esempio: memorizzare 1000 studenti usando dei numeri di matricola che vanno da 1 a 1000
- se i possibili valori della chiave sono **molti di più** dei record, l'array spreca troppo spazio
 - esempio: il numero di matricola usa 10 caratteri

Soluzione

- si usa una funzione (hash) che **associa ad ogni chiave un indirizzo** in uno spazio (di dimensione leggermente superiore al numero di tuple da memorizzare)



Funzioni hash

Una **funzione hash**

- È una funzione h tale che $h(\text{key})=i$:
 - key è una chiave
 - i un indirizzo di una tabella

Meccanismi per generare funzioni hash

- divisione per numeri primi
- trasformazioni basate su cambio della base
- folding
- calcolo della radice



Tabelle hash e collisioni

Le **collisioni**

- la funzione hash **non può essere iniettiva**: esiste la possibilità di collisioni
- le buone funzioni hash distribuiscono gli indirizzi in modo uniforme
 - le probabilità di collisione sono ridotte
- soluzioni
 - mettere l'elemento nella prima posizione libera
 - fare un blocco di overflow
 - funzioni hash "alternative"



- numero medio di accessi: 1,125

60600	0	201159	9	200268	18	200430	30	102690	40
66301	1	200459	9	205619	19			115541	41
205751	1	205610	10	210519	19			206092	42
205802	2	201260	10	200419	19	210533	33	205693	43
200902	2	102360	10	205724	24				
116202	2	205460	10					205845	45
200604	4	205912	12					200296	46
66005	5	205762	12	205977	27	205887	37	205796	46
116455	5	205617	17	205478	28	200138	38		
200205	5	205667	17			102338	38	206049	49



- la funzione hash produce l'indirizzo di un blocco
- le tuple del blocco sono organizzate in modo sequenziale
- diminuisce la probabilita' di overflow fra pagine diverse

- 1 collision2
 - numero medio di accessi:
1,025

60600	66301	205802	200268	200604
66005	205751	200902	205478	201159
116455	115541	116202	210533	200419
200205	200296	205912	200138	205619
205610	205796	205762	102338	205724
201260		205617	205693	206049
102360		205667		210519
205460		206092		200459
200430		205977		
102690		205887		
205845				

File hash II

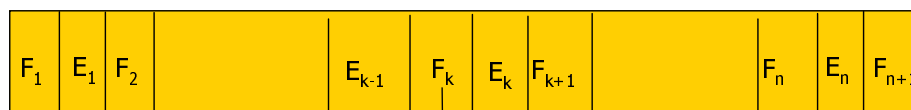
L'accesso tramite file hash

- È molto efficiente per l'accesso diretto basato su valori della chiave con condizioni di uguaglianza:
 - costo medio di poco superiore all'unità
- Non è efficiente per ricerche basate su intervalli e ricerche non basate sulla chiave
- I file hash "degenerano" se si riduce lo spazio sovrabbondante
 - funzionano con file la cui dimensione non varia molto
 - in tal caso occorre riordinare il file

Strutture basate su alberi di ricerca

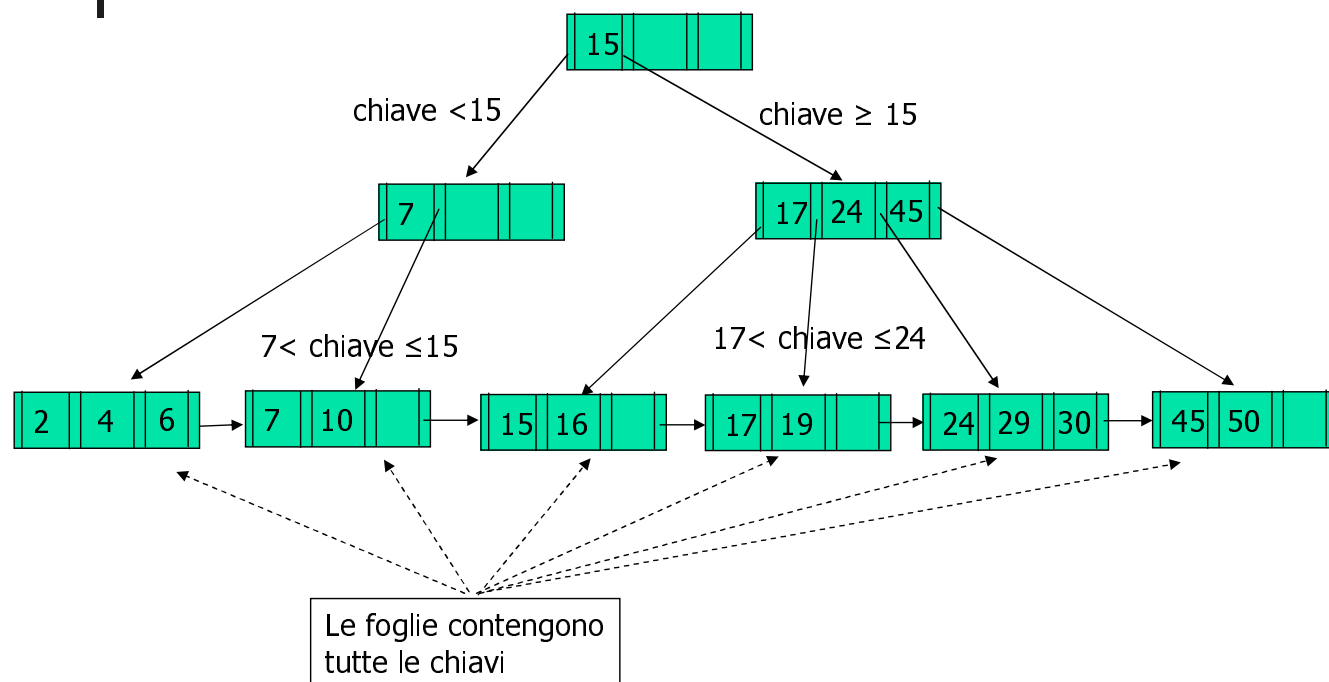
Alberi di ricerca di ordine $n+1$

- Ogni nodo ha $n+1$ figli e n etichette,
- Nell' i -esimo sottoalbero abbiamo tutte etichette maggiori della $(i-1)$ -esima etichetta e minori della i -esima
- Ogni ricerca comporta la visita di un cammino radice foglia
- Per rendere massimo il fan-out, un nodo si sceglie n in modo che ogni nodo corrisponda ad un blocco



punta alle tuple le cui chiavi c soddisfano $E_{k-1} < c \leq E_k$

B+ Tree: un esempio



Scarselli Franco

Sistemi per basi di dati 2005-2006

45

B+ tree

Un **B+ tree** è un albero di ricerca che

- viene mantenuto bilanciato
 - riempimento parziale
 - ogni nodo interno contiene almeno $\lceil (n+1)/2 \rceil - 1$ chiavi
 - ogni foglia contiene almeno $\lceil (n+1)/2 \rceil$ chiavi
 - Riorganizzazioni (locali) in caso di sbilanciamento

Inserimenti

- si fa una ricerca per trovare il punto di inserimento
- se non c'è posto nella foglia il nodo va suddiviso
- la suddivisione può propagarsi eventualmente fino alla radice

Cancellazioni e modifiche

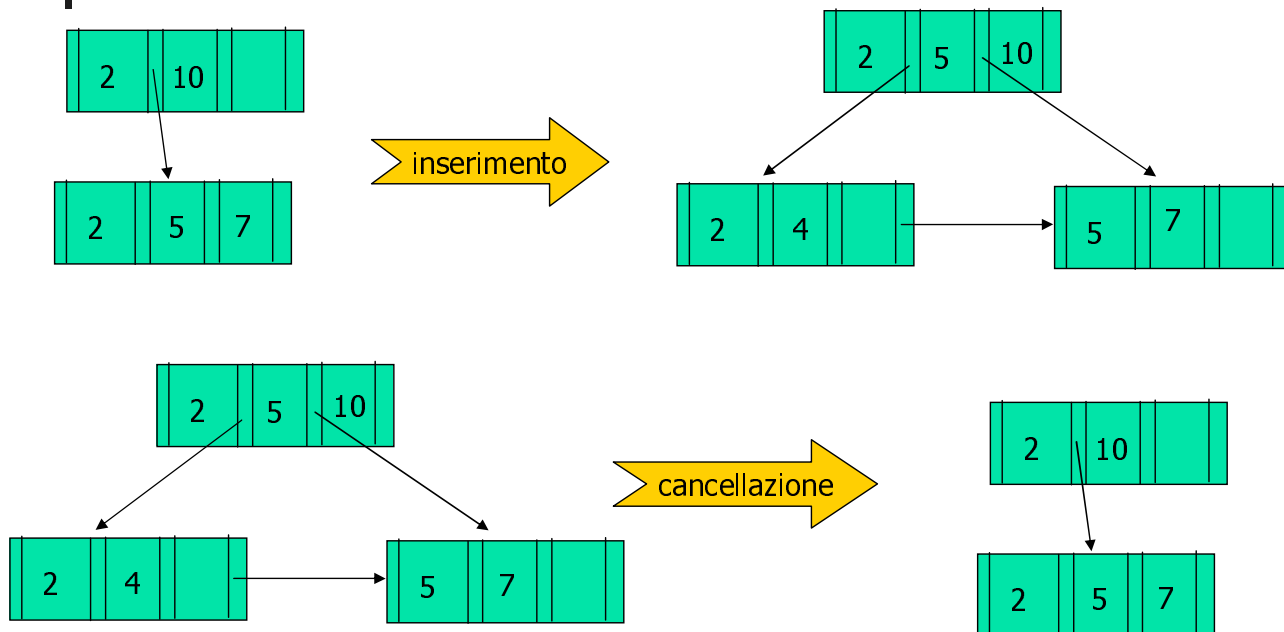
- le eliminazioni possono portare a riduzioni di nodi
- le modifiche si trattano come eliminazioni seguite da inserimenti

Scarselli Franco

Sistemi per basi di dati 2005-2006

46

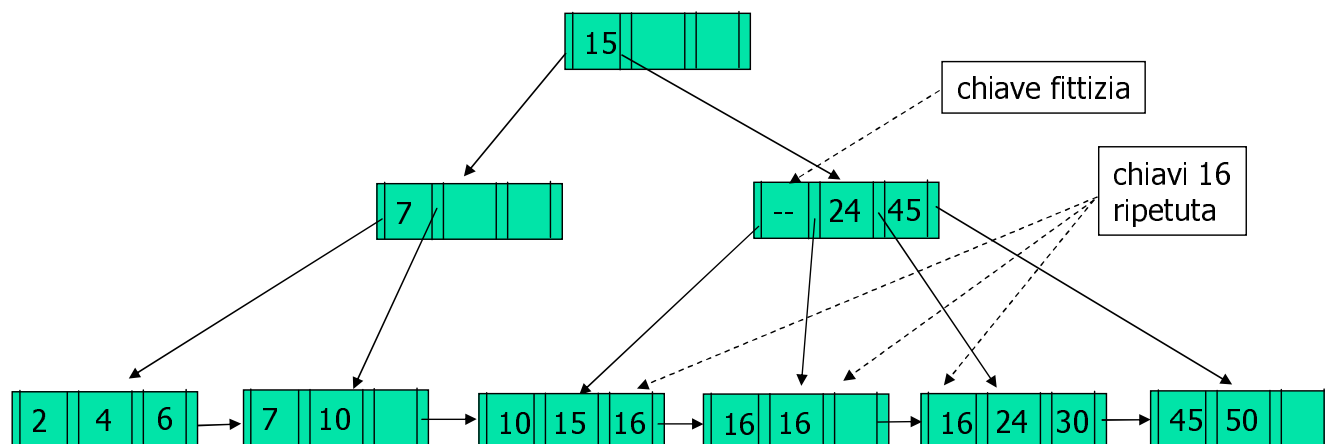
Inserimenti e cancellazioni



B+ Tree con chiavi ripetute

Cosa cambia quando ci sono **chiavi ripetute**?

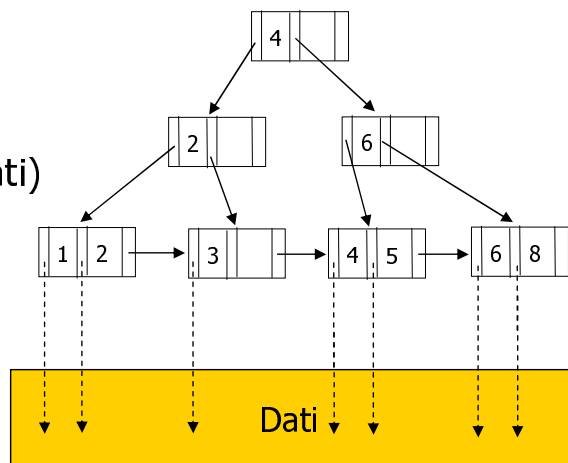
- il puntatore destro indica la foglia in cui si trova la **prima occorrenza** della chiave
- i nodi possono contenere **chiavi fittizie** (vuote)



B+ Tree e B tree

B+ tree

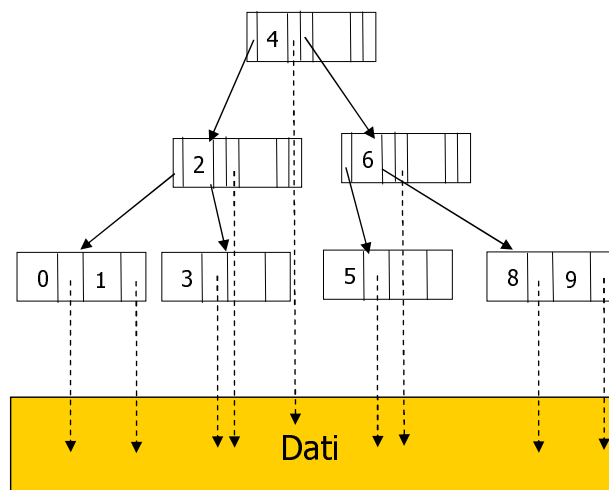
- le foglie contengono tutte le chiavi
 - alcune chiavi sono ripetute nei nodi interni
- le foglie sono collegate in una lista
- le foglie puntano (o contengono i dati)
- ottimi per le ricerche su intervalli
- molto usati nei DBMS



B+ Tree e B tree II

B tree

- le chiavi non sono ripetute
- ogni nodo punta ai dati corrispondenti
- la struttura dati è piu' piccola rispetto ai B+ tree
- le foglie sono diverse dai nodi interni
- meno adatti alle ricerche su intervalli
- prestazioni peggiori in caso di accesso condiviso in modifica





Efficienza dei B-Tree

Quanto sono efficienti i B-Tree ?

- con piccole chiavi un B-Tree a tre livelli puo' indicizzare la maggior parte delle tabelle
- poiche' la radice dell'indice puo' essere tenuta nel buffer, tre accessi sono sufficienti a trovare il puntatore ad un record

Esempio

- Blocchi 4KB, chiavi intere 4B, puntatori 8B
- Si puo' calcolare il numero di chiavi per nodo:
il valore n massimo t.c. $4n + 8(n+1) < 4096$ implica $n=340$
- Supponendo che il riempimento dei nodi sia medio avremo: 255 figli per nodo
- Un B-Tree con tre livelli ha: $255^3 = 16.581.375$ foglie
- Se le foglie puntano a blocchi: si puo indicizzare 68G



Indici

indice primario

- su un campo sul cui ordinamento è basata la memorizzazione (es. indice generale di un libro)

```
CREATE CLUSTERED INDEX mio_indice_primario ON studenti(matricola)
```

indice secondario

- su un campo con ordinamento diverso da quello di memorizzazione (es. indice analitico di un libro)

```
CREATE INDEX mio_indice_secondario ON studenti(cognome)
```



Indici II

Osservazioni

- Ogni file può avere al più un indice primario e un numero qualunque di indici secondari
 - es. una guida turistica può avere l'indice dei luoghi e quello degli artisti
- Un file hash o ad accesso sequenziale non può avere un indice primario
- L'accesso a file con indice richiede $O(\log n)$, migliore dell'accesso sequenziale, peggiore dell'accesso a file hash



Indici III

indice denso

- contiene un record per ciascun record del file

indice sparso

- contiene solo alcuni record del file e ha puntatori ai blocchi in cui sono contenuti i record

Osservazioni

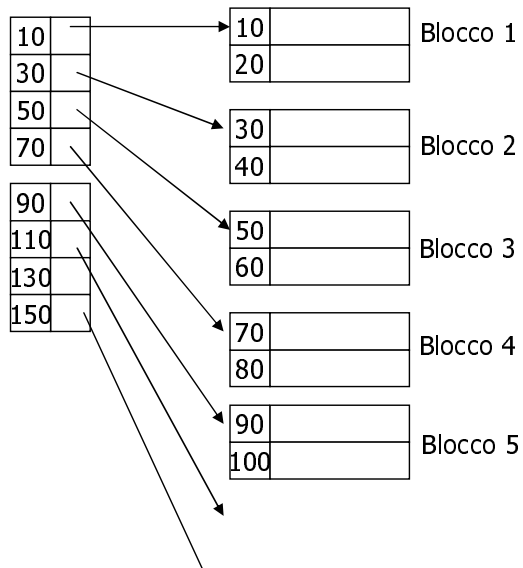
- Un indice primario può essere sparso, uno secondario deve essere denso
- gli indici sparsi occupano meno spazio
- gli indici densi permettono di far alcune operazioni senza accedere ai dati

```
SELECT * FROM studenti
WHERE  cognome LIKE 'B%'
      AND data_nascita >= '1/1/1980'
```

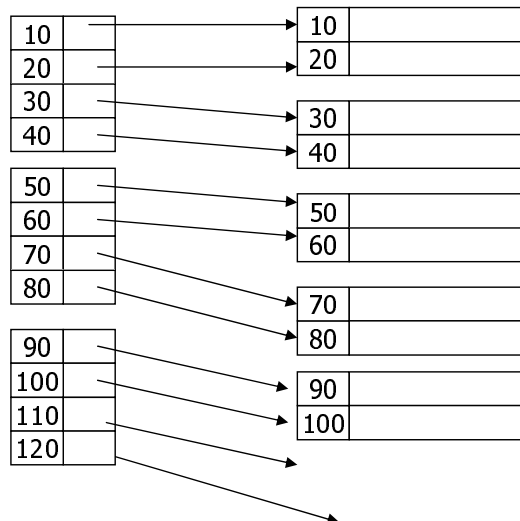


Un esempio: indici primari

Indice sparso

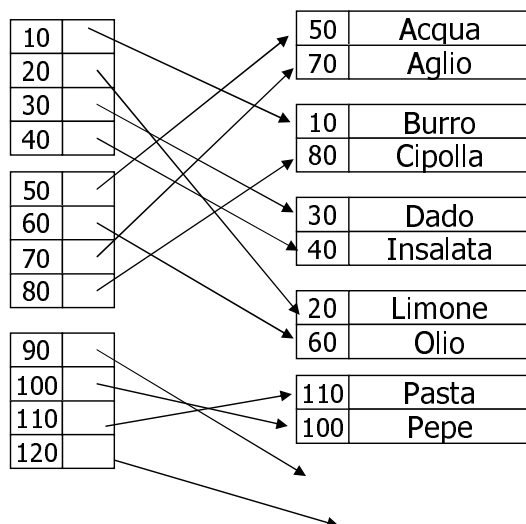


Indice denso



Un esempio: indice secondario

Indice denso





Inserimenti, cancellazioni e update

Cosa comporta la modifica dei dati ?

Azione	Indice denso	Indice sparso
Creare un blocco vuoto di overflow	Nessuna modifica	Nessuna modifica
Eliminare un blocco vuoto di overflow	Nessuna modifica	Nessuna modifica
Creare un blocco vuoto in sequenza	Nessuna modifica	Inserimento
Eliminare un blocco vuoto in sequenza	Nessuna modifica	Cancellazione
Inserire un record	Inserimento	Nessuna o modifica
Cancellare un record	Cancellazione	Nessuna o modifica
Modificare un record	Modifica	Nessuna o modifica



Indici: osservazioni

- Accesso diretto (per chiave) efficiente
 - sia puntuale che per intervalli
- 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

Documenti di testo

In information retrieval, un **documento testuale** è rappresentabile con

- un record con un numero di campi uguale alla **dimensione del vocabolario**
- l'i-esimo campo è **True** o **False** a seconda se l'i-esima parola del vocabolario sia presente o meno nel documento

Questo è un documento che parla di Linux



a	abbaiare	abbaiano	...	documento	...	parla	...	linux	...
False	False		...	True	...	True	...	True	...

Per accedere efficacemente ai documenti che contengono una data parola

- si crea un unico indice, detto **indice inverso**, che combina gli indici su ogni campo
- l'indice **punta solo a documenti che contengono le parole** (True) e non a quelli che non le contengono (False)

Documenti di testo II

Gli indici inversi

- usano dei buckets (indicizzazione indiretta) per gestire le ripetizioni
- possono contenere ulteriori informazioni sulle parole
 - posizione nel documento dove si trova parola
 - caratteristiche del formato della parola (bold, nel titolo, carattere normale, ..)

