

La struttura interna dei DBMS



Caratteristiche dei DBMS

Devono garantire

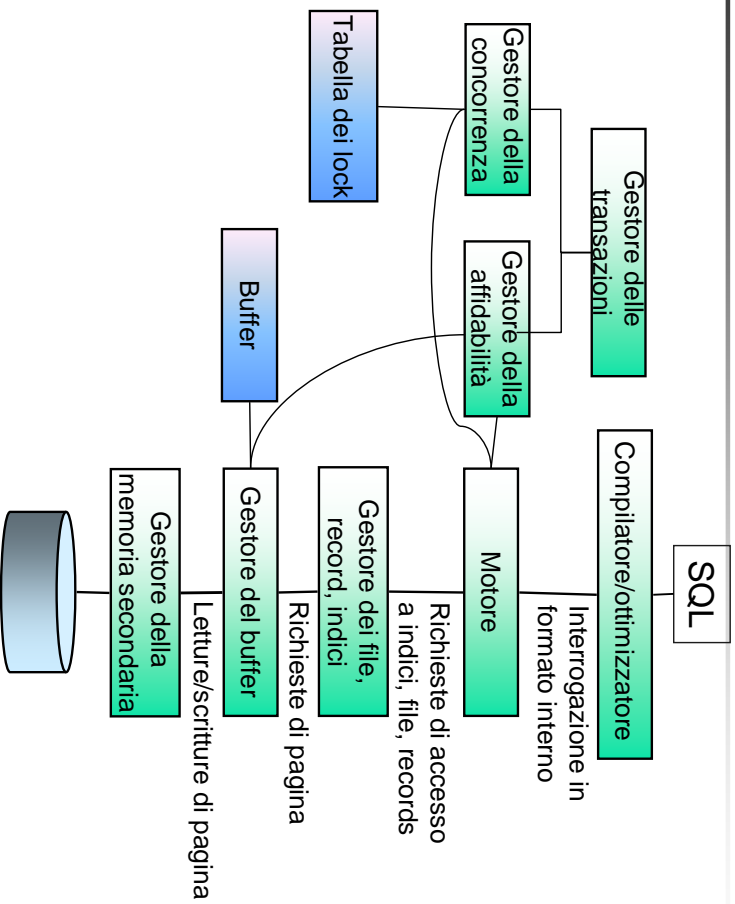
- **Affidabilità**
(Non si perdono informazioni in caso di malfunzionamento)
- **Privatizza**
(Restrizione degli accessi)
- **Efficienza**
(Il sistema svolge inserimenti, modifiche, ricerche in un tempo ragionevole)

Gestiscono **collezioni di dati**:

- **Grandi**
- **Persistenti**
- **Condivise**



Le componenti di un DBMS

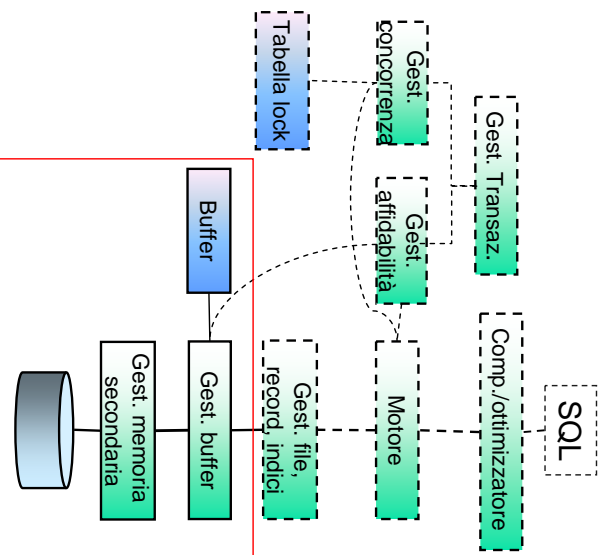


Scarselli Franco

Sistemi per basi di dati 2006-2007

3

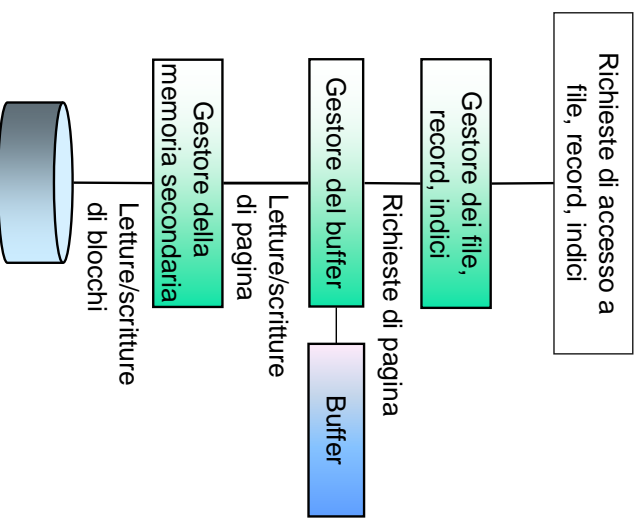
Gestione della memoria secondaria



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



Scarselli Franco

Sistemi per basi di dati 2006-2007

5

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

Scarselli Franco

Sistemi per basi di dati 2006-2007

6



Memoria secondaria (dischi) II

L'accesso alla memoria secondaria

- È estremamente più costoso dell'accesso alla memoria primaria
- 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**
- Con i DBMS, spesso il tempo speso in accesso alla memoria primaria e' ininfluente:
 - la complessità delle operazioni si può misurare in termini di **numero di accessi alla memoria secondaria**

Per questo motivo

- **Gli algoritmi e le strutture date usati nei DBMS possono essere molto diverse da quelle tradizionali!!**

Scarselli Franco

Sistemi per basi di dati 2006-2007

7



Un esempio: l'ordinamento

Cosa succede quando la memoria primaria non contiene i dati?

Per l'ordinamento si utilizza un algoritmo divide et impera

- Si suddividono i dati in sottoparti, in modo che ogni parte sia contenuta 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

Scarselli Franco

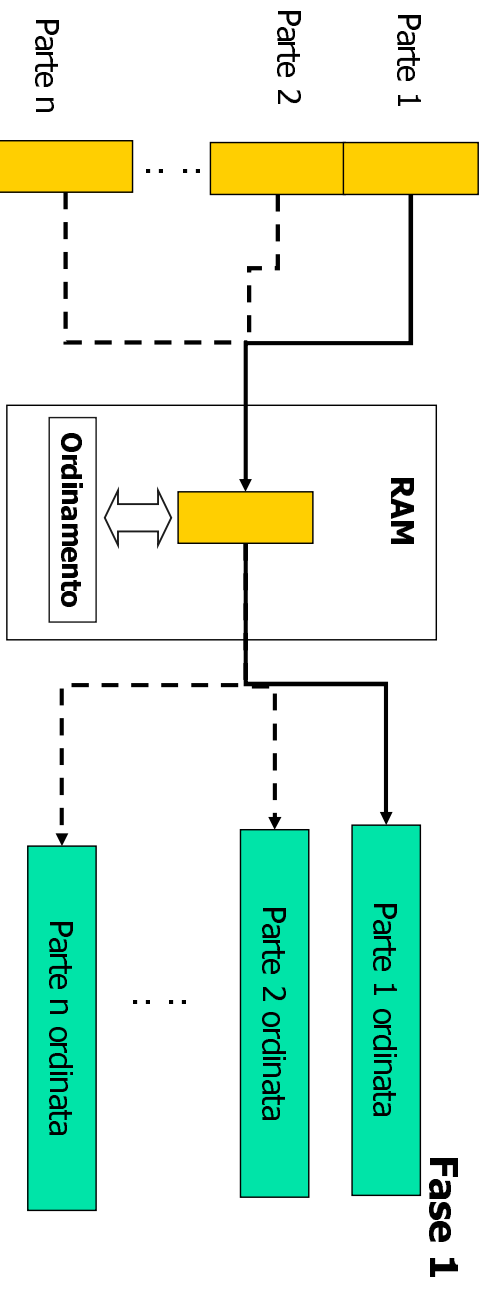
Sistemi per basi di dati 2006-2007

8

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



Scarselli Franco

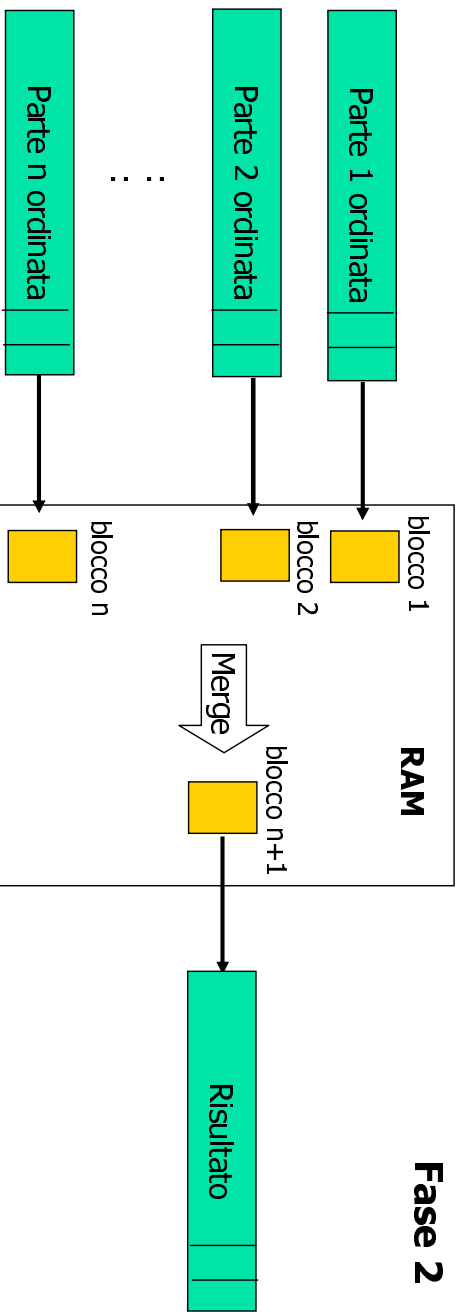
Sistemi per basi di dati 2006-2007

9

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 2006-2007

10

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 2006-2007

11

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)
- 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

Scarselli Franco

Sistemi per basi di dati 2006-2007

12

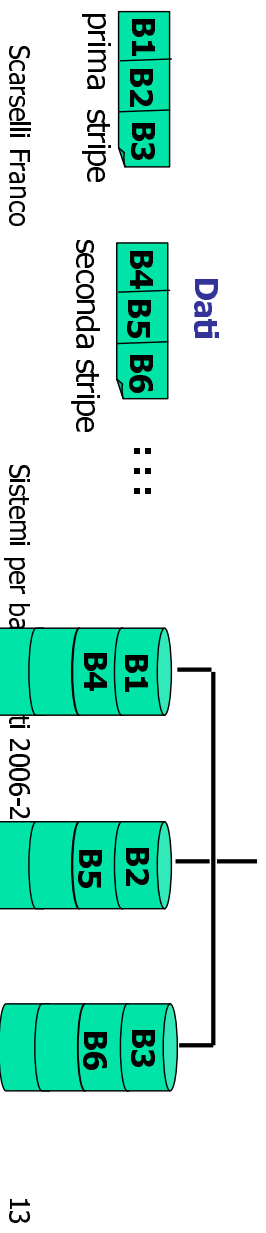
Migliorare l'accesso alla memoria secondaria: dischi multipli

■ Dischi RAID

- i dati sono partizionati in unità di uguale lunghezza chiamate **stripe** e distribuiti su n dischi (ad esempio con una strategia round robin): le stripe possono essere scritte e lette in parallelo

■ RAID 0

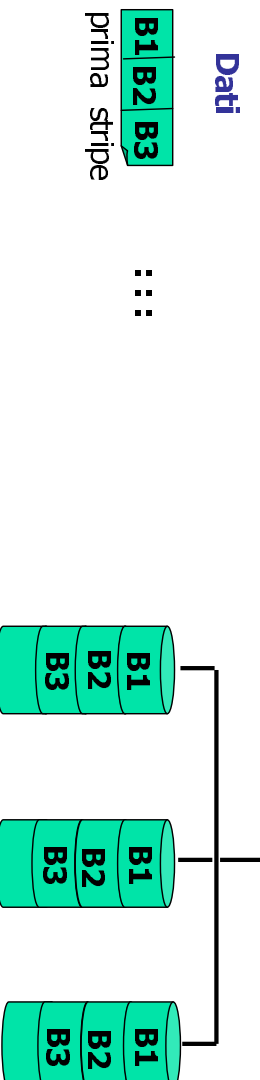
- i dati sono distribuiti su entrambi i dischi
- aumento di velocità di lettura e scrittura (per un fattore n), anche nel caso in cui i blocchi siano acceduti con un ordine imprevedibile
- diminuzione dell'affidabilità (MTBF) di un fattore n



Migliorare l'accesso alla memoria secondaria: dischi multipli II

■ RAID 1 (mirroring)

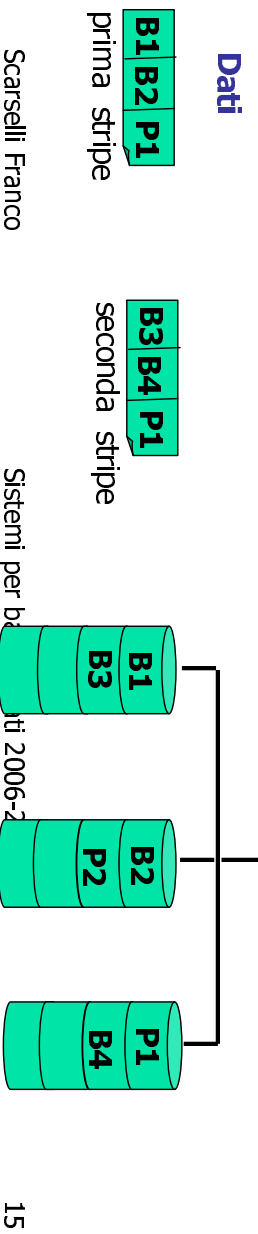
- i dati sono duplicati e una copia viene memorizzata su ogni disco
- aumento di velocità di lettura, ma non di scrittura
- aumento dell'affidabilità di un fattore n



Migliorare l'accesso alla memoria secondaria: dischi multipli III

■ RAID 5

- ogni stripe è costituita da $n-1$ blocchi più un blocco di parità:
 - i blocchi vengono distribuiti su dischi diversi
 - i blocchi di parità permettono di ricostruire le stripe in caso di malfunzionamento di uno dei fischi
- aumento di velocità di lettura e scrittura all'incirca di un fattore $n-1$ (ad eccezione di scritture o letture di blocchi sullo stesso disco)
- aumento dell'affidabilità all'incirca di un fattore $n-1$ (anche se più dischi insieme tendono a riscaldarsi di più, a rompersi insieme,)



Migliorare l'accesso alla memoria secondaria: dischi multipli IV

■ Dischi diversi per strutture dati diverse

- Il sistemista decide come distribuire le tabelle, gli indici e i log
- L'aumento delle prestazioni e dell'affidabilità dipende dal particolare database

Migliorare l'accesso alla memoria secondaria: buffer e prefetch

- **Buffer e 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

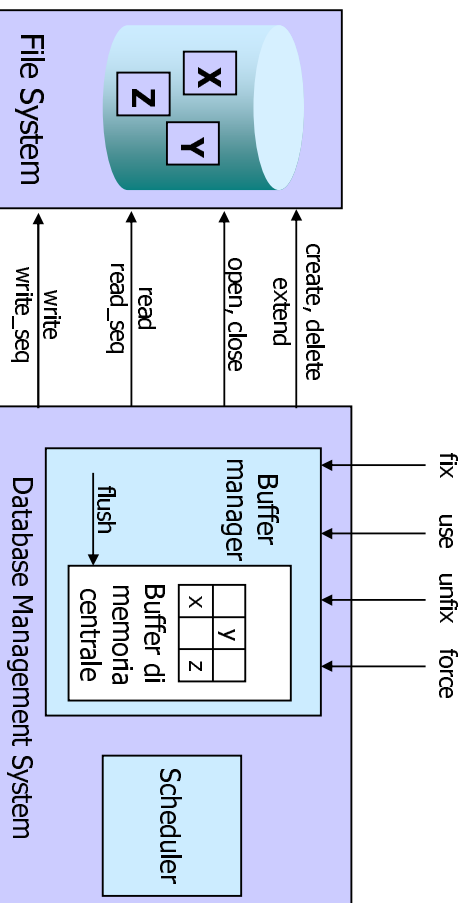
Scarselli Franco

Sistemi per basi di dati 2006-2007

17

Gestione dei buffer

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



Scarselli Franco

Sistemi per basi di dati 2006-2007

18



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

Scarselli Franco

Sistemi per basi di dati 2006-2007

19



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 e' in uso o meno
 - se la pagina e' stata modificata

Scarselli Franco

Sistemi per basi di dati 2006-2007

20



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**

Scarselli Franco

Sistemi per basi di dati 2006-2007

21



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 le transazione si sospende fino a che non è terminata l'esecuzione della primitiva

- **flush**

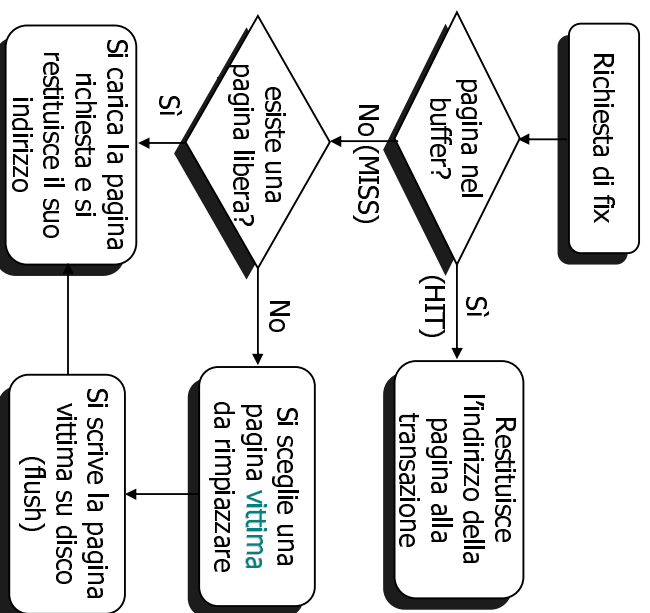
- scrive su disco le pagine non più valide e inattive da più tempo
- la scrittura è **asincrona** e indipendente dalle transazioni
- rende **libere** le pagine del buffer salvate su disco
- può essere decisa dal gestore del buffer per **migliorare l'efficienza**

Scarselli Franco

Sistemi per basi di dati 2006-2007

22

Esecuzione di fix



Scarselli Franco

Sistemi per basi di dati 2006-2007

23

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

Scarselli Franco

Sistemi per basi di dati 2006-2007

24



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 può essere lasciata al sistema operativo

- **Soluzione 2**

Il DBMS usa alcuni file

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

Scarselli Franco

Sistemi per basi di dati 2006-2007

25



DBMS e file system

- **Soluzione 3**

Una zona del disco viene allocata al DBMS

- Il DBMS gestisce autonomamente la zona allocata
- il DBMS può 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 affidabilità**, soluzione 1 minima affidabilità
- 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 più efficienti e affidabili si va verso soluzione 1

Scarselli Franco

Sistemi per basi di dati 2006-2007

26

Organizzazione dei dati

I dati sono organizzati in una gerarchia

- **campi**: contengono dati elementari (char, integer,...) rappresentati come noto
- **record**: una riga di una tupla
- **blocchi**: unità minima leggibile dalla memoria secondaria, assumeremo blocco=pagina

- **file**: **non corrisponde** necessariamente al file del file system. È un insieme di dati correlati

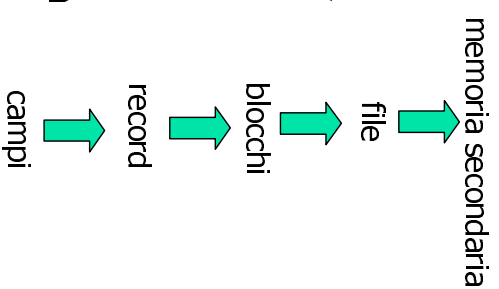
Per ogni elemento della gerarchia

- occorre scegliere la struttura dati da usare per allocarlo in memoria secondaria (all'interno dell'elemento figlio)
- ogni DBMS ha le proprie soluzioni: di seguito vediamo alcuni esempi

Scarselli Franco

Sistemi per basi di dati 2006-2007

27



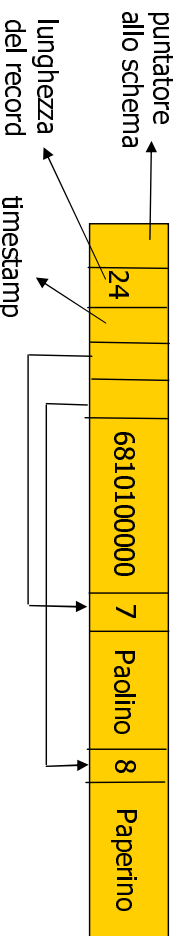
I campi nei record

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)  
)
```

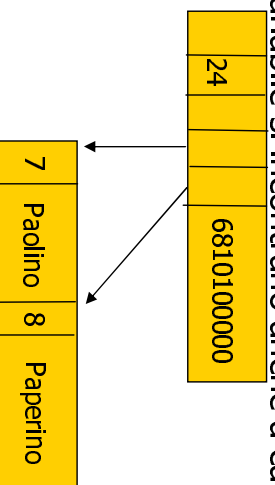
I record a dimensione variabile

I campi a dimensione variabile possono essere **memorizzati anche 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
- adatta per campi grandi

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

- campi multipli
- record a formato variabile
- BLOB



Scarselli Franco

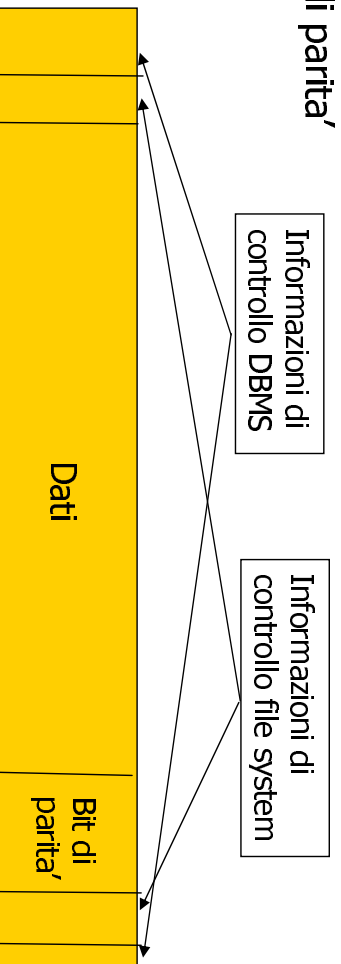
Sistemi per basi di dati 2006-2007

29

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'



Scarselli Franco

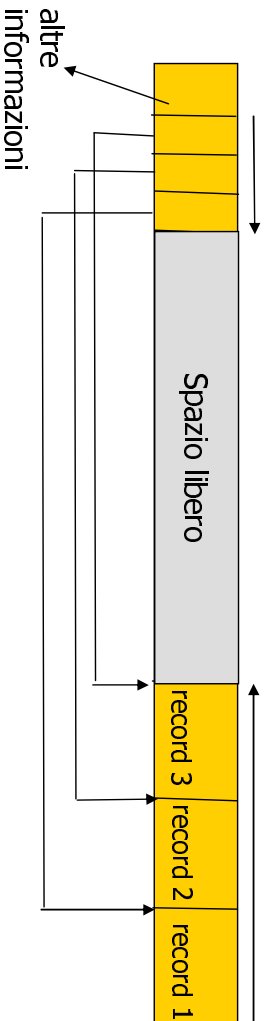
Sistemi per basi di dati 2006-2007

30

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



Scarselli Franco

Sistemi per basi di dati 2006-2007

31

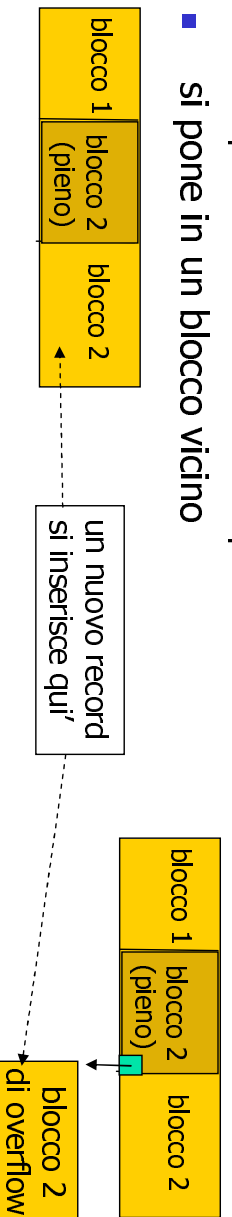
Inserimenti, cancellazioni e modifiche

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 rifiutato)
- 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



Scarselli Franco

Sistemi per basi di dati 2006-2007

32



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 è intero e in una pagina può rimanere spazio non occupato
- i DBMS possono anche permettere
 - che un record sia allocato su pagine differenti
 - il caso in cui $L_t > L_p$(è possibile allocare record più grandi della dimensione della pagina)

Scarselli Franco

Sistemi per basi di dati 2006-2007

33



Strutture per l'accesso ai dati

- Fino ad adesso abbiamo visto come i dati sono memorizzati, ma ... come sono ordinati e com'è possibile ritrovarli ?

Le strutture dati

- Nelle basi di dati le strutture dati usate servono ad organizzare i dati e a garantire un accesso efficiente
- Si dividono in **primarie** e **secondarie**

Le strutture dati secondarie

- non contengono i dati, ma facilitano l'accesso ai
- nei database relazionali sono gli indici

Scarselli Franco

Sistemi per basi di dati 2006-2007

34



Strutture per l'accesso ai dati

Strutture **primarie**

- contengono i dati veri e propri e le strutture che ne facilitano l'accesso
- implementano le tabelle
- possono implementare anche un indice

Le strutture primarie

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

Scarselli Franco

Sistemi per basi di dati 2006-2007

35



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

Scarselli Franco

Sistemi per basi di dati 2006-2007

36



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

File 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)

Scarselli Franco

Sistemi per basi di dati 2006-2007

39



Funzioni hash

Una **funzione hash**

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


Meccanismi per generare funzioni hash

- trasformazioni basate su cambio della base
- folding
- calcolo della radice
- ...

Scarselli Franco

Sistemi per basi di dati 2006-2007

40



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 una blocco di overflow
 - funzioni hash "alternative"



Tabelle e file hash

Tabelle hash

- l'indirizzo hash identifica esattamente la locazione dove memorizzare il record
- sono usate in memoria primaria
- minimizzano il numero di record acceduti

File hash

- l'indirizzo hash identifica il blocco dove memorizzare il record: nel blocco i record sono disposti usando un altro criterio, ad esempio sequenzialmente
- sono usate in memoria secondaria
- minimizzano il numero di blocchi acceduti

Un esempio di tabella hash

40 record

- **tavola hash** con 50 posizioni divisa in 5 blocchi:

- 5 record in un blocco diverso da quello ricercato

numero medio di accessi: 1,125

chiave chiave mod 50

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

Scarselli Franco

Sistemi per basi di dati 2006-2007

43

File hash

File hash

- 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

File hash con

- 40 record
- fattore di blocco 10
- 1 collisione
 - 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				

Scarselli Franco

Sistemi per basi di dati 2006-2007

44

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

Scarselli Franco

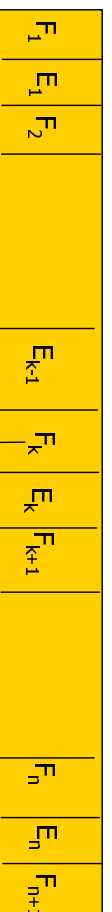
Sistemi per basi di dati 2006-2007

45

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



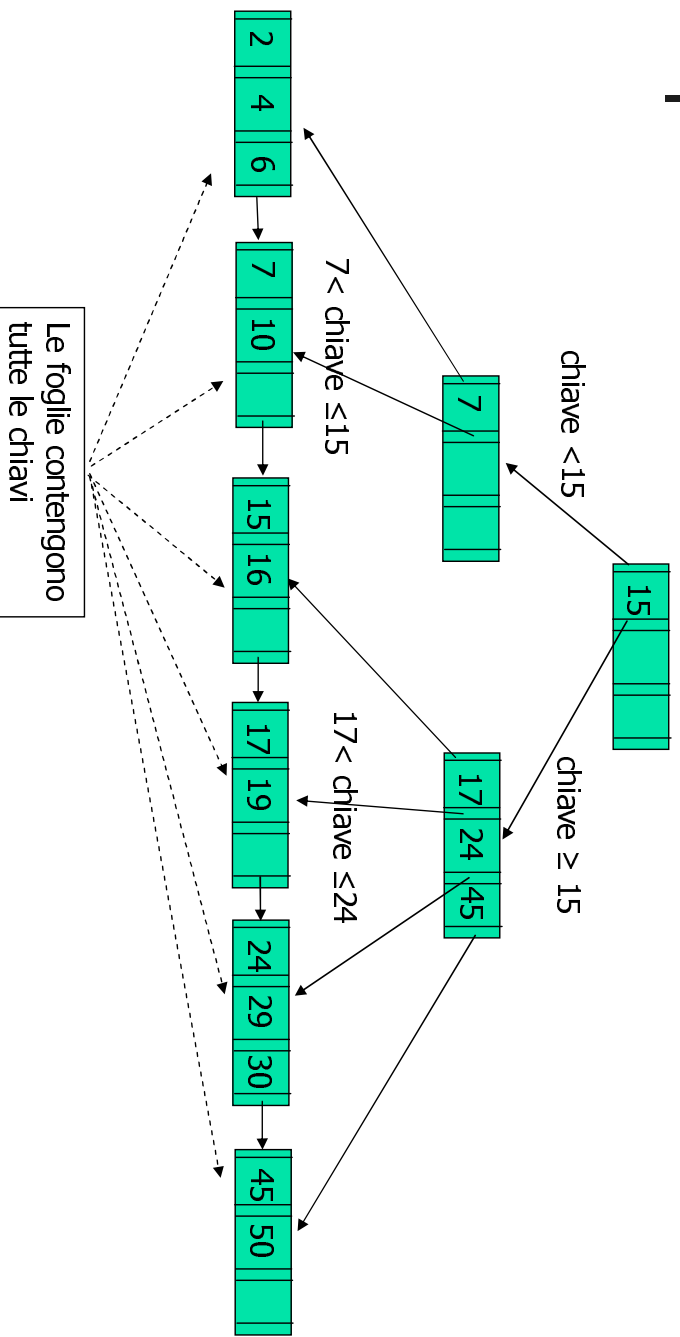
punta alle tuple le cui chiavi c soddisfano $E_{k-1} < c \leq E_k$

Scarselli Franco

Sistemi per basi di dati 2006-2007

46

B+ Tree: un esempio



Scarselli Franco

Sistemi per basi di dati 2006-2007

47

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 2006-2007

48



B-tree e fan out

Qual è il fan out ottimo per un B-tree ?

- all'aumentare del fan out diminuiscono i livelli dell'albero
 - minore numero di nodi da scandire per accedere a un dato
- se il fan out è troppo grande, la memorizzazione di un nodo può richiedere più blocchi
 - numero di accessi alla memoria secondaria maggiore
- **La soluzione migliore si ha scegliendo il fan out maggiore possibile fra quelli per i quali un nodo è memorizzabile in un blocco**

Scarselli Franco

Sistemi per basi di dati 2006-2007

49



Efficienza dei B-Tree

Quanto sono efficienti i B-Tree ?

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

Esempio

- Blocchi 4KB, chiavi intere 4B, puntatori 8B
- Si può 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 può indicizzare 68G**

Scarselli Franco

Sistemi per basi di dati 2006-2007

50

B+ Tree e B tree

B+ tree

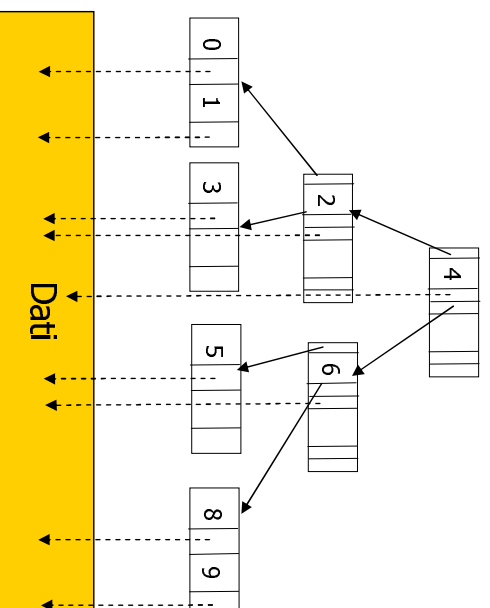
- le foglie contengono tutte le chiavi
 - alcune chiavi sono ripetute nei nodi interni
- le foglie sono collegate in una lista
- le foglie puntano (o contengono i dati)
- ottimi per le ricerche su intervalli
- molto usati nei DBMS



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





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)
```

Scarselli Franco

Sistemi per basi di dati 2006-2007

55



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

Scarselli Franco

Sistemi per basi di dati 2006-2007

56

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'
```

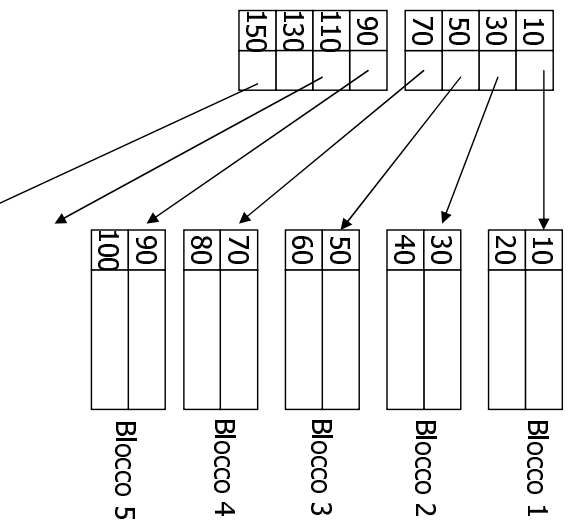
Scarselli Franco

Sistemi per basi di dati 2006-2007

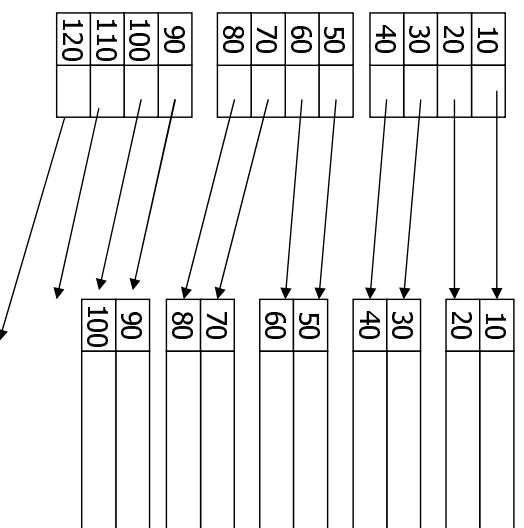
57

Un esempio: indici primari

Indice sparso



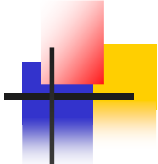
Indice denso



Scarselli Franco

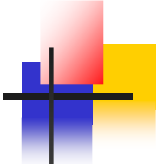
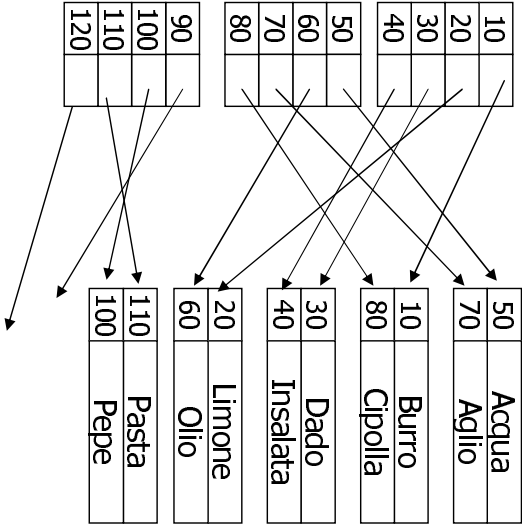
Sistemi per basi di dati 2006-2007

58



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 possono richiedere una riorganizzazione dell'indice
 - tecniche per alleviare i problemi:
 - file o blocchi di overflow
 - marcatura per le eliminazioni
 - riempimento parziale
 - blocchi collegati (non contigui)
 - riorganizzazioni periodiche

Scarselli Franco

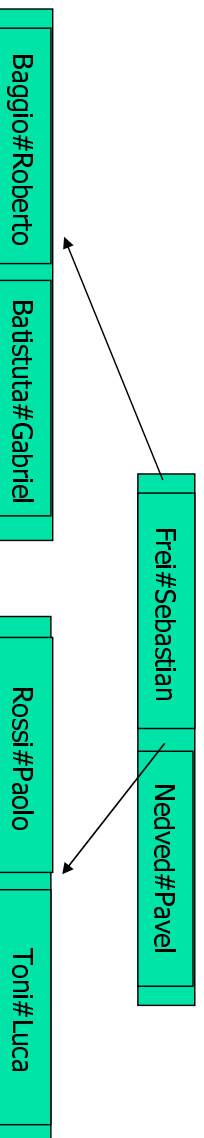
Sistemi per basi di dati 2006-2007

61

Multi-indici e strutture dati per applicazioni particolari

- Gli indici su più campi possono essere facilmente realizzati con alberi ordinati rispetto alla concatenazione

```
CREATE INDEX indice1 ON studenti(cognome, nome)
```



- Anche file hash con chiavi composte possono essere costruiti nello stesso modo
- In alcuni casi, però, conviene usare strutture dati multidimensionali ad hoc per il problema considerato

Scarselli Franco

Sistemi per basi di dati 2006-2007

62

Esempio: i 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	abbaiaare	abbaiano	...	documento	...	parla	...	linux	...
False	False		...	True	...	True	...	True	...

L'uso di indici tradizionali in questo caso non è indicato perché

- Ogni campo accetta solo due valori: un indice su un campo si ridurrebbe ad un albero con due soli rami
- C'è un numero grande e variabile di campi

Esempio: i documenti di testo II

Intuitivamente

- 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)

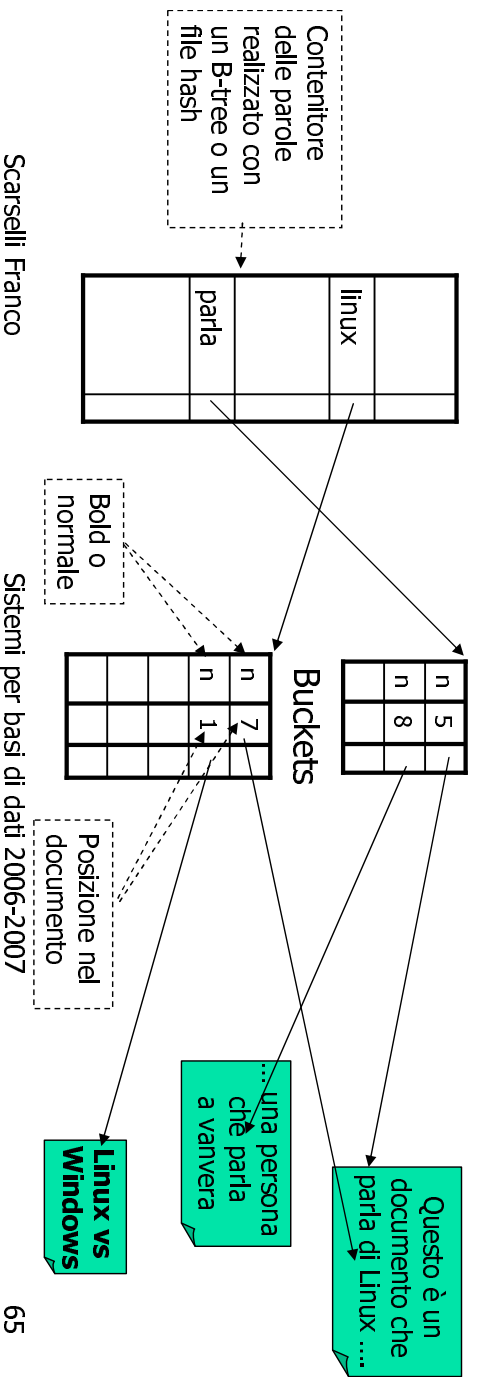
La struttura dell'indice

- l'indice contiene tutte le parole presenti nell'archivio e può essere organizzato usando una tecnica tradizionale (B-tree, file hash)
- l'indice punta a dei **bucket** che contengono una lista delle occorrenze di ciascuna parola eventualmente con l'aggiunta di altre informazioni sull'occorrenza

Esempio: i documenti di testo III

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, ..)



65

Altre applicazioni

Geographic Information System (GIS)

- Memorizzano dati rappresentabili in due o tre dimensioni
- Servono a gestire mappe geografiche, ma anche disegni di circuiti, architettura,
- Permettono di registrare punti, rettangoli, cerchi e più in generale ponti, strade, etc

Data Cube

- Memorizzano informazione multi-dimensionale per applicazioni di supporto alle decisioni: ad esempio, una catena potrebbe memorizzare per ogni acquisto data, prezzo, negozio, oggetto acquistato, guadagno ...
- Permettono di visualizzare i dati come cubi dove ogni dimensione corrisponde ad un attributo

Altre applicazioni II

- GIS e Data Cube implementano operazioni che possono essere rese più efficienti utilizzando indici ad hoc
- **Aggregazioni**
 - “Quante magliette azzurre sono state vendute in nei negozi toscani nel primo trimestre del 2005 ?”
- **Interrogazioni con match parziale**
 - “Quali sono le università a nord di Siena?”
 - “Quali sono le università ad est di Siena?”
- **Interrogazioni su intervalli**
 - “Quali sono le città fra il meridiano xx e quello yy?”
 - “Quali sono i fiumi fra il meridiano xx e quello yy?”
- **Ricerca dei più vicini**
 - “Qual è l'autostrada più vicina a Siena?”
 - “Qual è la regione non confinante più vicina alla toscana?”
- **Interrogazioni “dove mi trovo?”**
 - “In quale contrada mi trovo?”

Scarselli Franco

Sistemi per basi di dati 2006-2007

67

Strutture dati per dati multi-dimensionali hash-like

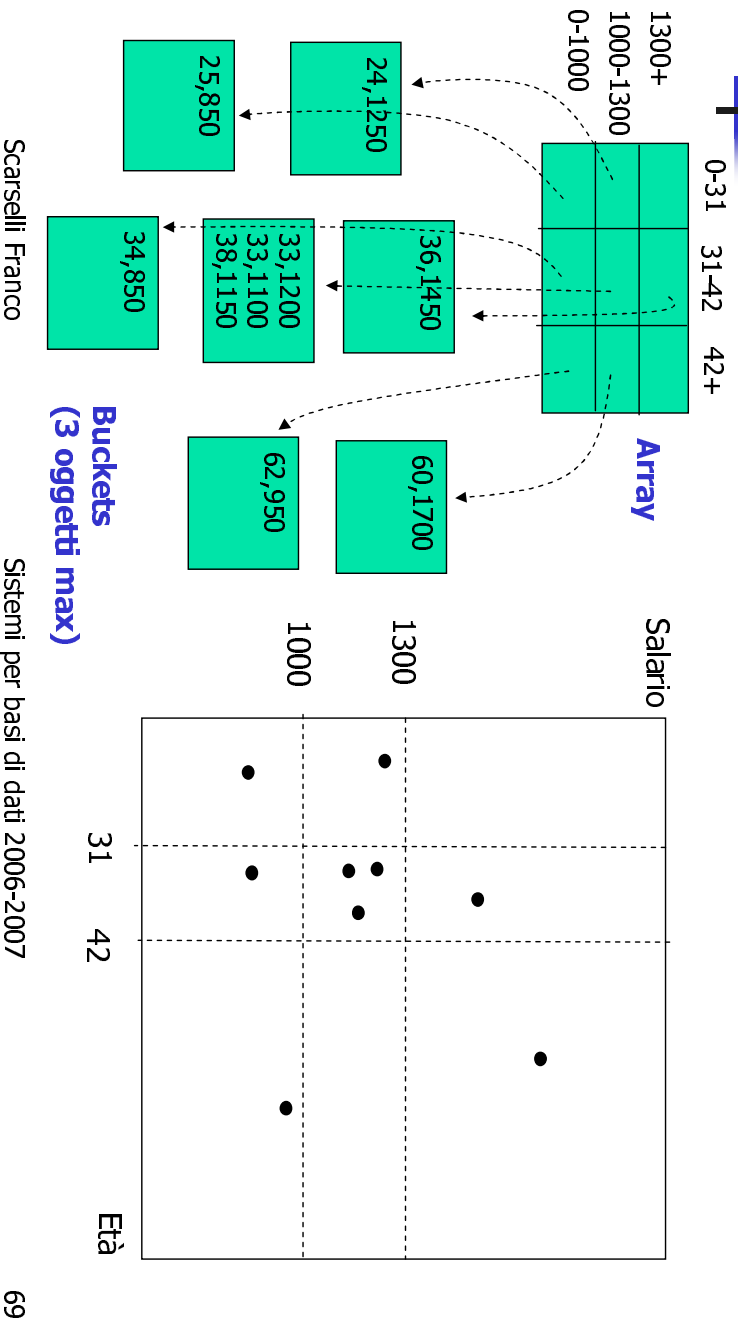
- Si partiziona lo spazio in regioni.
- Per ogni regione si realizza un bucket dove si inseriscono tutti gli oggetti in essa contenuta.
- L'indirizzo del bucket viene calcolato applicando una funzione alla chiave.
- Esempi di tecniche di questo tipo:
 - **Grid Files**
 - Le regioni sono definite dal linee orizzontali e linee verticali
 - Una matrice contiene i puntatori ai bucket
 - **Partitioned Hashing**
 - si applica una funzione hash h ad ogni dimensione v_i
 - la chiave hash globale $h(v_1)..h(v_n)$ che definisce quale bucket usare è data dalla concatenazione delle singole chiavi hash $h(v_i)$

Scarselli Franco

Sistemi per basi di dati 2006-2007

68

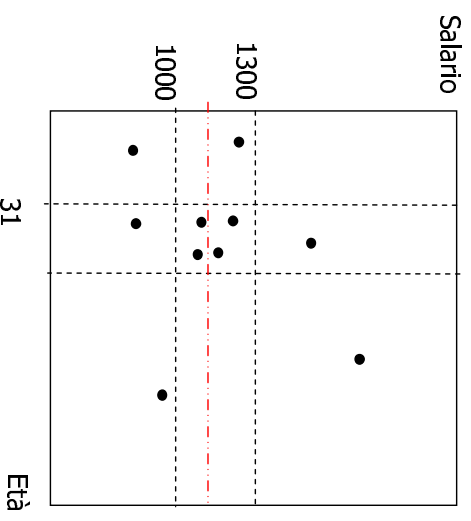
Grid Files



69

Grid Files

- Inserimenti e cancellazioni
 - possono richiedere la divisione o la fusione delle regioni e dei bucket
- Interrogazioni su intervalli e "dove mi trovo?"
 - vengono eseguite considerando le regioni coinvolte
- Ricerca dei più vicini ad un oggetto *o*
 - Si fissa un valore *d* e si cercano in tutte le regioni che possono contenere oggetti più vicini di *d* da *o*
- Se non si trova nessun oggetto si riesegue l'interrogazione con un valore più grand di *d*



Strutture dati per dati multi-dimensionali basate su alberi

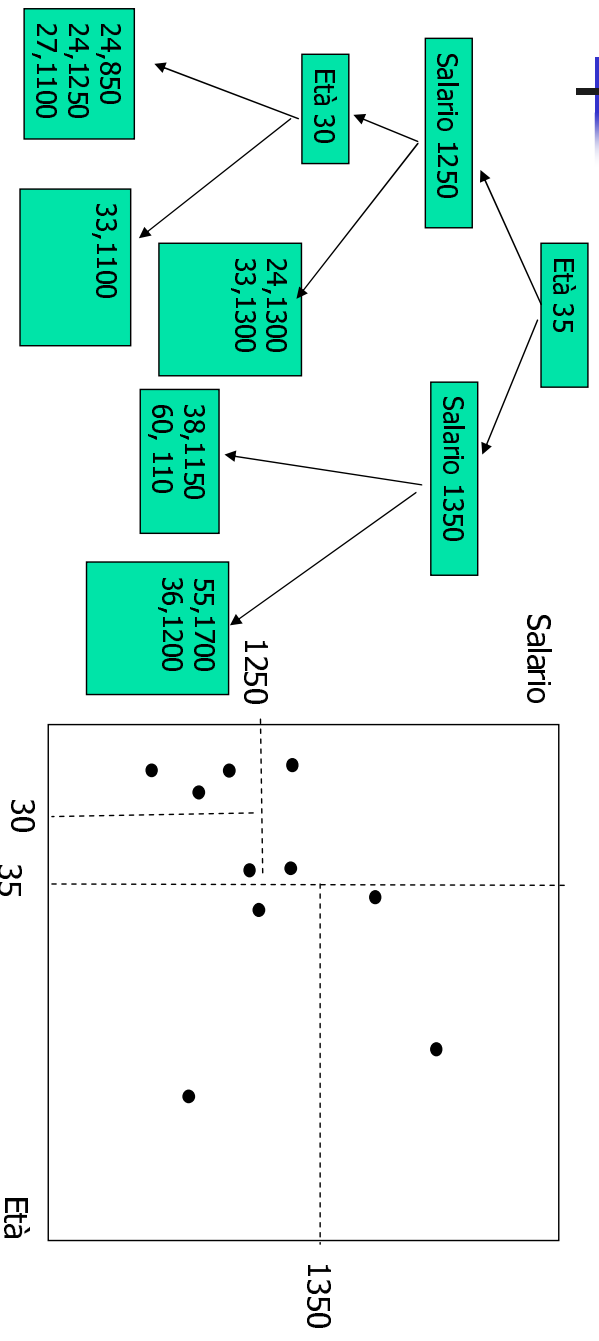
- Si partiziona lo spazio in regioni e sottoregioni
- Si costruisce un albero in cui ogni nodo rappresenta una regione
- La radice rappresenta la regione di tutto lo spazio ammissibile, mentre i figli di ciascun un nodo rappresentano sottoregioni di quella del nodo
- Le foglie dell'albero sono i bucket in cui sono contenuti gli oggetti
- Esempi di tecniche di questo tipo:
 - **KD-Tree**
 - In ogni nodo la regione viene divisa rispetto ad un valore **a** di un attributo: in un ramo gli oggetti maggiori di **a**, nell'altro quelli minori
 - L'attributo scelto è lo stesso per tutti i nodi di un livello
 - **Quad-Tree**
 - Ogni regione viene suddivisa in quattro sottoregioni uguali corrispondenti ai quattro quadranti
 - **R-Tree**
 - Le regioni hanno tutte forma predefinita: ad. es. un rettangolo
 - Le sottoregioni sono scelte possono avere delle intersezioni e non coprire tutta la regione padre

Scarselli Franco

Sistemi per basi di dati 2006-2007

71

KD-Tree

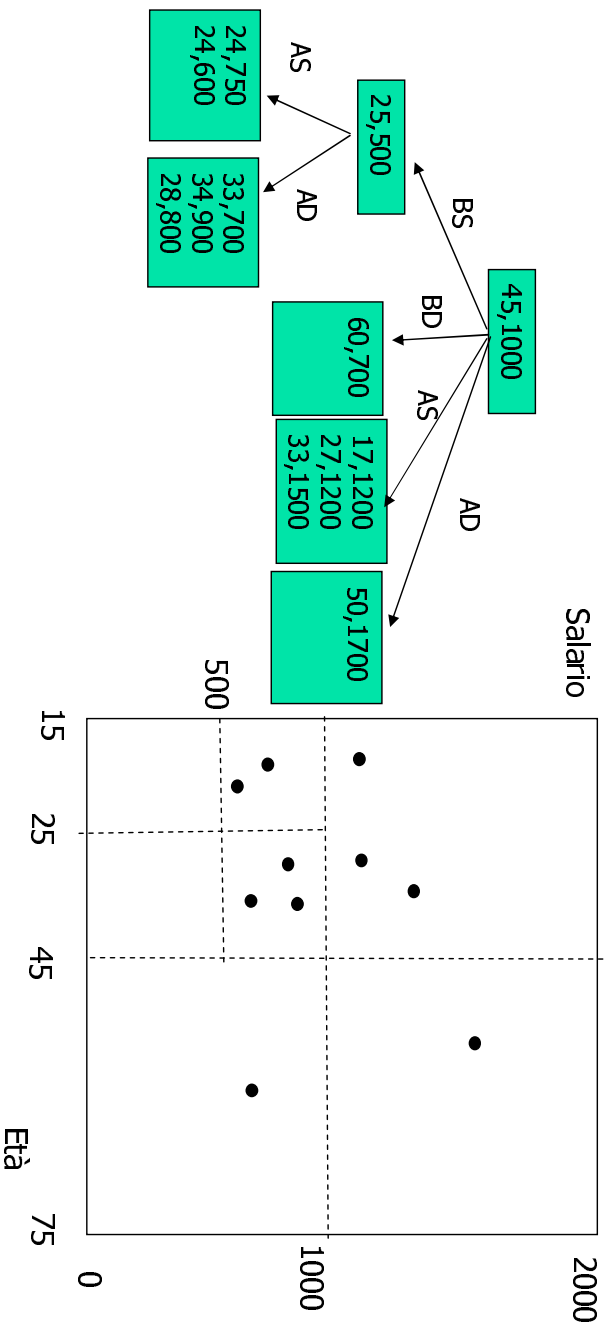


Scarselli Franco

Sistemi per basi di dati 2006-2007

72

Quad-Tree

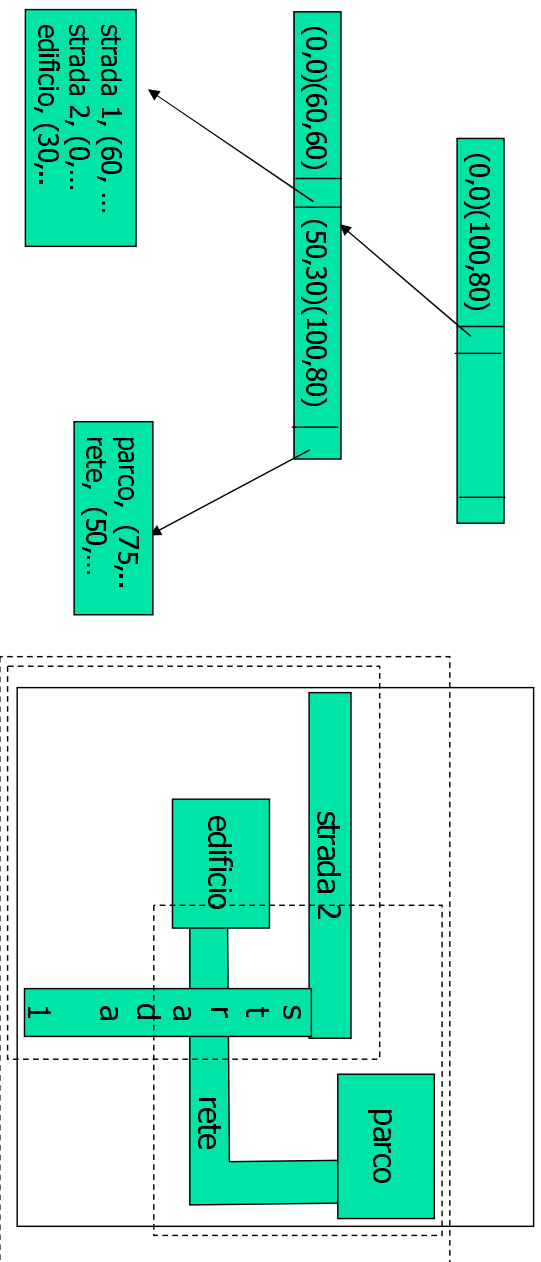


Scarselli Franco

Sistemi per basi di dati 2006-2007

73

R-Tree



Scarselli Franco

Sistemi per basi di dati 2006-2007

74



Strutture dati per dati multi-dimensionali basate su bitmap

- Per ogni attributo si considerano tutti i valori v_{ij}, \dots, v_m che compaiono nell'archivio
- Un **indice bitmap** su un attributo consiste in un vettore di m bit per ciascun record
- Ogni vettore è costituito da tutti 0 e un solo 1 in corrispondenza del valore del record
- Un indice multi-dimensionale si ottiene concatenando gli indici bitmap degli attributi coinvolti
- Con gli indici bitmap si implementano facilmente attraverso operazioni di OR e AND su bit, le operazioni di intervallo incluse in un'interrogazione

Scarselli Franco

Sistemi per basi di dati 2006-2007

75



Indici bitmap: un esempio

Esempio

- Dati i record (Età, salario): (30,1000),(30,1300), (40,1000), (50,1300)
- con 100 si rappresenta 30, con 010 si rappresenta 40, con 001 si rappresenta 40
- con 10 si rappresenta 1000, con 01 si rappresenta 1300

Indice bitmap su Età	Indice bitmap su salario	Indice bitmap su entrambi
100	10	10010
100	01	10001
010	10	01010
001	01	00101

.... per cercare gli impiegati di età compresa fra 30 e 40, si cercano i bitmap dove il primo o il secondo bit sono 1

Scarselli Franco

Sistemi per basi di dati 2006-2007

76



Indici bitmap

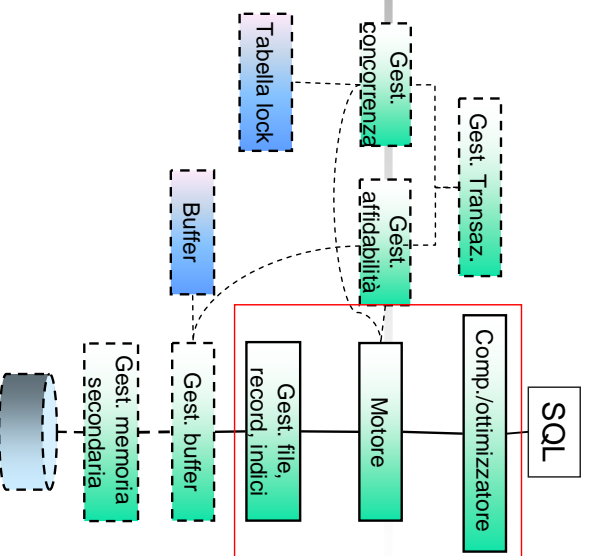
- Per diminuire l'occupazione di memoria gli indici bitmap devono essere compressi
 - Un metodo possibile è il seguente
 - Si codificano solo le lunghezze delle sequenze di 0: si suppone che ci sia un 1 fra ogni sequenza
 - Ogni lunghezza è codificata dal valore v in binario preceduto da uno 0 e tanti 1 (meno uno) quanti sono i bit di v
 - Il metodo è efficiente quando ci sono pochissimi 1: ad esempio, quando nella codifica del testo dove un attributo indica al presenza di una parola
- | | |
|----------------------------|-------------------------------|
| Codifica della lunghezza 5 | Codifica della bitmap 0001000 |
| 110 101 | 1011 1011 |
| Codifica della lunghezza 3 | Codifica della bitmap 0100000 |
| 10 11 | 01 110101 |
| Codifica della lunghezza 1 | Codifica della bitmap 0111000 |
| 0 1 | 01 00 00 1011 |
| Codifica della lunghezza 0 | |
| 0 0 | |
- Scarselli Franco
- Sistemi per basi di dati 2006-2007
- 77



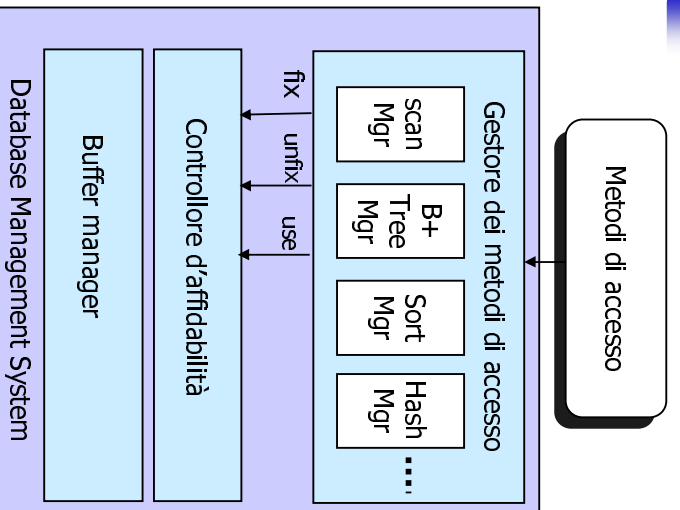
Osservazioni

- Per alcune delle strutture ad albero presentate, esistono delle estensioni ad alberi n-ari: l'obiettivo è quello di memorizzare un nodo esattamente in blocco come per i B-Tree
- In alternativa, in un blocco si memorizzano un nodo e una parte dei suoi discendenti: questo permette di rendere più efficiente la ricerca
- Per una dimostrazione delle strutture dati presentate si veda. ad esempio,
<http://donar.umiacs.umd.edu/quadtrees/>

Esecuzione delle interrogazioni

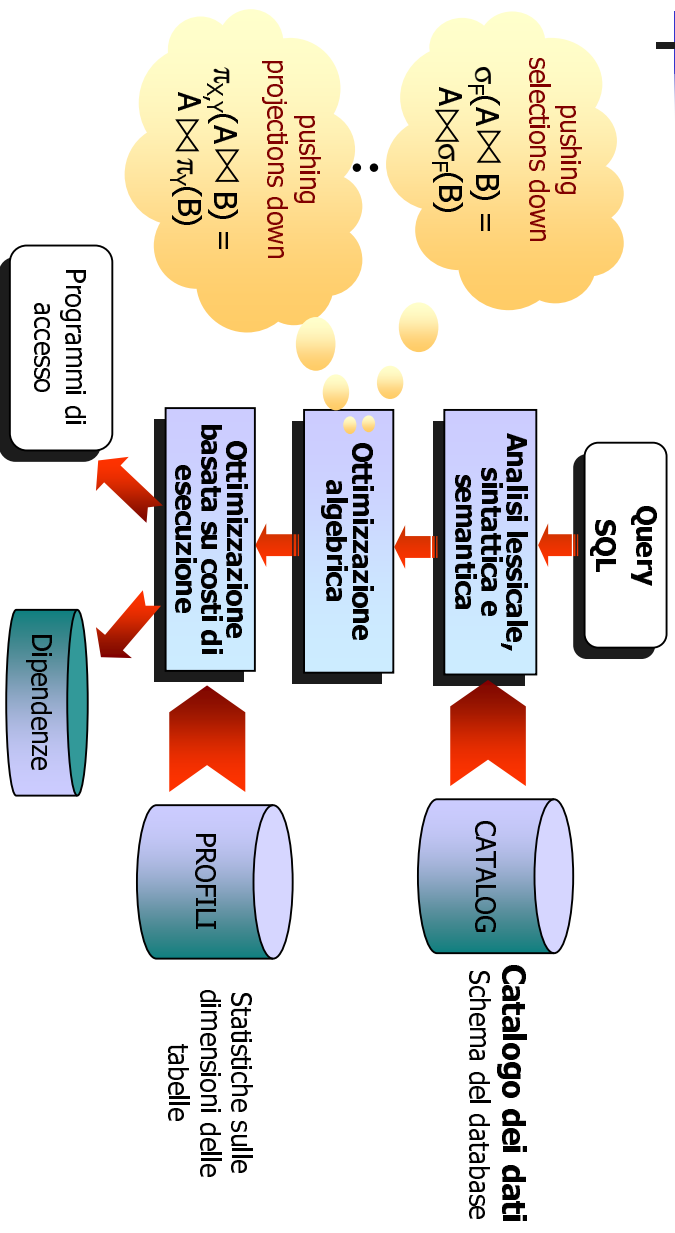


Strutture fisiche di accesso



- Riguardano l'organizzazione dei dati per rendere efficienti le operazioni di ricerca e modifica
- Si possono definire indici
- I metodi di accesso gestiscono una particolare organizzazione fisica
- Il metodo di accesso individua i blocchi fisici che devono essere caricati in memoria dal buffer manager

Ottimizzazione delle interrogazioni



Scarselli Franco

Sistemi per basi di dati 2006-2007

81

Compilazione delle query

- I programmi di accesso sono in formato "oggetto"
- **compile-and-store**
 - l'interrogazione viene compilata una sola volta
 - il codice oggetto viene memorizzato insieme alle dipendenze dalle versioni di tabelle e indici
 - il codice oggetto viene invalidato se la struttura della base di dati cambia significativamente per l'interrogazione (es. aggiunta di un indice)
- **compile-and-go**
 - l'interrogazione viene compilata ed eseguita, ma non è memorizzata

Scarselli Franco

Sistemi per basi di dati 2006-2007

82

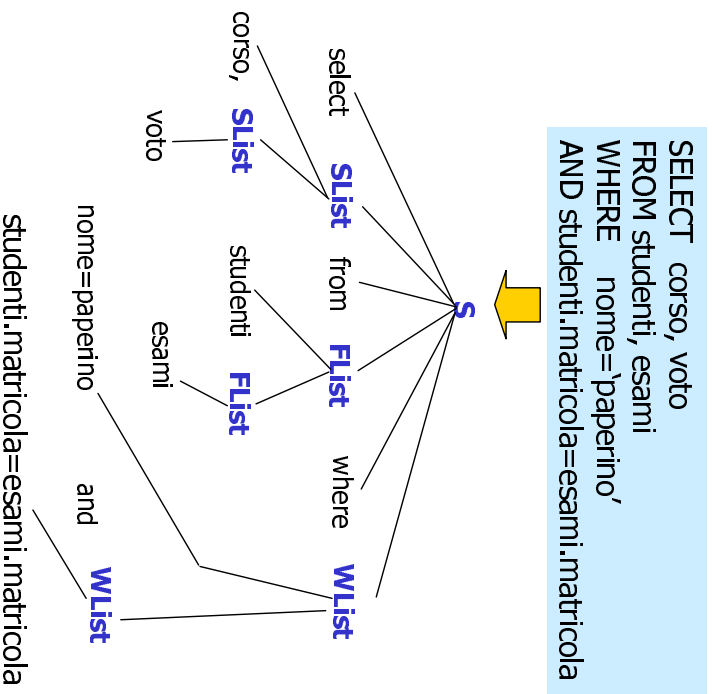
Analisi lessicale, sintattica e semantica

Analisi lessicale e sintattica

- un parser legge la query in SQL e produce il rispettivo albero di analisi
- il parser è costruito a partire dalla grammatica dell'SQL con le tecniche usate anche per gli altri linguaggi

Analisi semantica:

- si controlla l'esistenza delle tabelle e attributi indicati nell'interrogazione
- si associa ad ogni nome il relativo oggetto
- si controlla che i tipi dei dati coinvolti nelle operazioni sia corretto



Scarselli Franco

Sistemi per basi di dati 2006-2007

83

Traduzione dell'interrogazione in algebra relazionale

Traduzione in algebra relazionale

- l'albero sintattico viene tradotto in un altro albero che rappresenta l'interrogazione in algebra relazionale estesa
 - le foglie contengono tabelle
 - i nodi interni operatori relazionali
- operatori dell'algebra
 - unione, intersezione, differenza
 - selezione (ad es. $\sigma_{\text{conome=paperino}}(\text{STUDENTI})$)
 - proiezione (ad es. $\pi_{\text{matricola}}(\text{STUDENTI})$)
 - prodotto e join ($\times, \triangleright \triangleleft$)
 - aggregazione (ad es. $\gamma_{\text{matricola, AVG(voto)} \rightarrow \text{media}}(\text{ESAMI})$)
 - ordinamento (ad es. $\tau_{\text{nome}}(\text{STUDENTI})$)

Scarselli Franco

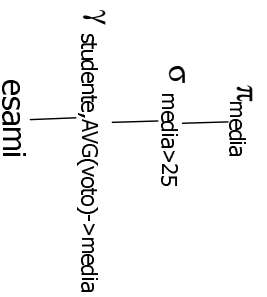
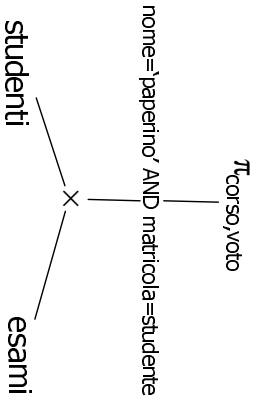
Sistemi per basi di dati 2006-2007

84

Traduzione dell'interrogazione in algebra relazionale II

- nel caso di una interrogazione select-from-where la trasformazione è semplice
- se l'interrogazione contiene operatori di raggruppamento o ordinamento, la traduzione richiede l'uso dell'algebra estesa

```
SELECT corso, voto
FROM studenti, esami
WHERE nome='paperino'
AND matricola=studente
```



```
SELECT AVG(voto) AS media
FROM esami
GROUP BY studente
HAVING media > 25
```

Scarselli Franco

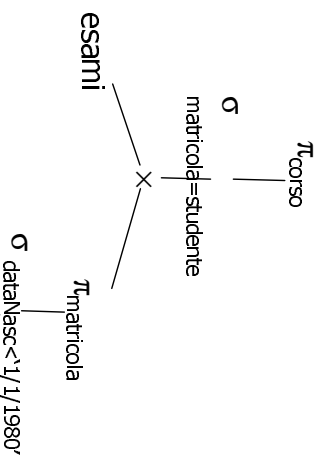
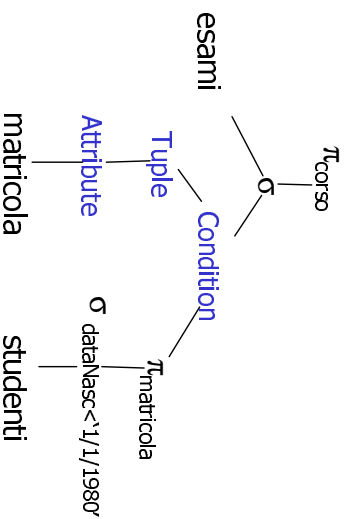
Sistemi per basi di dati 2006-2007

85

Traduzione dell'interrogazione in algebra relazionale III

- nel caso di una interrogazione che contenga una sottointerrogazione
 - prima, si trasforma parzialmente l'interrogazione in un albero con un nodo speciale che rappresenta la sottointerrogazione
 - poi, si tenta di riscrivere la sottointerrogazione in una espressione equivalente

```
SELECT corso FROM esami
WHERE studente IN (SELECT matricola FROM studenti WHERE dataNasc < '1/1/1980')
```



Scarselli Franco

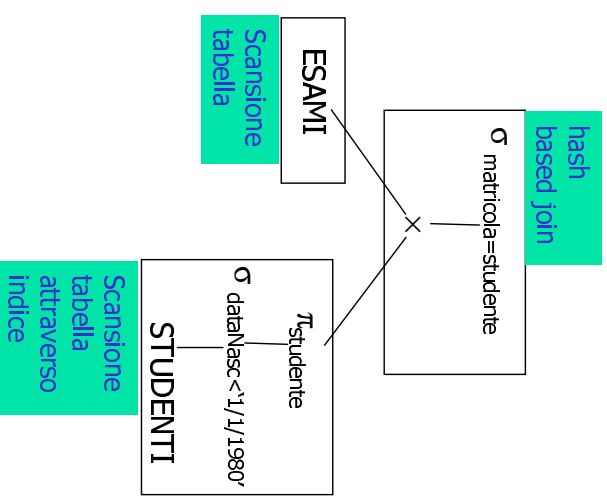
Sistemi per basi di dati 2006-2007

86

Query plan

Query plan

- è una **rappresentazione interna** della query
- **query plan logico**
 - è un albero rappresentante un'espressione relazionale
 - suggerisce le operazioni logiche da eseguire e l'ordine in cui devono essere eseguite
- **query plan fisico**
 - è un albero indicante le operazioni fisiche necessarie ad implementare l'interrogazione
 - ogni nodo implementa un insieme di operazioni descritte dal query plan logico



Scarselli Franco

Sistemi per basi di dati 2006-2007

87

Query plan II

La trasformazione da query plan logico a query plan fisico

- **ottimizzare il query plan logico** (ottimizzazione algebrica)
 - si cercano, fra quelle algebricamente equivalenti, le espressioni piu' efficienti
- **implementare gli operatori dell'algebra relazione**
 - trasformare gli operatori in operazioni da richiedere al gestore dei metodi d'accesso
- **ottimizzare il query plan fisico** (ottimizzazione basata sui costi di esecuzione previsti)
 - scegliere fra le possibili implementazioni quella piu' efficiente sulla base dello stato attuale del database

Scarselli Franco

Sistemi per basi di dati 2006-2007

88

Ottimizzazione algebrica

Il primo passo dell'ottimizzazione

- si riscrive l'espressione relazionale in nuove espressioni
 - equivalenti
 - piu' efficienti
- in questa fase **non si usa informazione** sullo stato attuale del database
- la riscrittura si basa su formule di equivalenza

Idee guida

- minimizzare i numero di record nelle relazioni intermedie
- minimizzare la dimensione delle relazioni intermedie

Scarselli Franco

Sistemi per basi di dati 2006-2007

89

Ottimizzazione algebrica II

Pushing selections

- consiste nel spingere il piu' possibile **verso le foglie** (anticipare) le selezioni
- minimizza il numero di record trattati
- alcune equivalenze:
 - $\sigma_c(R-S) = \sigma_c(R) - \sigma_c(S)$ (puo' convenire piu' di $\sigma_c(R-S) = \sigma_c(R) - S$)
 - $\sigma_c(R \bowtie_b S) = \sigma_c(R) \bowtie_b \sigma_c(S)$
 - $\sigma_c(R \bowtie_b S) = \sigma_c(R) \bowtie_b S$ (se C non contiene attributi di S)
 -

$\pi_{\text{corso,voto}}(\sigma_{\text{nome='paperino'}}(\text{ESAMI} \bowtie_{\text{matricola=studente}} \text{STUDENTI}))$



$\pi_{\text{corso,voto}}(\text{ESAMI} \bowtie_{\text{matricola=studente}} (\sigma_{\text{nome='paperino'}}(\text{STUDENTI})))$

Scarselli Franco

Sistemi per basi di dati 2006-2007

90

Ottimizzazione algebrica III

Pushing projections

- consiste nel spingere il piu' possibile verso le foglie (anticipare) le proiezioni
- piu' in generale le proiezioni possono essere aggiunte ovunque, purché tolgano solo **attributi che non verranno piu' usati**
- alcune equivalenze
 - $\pi_L(\sigma_C(R)) = \pi_L(\sigma_C(\pi_M(R)))$
dove M sono gli attributi di R che compaiono in C o in L
 - $\pi_L(R \bowtie_D S) = \sigma_C(\pi_M(R)) \bowtie_D \sigma_C(\pi_N(S))$
dove M (N) sono gli attributi di R (S) che compaiono in D o in L

$\pi_{\text{corso, voto}}(\sigma_{\text{nome='paperino'}}(\text{ESAMI} \bowtie_{\text{matricola=studente}} \text{STUDENTI}))$



$\pi_{\text{corso, voto}}(\text{ESAMI} \bowtie_{\text{matricola=studente}} (\sigma_{\text{nome='paperino'}} (\pi_{\text{nome, matricola}}(\text{STUDENTI}))))$

Scarselli Franco

Sistemi per basi di dati 2006-2007

91

Realizzazione delle operazioni fisiche

Le operazioni implementate in RDBMS tipici sono

- **scansioni sequenziali**
- **join**
- **ordinamenti**
- **accessi tramite indice**

Un'altra classificazione

- metodi **one-pass** che richiedono **una sola lettura** dei dati
 - di solito funzionano se uno degli operandi sta tutto in memoria
- metodi **two-pass** che richiedono **due letture** dei dati
 - ad esempio, il two phase multiway merge-sort
- metodi che richiedono **piu' letture** dei dati

Scarselli Franco

Sistemi per basi di dati 2006-2007

92



Scansioni (scan)

- Si accede in sequenza alle tuple
 - open - si apre la scansione
 - next - si avanza il puntatore alla tupla successiva
 - read/modify/delete - legge/modifica/cancella la tupla corrente
 - insert - inserisce una nuova tupla nella posizione corrente
 - close - chiude la scansione
- Durante lo scan si possono applicare delle operazioni
 - proiezione, selezione su un predicato semplice, ordinamento

Scarselli Franco

Sistemi per basi di dati 2006-2007

93



Accesso tramite indici (index scan)

- Gli indici sono spesso realizzati con alberi
- Favoriscono l'accesso associativo per interrogazioni che comprendono predicati tipo $A_i = v$ oppure $v_1 \leq A_i \leq v_2$ (predicati valutabili con l'indice)
- $C_1 \wedge C_2$ dove entrambe le condizioni sono valutabili con indici (riguardano le chiavi), si sceglie la più selettiva fra le due per l'accesso tramite indice. Il secondo predicato viene valutato sulle pagine caricate nel buffer. Se gli indici sono densi si possono valutare entrambe le condizioni sugli indici e poi unire i risultati.
- $C_1 \vee C_2$ se una delle due non è valutabile con indice, occorre fare una scansione completa
- $C_1 \vee C_2$ se sono entrambe valutabili si possono usare gli indici ma non è detto che sia conveniente (se c'è molta sovrapposizione)

Scarselli Franco

Sistemi per basi di dati 2006-2007

94

Accesso tramite indici II

Definiamo

- $B(R)$ – numero di blocchi che costituiscono la relazione R
- $T(R)$ – numero di record che costituiscono la relazione R
- $V(R,A)$ – numero di valori diversi per l'attributo A di R
- M dimensione in blocchi della memoria disponibile

Costo di realizzazione della selezione $\sigma_{A=v}(R)$

- **clustered index su $R.A$ (tipico per indici primari)**
se i record che hanno valore uguale per A sono memorizzati il più vicino possibile nell'indice
 - circa $B(R)/V(R,A)$
- **nonclustered index su $R.A$ (tipico indici secondari densi)**
se i record che hanno valore uguale sono memorizzati in blocchi diversi
 - circa $T(R)/V(R,A)$
- **senza indice**
 - circa $B(R)$

Scarselli Franco

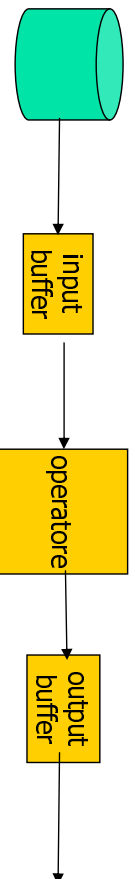
Sistemi per basi di dati 2006-2007

95

Metodi one-pass: esempi

Selezione, proiezione e altri operatori unari

- funzionamento
 - si leggono i dati dalla memoria secondaria nel buffer
 - si applica l'operatore
 - si scrive il risultato nel buffer di output
- costo (senza la scrittura del risultato su disco!)
 - se i dati sono **clustered**, B (numero dei blocchi) accessi alla memoria secondaria
 - se i dati sono **nonclustered**, T (numero di record) accessi alla memoria secondaria
 - funziona solo se $B < M$



Scarselli Franco

Sistemi per basi di dati 2006-2007

96

Metodi one-pass: esempi II

Join

- funzionamento
 - si carica la prima relazione R dalla memoria secondaria alla memoria primaria (ad. es. tabella hash)
 - si costruisce una struttura di accesso ad R
 - si legge la seconda relazione S un blocco alla volta
 - per ogni record di S si controlla se esiste uno o più record in R che soddisfano il join e si scrive il record risultato in un buffer di uscita
- costo
 - se i dati sono **clustered** $B(R)+B(S)$
 - funziona solo se $B(R) < M$

Scarselli Franco

Sistemi per basi di dati 2006-2007

97

Metodi two pass

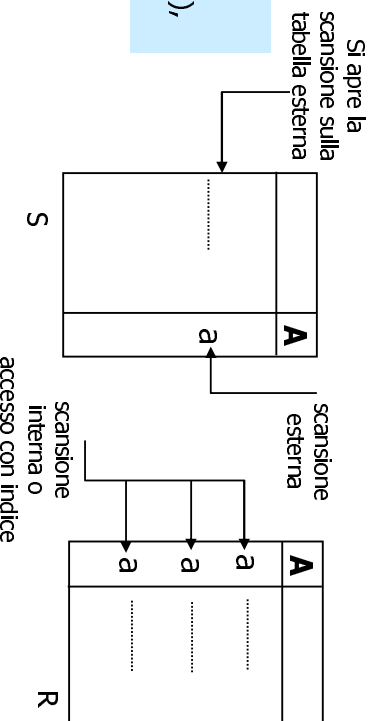
Cosa succede se le relazioni non entrano in memoria ?

Nested-loop Join

- si sceglie la relazione con un **numero minore di record (S)**
- si scandisce S e per ogni record di S si lancia una scansione su R

Servono $B(S)+T(S)*B(R)$ letture

```
FOR EACH tuple s in S DO
  FOR EACH tuple r in R DO
    IF (r and s soddisfano la condizione di join),
      THEN costruisci la tupla di unione
```



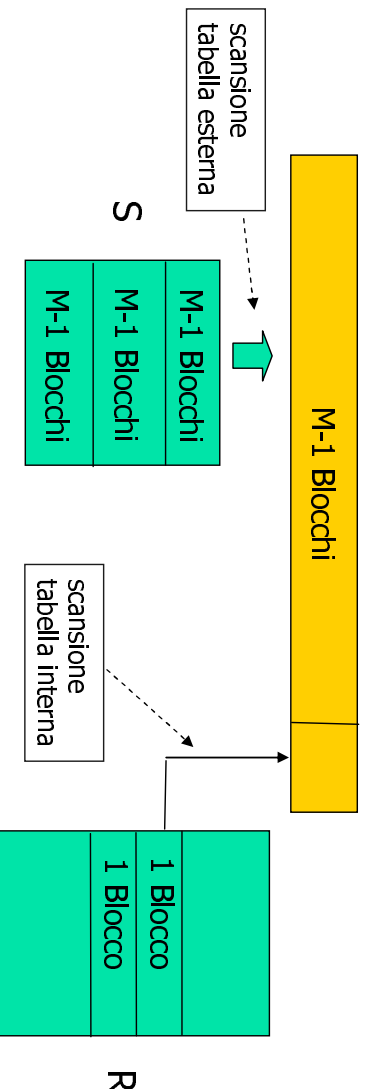
Scarselli Franco

Sistemi per basi di dati 2006-2007

98

Nested-loop join

- un miglioramento si ottiene di usare un indice per R
 - costo con R e S clustered: circa $B(S) + T(S) * (B(R) / N(R, A))$
- oppure caricando nei buffer M-1 record di S e confrontandoli tutti con i record di R recuperati dalla seconda scansione
 - costo con R e S clustered: circa $(B(S) / (M-1)) * ((M-1) + B(R))$
 - **Se non esiste un indice, si usa sempre questa strategia che è sempre migliore di quella della slide precedente!**



Scarselli Franco

Sistemi per basi di dati 2006-2007

99

Merge scan join

Funzionamento

- si ordinano le due relazioni R e S
- si fondono le due relazioni scandendole contemporaneamente

Osservazioni

- il costo di questo metodo è $5 (B(R) + B(S))$
 - $4 (B(R) + B(S))$ è il costo dell'ordinamento, $(B(R) + B(S))$ il costo del merge
- un miglioramento si ottiene fondendo la seconda fase dell'ordinamento con il "merge" $3(B(R) + B(S))$
 - **questa soluzione è sempre preferibile alla precedente**
- se sono definiti degli indici, l'ordinamento non è più necessario e il costo diviene $(B(R) + B(S))$
- nested-loop join è più efficiente solo se una delle due relazioni è particolarmente piccola
- L'algoritmo richiede $M^2 > (\max(B(R), B(S)))$

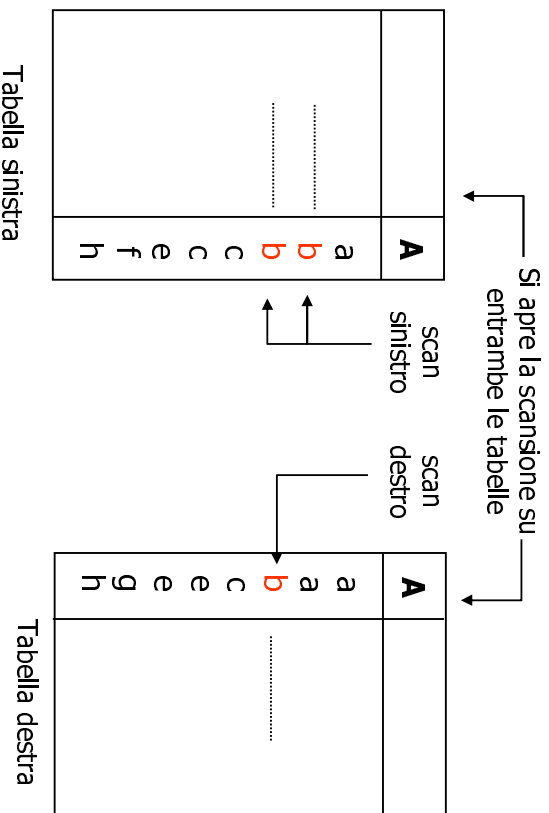
Scarselli Franco

Sistemi per basi di dati 2006-2007

100

Merge scan join II

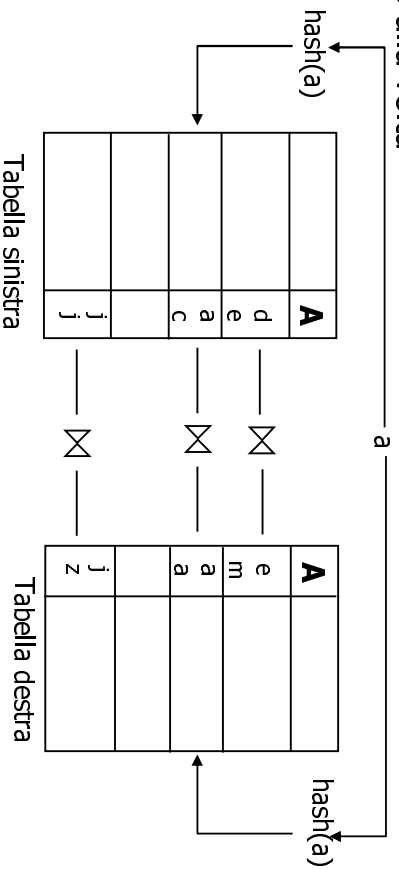
- Le tabelle devono essere ordinate in base agli attributi di join



Hash based join

Funzionamento

- le relazioni R e S sono **riorganizzate** attraverso una **funzione hash** che indica in quale bucket (fra M-1) ogni record deve essere memorizzato
 - in questo modo record con lo **stesso campo** finiscono nello **stesso bucket**
- si carica un bucket di S e il corrispondente di R e si uniscono i record
 - la memoria deve contenere uno dei bucket, l'altro può essere caricato un blocco alla volta



Hash based join

Osservazioni

- il costo di questo metodo è $3 (B(R) + B(S))$
 - $2 (B(R) + B(S))$ è il costo della prima fase
 - $(B(R) + B(S))$ il costo della seconda fase
- si richiede che la memoria contenga i bucket di S
 - implica $M^2 > \min(B(R), B(S))$
- l'hash based join può essere leggermente più veloce del merge scan join se non sono presenti indici
- l'hash based join può trattare relazioni leggermente più grandi
- il merge scan join ha il vantaggio di lasciare ordinati i dati

Scarselli Franco

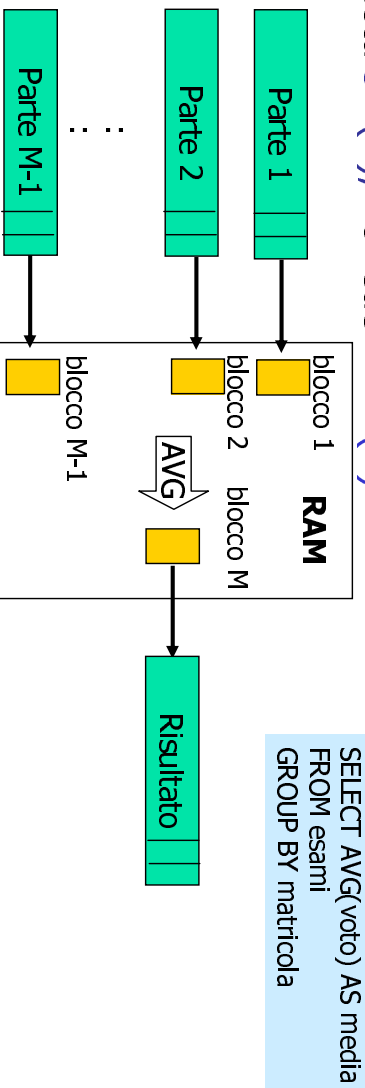
Sistemi per basi di dati 2006-2007

103

Altri esempi di algoritmi two pass

Raggruppamento basato su ordinamento

- si suddivide la relazione R in M-1 sottoparti R_1, \dots, R_{M-1}
- si ordinano R_1, \dots, R_{M-1} in base alla chiave del group by
- si caricano R_1, \dots, R_{M-1} in M-1 blocchi
- si scelgono i record con la stessa chiave valutando contemporaneamente l'espressione
- Costa $3 B(R)$, richiede $M^2 > B(R)$



Scarselli Franco

Sistemi per basi di dati 2006-2007

104

Altri esempi di algoritmi two pass II

Raggruppamento basato su Hashing

- si suddivide la relazione R in $M-1$ buckets usando la funzione hash sulle chiavi di raggruppamento
- ogni bucket viene caricato in memoria, si scelgono i record con la stessa chiave valutando contemporaneamente l'espressione
- occorrono $3B(R)$ accessi alla memoria secondaria
- deve essere verificato $M^2 > B(R)$

Algoritmi che richiedono piu' di tre passi

Algoritmi Multipass

- servono ad elaborare relazioni troppo grandi per essere elaborate in due passi
- esistono estensioni degli algoritmi basati sull'ordinamento e di quello basati su hashing

Basati su ordinamento

- i dati vengono **ricorsivamente** divisi in $M-1$ parti R_1, \dots, R_n uguali ($n > M$) fino quando la memoria non li contiene
- le parti R_1, \dots, R_n sono riunite insieme valutando contemporaneamente la funzione voluta

Basati su hashing

- i dati vengono **ricorsivamente** divisi in $M-1$ bucket R_1, \dots, R_n ($n > M$) con una funzione di hash fino quando la memoria non li contiene
- la funzione voluta viene calcolato separatamente su tutti i bucket R_1, \dots, R_n o coppie di bucket

Ottimizzazione basata sui costi di esecuzione

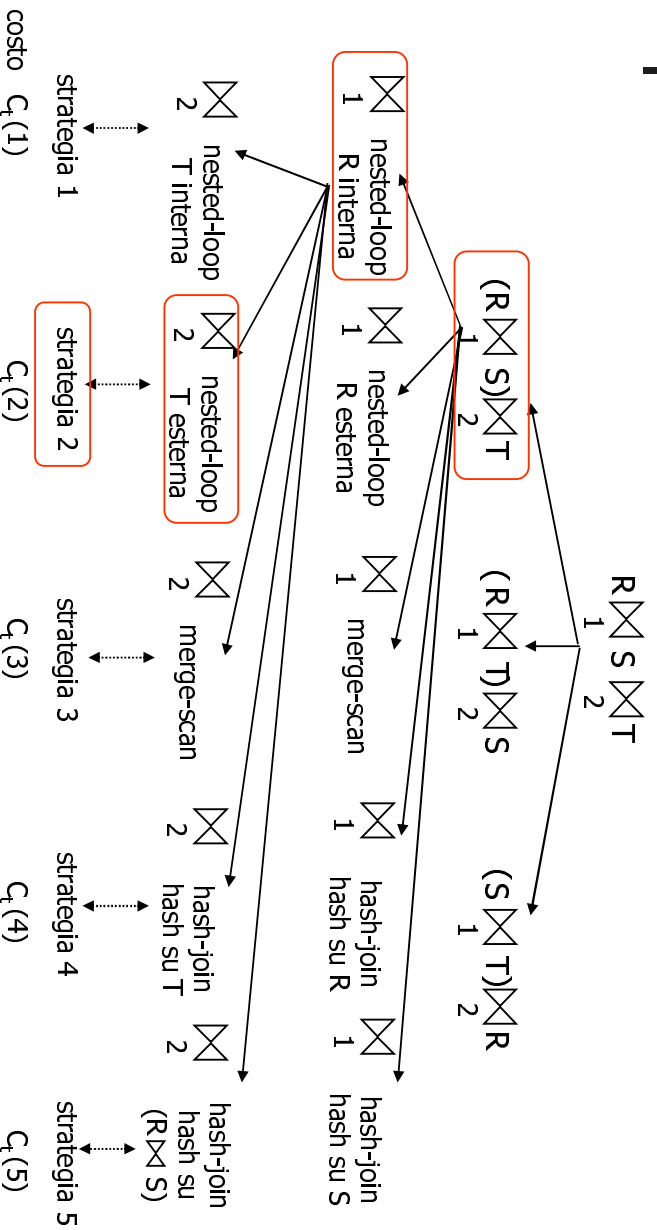
- Le dimensioni per l'ottimizzazione sono molte
 - tipologia dell'operazione di accesso ai dati (es. scan/indice)
 - ordine delle operazioni (es. ordine dei join)
 - modalità di realizzazione di una operazione (es. modalità di join)
 - livello in cui eseguire gli ordinamenti
- Si costruisce un albero delle alternative
 - ogni nodo corrisponde ad una scelta di un'opzione
 - ogni foglia rappresenta una strategia di esecuzione (descritta dal cammino dalla radice alla foglia)
 - Si dovrebbe scegliere il percorso radice-foglia con il costo minore

Scarselli Franco

Sistemi per basi di dati 2006-2007

107

Esempio



Scarselli Franco

Sistemi per basi di dati 2006-2007

108

Profili delle relazioni

- Informazioni quantitative relative alle caratteristiche delle tabelle
 - cardinalità della tabella T - $CARD(T)$
 - dimensione in byte delle tuple di T - $SIZE(T)$
 - dimensione in byte di ciascun attributo - $SIZE(A_j, T)$
 - numero di valori distinti di ciascun attributo - $VAL(A_j, T)$
 - i valori minimo e massimo di ciascun attributo - $MIN(A_j, T)$ $MAX(A_j, T)$
- I profili sono costruiti periodicamente per le tabelle del database
- I profili permettono di fare delle stime sulle dimensioni dei risultati intermedi delle interrogazioni
- Poiché il risultato di interrogazione è essa stessa una tabella temporanea, stimarne le dimensioni vuol dire calcolarne il profilo

Scarselli Franco

Sistemi per basi di dati 2006-2007

109

Profili per le selezioni: un'implementazione

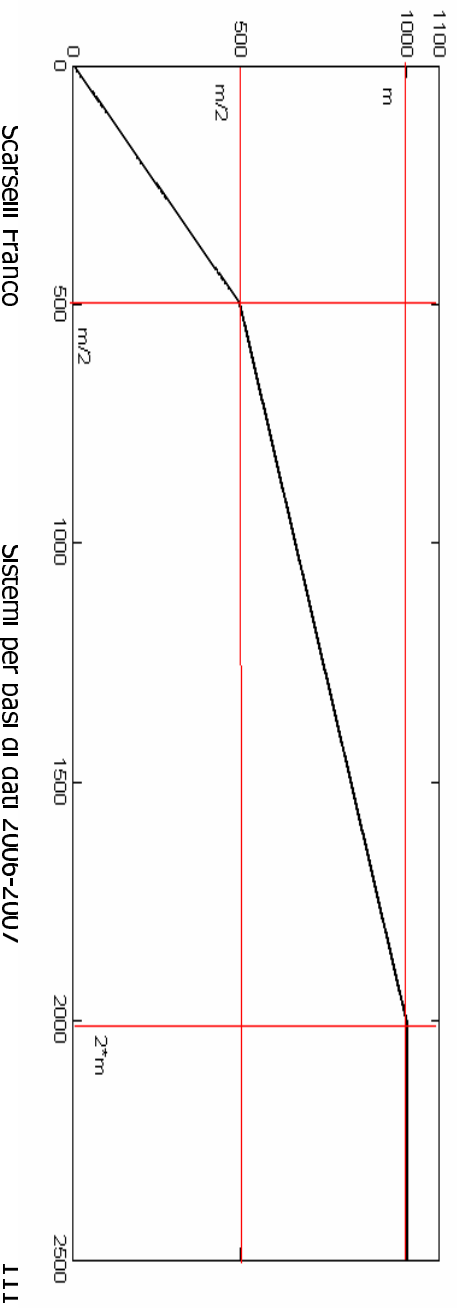
- Se $T' = \sigma_{A_i=v}(T)$
 - $CARD(T') = CARD(T)/VAL(A_i, T)$ (migliore stima in media)
 - $SIZE(T) = SIZE(T')$
 - $VAL(A_i, T')=1$
 - $VAL(A_j, T) = col(CARD(T), VAL(A_j, T), CARD(T'))$ $j \neq i$
 - numero di colori distinti selezionati scegliendo a caso $CARD(T')$ palline colorate da un insieme di $CARD(T)$ palline che hanno $VAL(A_j, T)$ colori diversi
 - $MAX(A_i, T') = MIN(A_i, T) = v$
 - $MAX(A_j, T') MIN(A_j, T') j \neq i$ mantengono i valori precedenti (ipotesi)
- $\sigma_{c_1 \wedge c_2}(T)$ è equivalente a $\sigma_{c_1} \sigma_{c_2}(T)$
- $\sigma_{c_1 \vee c_2}(T)$ è più difficile da stimare $CARD(T')$ - la somma dei risultati di una sovrastima sistemi per basi di dati 2006-2007

110

Esempio di implementazione di col

La seguente è un'implementazione possibile di $\text{col}(n,m,k)$

- Può essere implementata come
 - $\text{col}(n,m,k)=k$, se $k \leq m/2$
 - $\text{col}(n,m,k)=m$, $k > 2*m$
 - $\text{col}(n,m,k) = (k+m)/3$ se $m/2 < k < 2*m$



Profili per le proiezioni: un'implementazione

- Se $T' = \pi_L(T)$ $L = \{A_1, A_2, \dots, A_n\}$
 - $\text{CARD}(T') = \min(\text{CARD}(T), \prod_{i=1,n} \text{VAL}(A_i, T))$
 - $\text{SIZE}(T') = \sum_{i=1,n} \text{SIZE}(A_i, T)$
 - $\text{VAL}(A_i, T') \text{ MAX}(A_i, T') \text{ MAX}(A_i, T')$ mantengono i valori precedenti

Profili per il join: un'implementazione

- $T_J = T' \bowtie_{A=B} T''$
 - $CARD(T_J) = CARD(T') * CARD(T'') / \max(VAL(A, T'), VAL(B, T''))$
 - $SIZE(T_J) = SIZE(T') + SIZE(T'')$
 - $VAL(A_i, T_J) \text{ MAX}(A_i, T') \text{ MAX}(A_i, T'')$ mantengono i valori precedenti nelle rispettive tabelle
- Tutte le formule assumono una distribuzione uniforme dei valori e l'assenza di correlazione fra le varie condizioni presenti in una interrogazione
- Questi dati approssimati sono comunque sufficienti a ottenere un'ottimizzazione ragionevole

Scarselli Franco

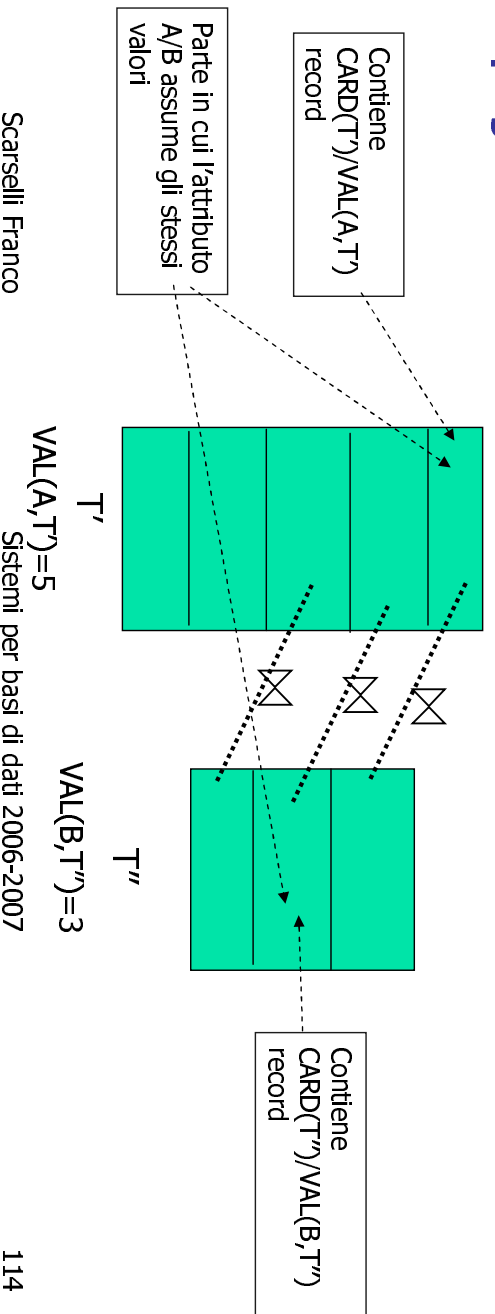
Sistemi per basi di dati 2006-2007

113

Profili per il join

- $T_J = T' \bowtie_{A=B} T''$
 - $CARD(T_J) = CARD(T') * CARD(T'') / \max(VAL(A, T'), VAL(B, T''))$
 $= \min(VAL(A, T'), VAL(B, T'')) * CARD(T') * CARD(T'') / (VAL(A, T') * VAL(B, T''))$

Spiegazione Intuitiva



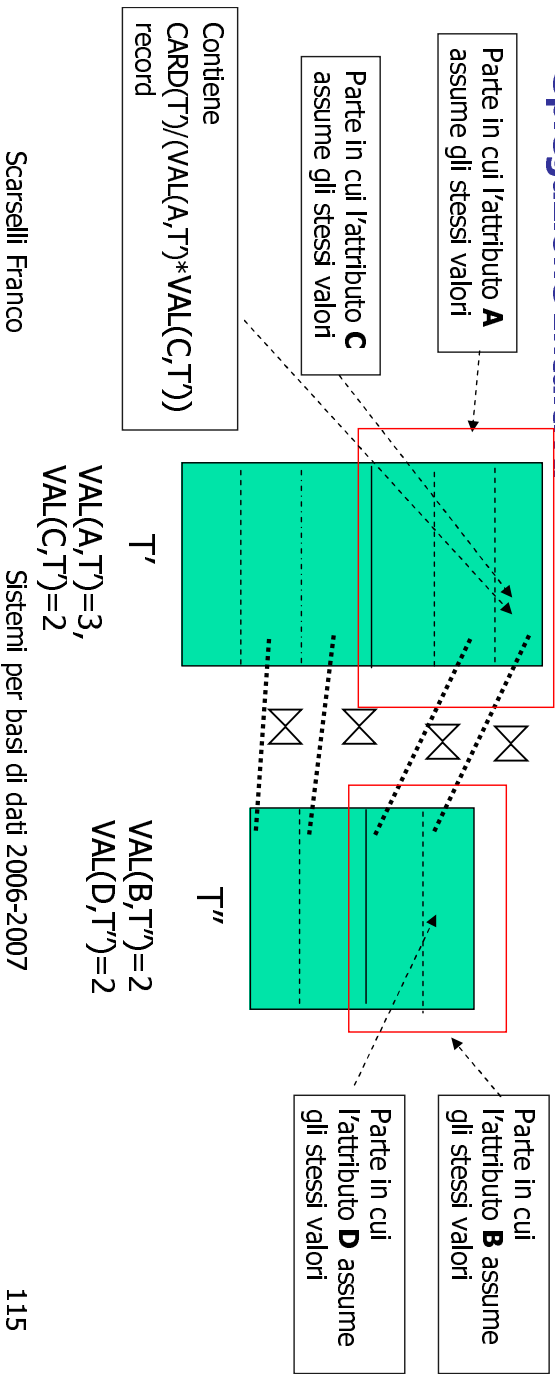
Scarselli Franco

114

Profili per il join: con più attributi

- $T^J = T' \bowtie_{A=B, C=D} T''$
 - $CARD(T^J) = CARD(T^J) * CARD(T'') / \max(VAL(A, T^J), VAL(C, T'')) * \max(VAL(B, T^J), VAL(D, T''))$

Spiegazione Intuitiva

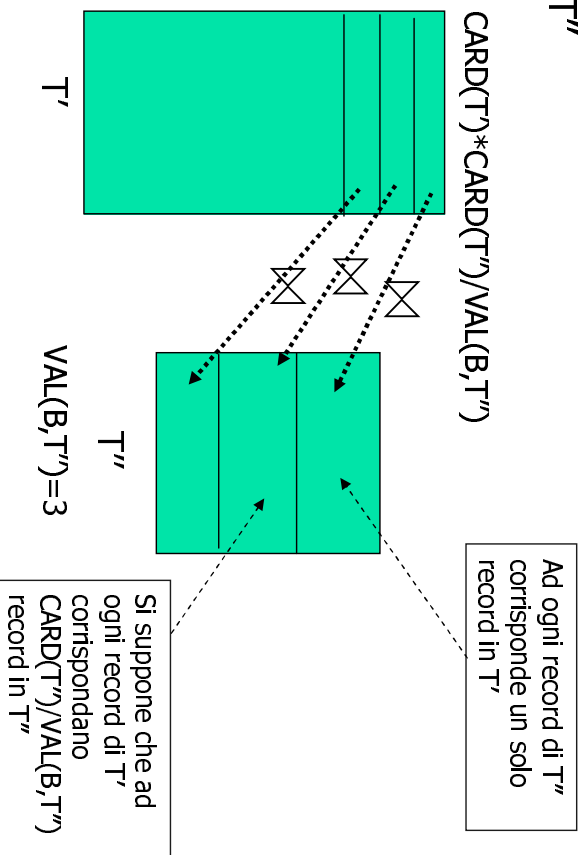


Scarselli Franco

115

Profili per il join: con tabelle slave e master

- Nel caso in A sia la chiave una tabella master T' e B la chiave esterna di una tabella slave T''
- $T^J = T' \bowtie_{A=B, T''} T''$
 - $CARD(T^J) = CARD(T^J) * CARD(T'') / VAL(B, T'')$



Scarselli Franco

116



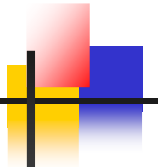
Progettazione fisica

- Definizione dei parametri fisici nel DBMS
- Scelta degli indici su ogni relazione
 - permettono di rendere più efficienti selezione e join
 - occorre scegliere gli attributi su cui crearli
 - creare un indice può essere costoso in spazio e tempo in modifica/inserimento
 - spesso le chiavi sono coinvolte in selezioni e join per cui conviene creare indici sui campi chiave (in alcuni DBMS sono creati automaticamente)
 - un indice può velocizzare anche le operazioni di ordinamento
 - su alcuni DBMS si può ottenere il piano degli accessi per verificare se gli indici sono utilizzati

Scarselli Franco

Sistemi per basi di dati 2006-2007

117



Controllo di affidabilità

Controllo di affidabilità

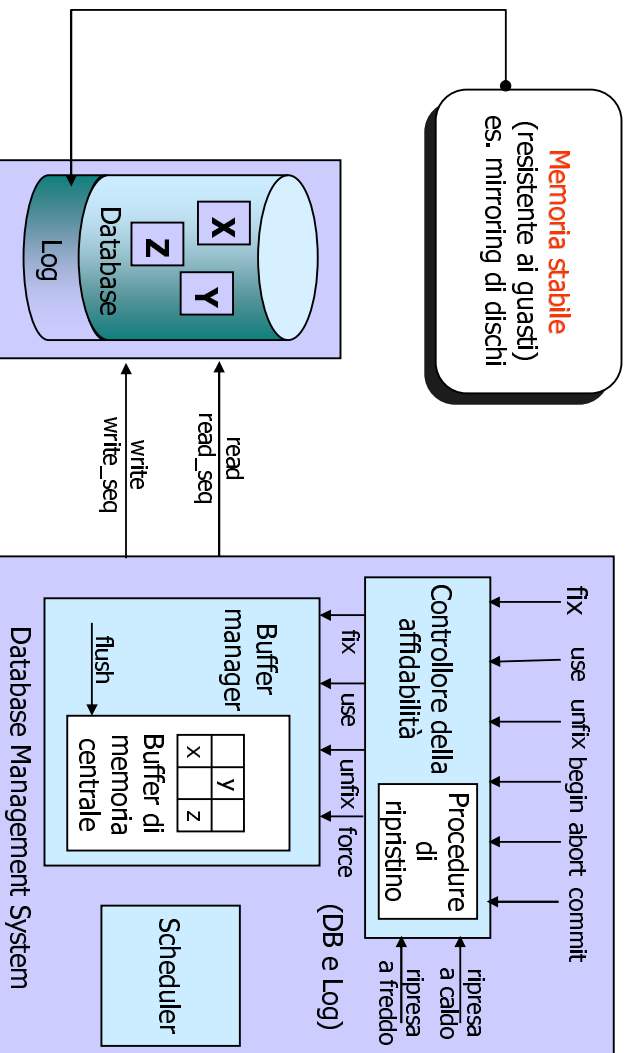
- Garantisce l'**atomicità** e la **persistenza** delle transazioni
- Si basa su un file di **log**
 - Il log registra le azioni (scritture) svolte dal DBMS
 - Il log permette di fare l'**undo** o il **redo** delle azioni
- Realizza i comandi transazionali
 - begin transaction (B)
 - commit work (C)
 - rollback work (A)
- Permette il ripristino in caso di malfunzionamenti (ripresa a caldo - ripresa a freddo)

Scarselli Franco

Sistemi per basi di dati 2006-2007

119

Controllore di affidabilità



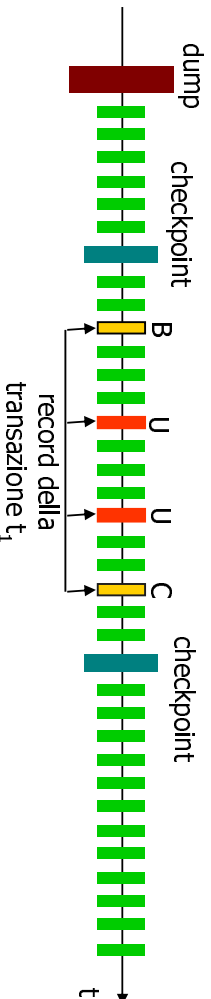
Scarselli Franco

Sistemi per basi di dati 2006-2007

120

Organizzazione del file di log

- Il file di log è un **file sequenziale** su cui vengono registrate le azioni svolte dalle transazioni in ordine temporale



- **record di transazione**
 - record di begin (**B**)
 - record relativi alle operazioni effettuate (**I**nsert - **U**ppdate - **D**elte)
 - record di commit (**C**) o abort (**A**)
- **record di sistema** (Dump - Checkpoint)

Scarselli Franco

Sistemi per basi di dati 2006-2007

121

Record di transazione

- **Begin** B(T) - **Commit** C(T) - **Abort** A(T)
Specificano l'identificativo T della transazione
- **Update** U(T,O,BS,AS)
Specificano la transazione T, l'oggetto O su cui si è fatto l'aggiornamento, il valore BS dell'oggetto prima della modifica (Before State) e quello AS dopo la modifica (After State)
- **Insert** I(T,O,AS)
Specificano la transazione T, l'oggetto O di cui si è fatto l'inserimento e il valore AS dopo l'inserimento
- **Delete** D(T,O,BS)
Specificano la transazione T, l'oggetto O di cui si è effettuata la cancellazione e il valore che aveva prima della cancellazione

Scarselli Franco

Sistemi per basi di dati 2006-2007

122



Checkpoint e dump

- **Checkpoint** - $C(T_1, T_2, \dots, T_k)$

- Operazione periodica per registrare tutte le transazioni attive
- Si scrivono su disco le pagine relative a transazioni terminate con commit o abort (flush)
- Non sono accettate operazioni di commit fino al termine del checkpoint
- Gli effetti delle transazioni che hanno eseguito un commit o un abort sono rese persistenti
- Si scrive nel log il record di checkpoint che contiene gli identificatori delle transazioni attive T_1, T_2, \dots, T_k
- **Dump** (backup) - DUMP
 - Copia completa della base di dati compiuta offline su supporto affidabile

Scarselli Franco

Sistemi per basi di dati 2006-2007

123



Undo e redo

- I record di transazione permettono di disfare (undo) o rifare (redo) le rispettive azioni

- **Undo**

- si ricopia in O il valore precedente BS
- per annullare l'insert si cancella l'oggetto O

- **Redo**

- si ricopia in O il valore AS
- per ripetere la cancellazione basta cancellare O

- Vale l'**idempotenza**: ripetendo più volte lo stesso undo (redo) si ottiene lo stesso effetto di una sola ripetizione

Scarselli Franco

Sistemi per basi di dati 2006-2007

124

Gestione delle transazioni

■ Regola WAL (Write Ahead Log)

- La parte BS deve essere scritta nel log prima di effettuare la corrispondente operazione nella base di dati
 - Viene mantenuto in modo affidabile il valore precedente alla scrittura
 - Permette di fare l'undo delle scritture di una transazione che non ha fatto il commit
 - In pratica il record di log è scritto completamente (BS e AS)
- ### ■ Regola di Commit-Precedenza
- La parte AS deve essere scritta nel log prima del commit
 - Permette di fare il redo delle transazioni che sono completate col commit - la scrittura del record di commit rende effettiva la transazione
 - In pratica il record di log è scritto completamente (BS e AS)

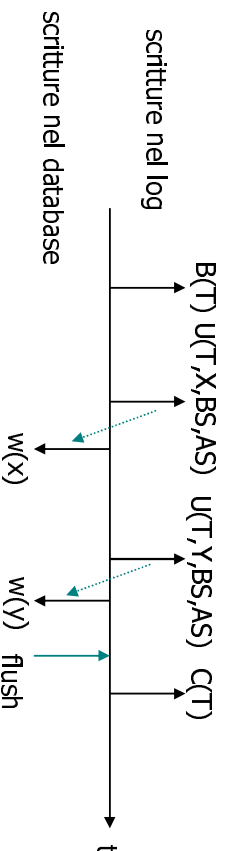
Scarselli Franco

Sistemi per basi di dati 2006-2007

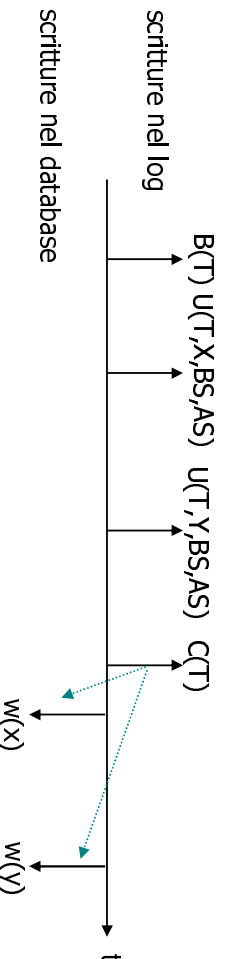
125

Protocolli di scrittura del log semplificata

- flush (force) prima del commit - non richiede il redo



- flush (force) dopo il commit - non richiede undo



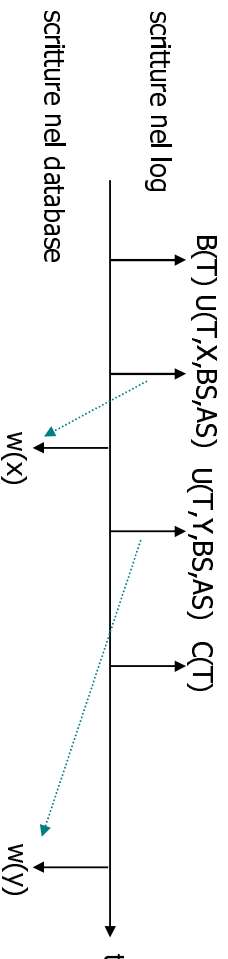
Scarselli Franco

Sistemi per basi di dati 2006-2007

126

Protocolli di scrittura del log

- nessun vincolo sul momento della scrittura rispetto al commit (solo le regole wall e commit-precedenza)



- Le operazioni di flush sono ottimizzate dal buffer manager
- Richiede redo e undo
- L'uso del file di log rappresenta un **carico aggiuntivo** per il sistema
- Devono essere adottate soluzioni efficienti per la scrittura del log

Scarselli Franco

Sistemi per basi di dati 2006-2007

127

Tipologie di guasti

■ Guasti di sistema

- Bug del software, cali di tensione
- Perdita del contenuto della memoria centrale (buffer)
- Persistenza dei dati in memoria secondaria (dati e log)

■ Guasti di dispositivo

- Rottura dei dispositivi di memoria di massa
- Si assume che il log sia salvato su memoria stabile (affidabile)
- Si perde il contenuto della base di dati ma non del log
- La perdita del log è un **evento catastrofico** senza recupero!

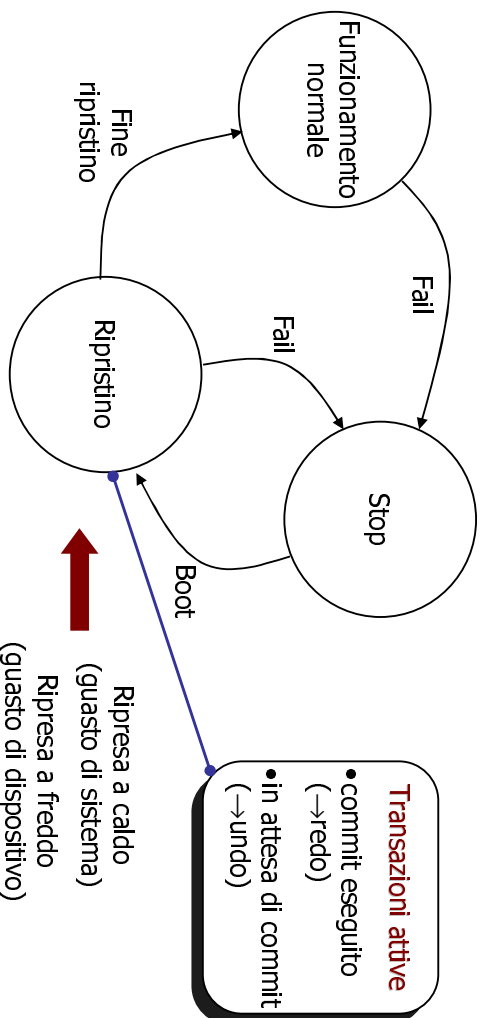
Scarselli Franco

Sistemi per basi di dati 2006-2007

128

Modello fail-stop

- **Fail-stop**
quando si rileva un guasto il sistema arresta tutte le transazioni



Scarselli Franco

Sistemi per basi di dati 2006-2007

129

Warm restart

- Si accede all'ultimo blocco del log e si ripercorre il log indietro fino al record di checkpoint
- Si determinano le transazioni da rifare (**REDO**) e da disfare (**UNDO**)
 - Inizializzazione
 - $UNDO = \{T_{transazioni\ attive\ al\ checkpoint}\}$
 - $REDO = \emptyset$
 - Scan del log in avanti
 - $B(T) \Rightarrow UNDO = UNDO \cup \{T\}$
 - $C(T) \Rightarrow REDO = REDO \cup \{T\}$ e $UNDO = UNDO / \{T\}$
 - Un'azione è nell'insieme $UNDO$ anche se la transazione è terminata con un record di abort $A(T)$

Scarselli Franco

Sistemi per basi di dati 2006-2007

130

Warm restart

- Si ripercorre all'indietro il log fino all'azione più vecchia delle transazioni in UNDO e REDO
 - Si può anche andare oltre il checkpoint (se le transazioni attive al checkpoint hanno azioni precedenti)
 - Si disfano le azioni delle transazioni nell'insieme UNDO
 - Si applicano le azioni delle transazioni nell'insieme di REDO nell'ordine in cui sono nel log
- **Atomicità**: viene garantito che le transazioni in corso al momento del guasto non siano lasciate in uno stadio intermedio
- **Persistenza**: le transazioni che hanno fatto il commit sono completate rendendo permanenti le modifiche al database

Scarselli Franco

Sistemi per basi di dati 2006-2007

131

Un esempio

- File di log che coinvolge le transazioni T_1, T_2, T_3, T_4, T_5
 $B(T_1) B(T_2) U(T_2, O_1, B_1, A_1) I(T_1, O_2, A_2) B(T_3) C(T_1) B(T_4)$
 $U(T_3, O_2, B_3, A_3) U(T_4, O_3, B_4, A_4) CK(T_2, T_3, T_4) C(T_4) B(T_5) U(T_3, O_3, B_5, A_5)$
 $U(T_5, O_4, B_6, A_6) D(T_3, O_5, B_7) C(T_5) I(T_2, O_6, A_8) FAIL$
- Si accede al checkpoint precedente al FAIL
 - $UNDO = \{T_2, T_3, T_4\}$ e $REDO = \{\}$
- Si aggiornano gli insiemi REDO e UNDO
 - $C(T_4)$: $UNDO = \{T_2, T_3\}$ e $REDO = \{T_4\}$
 - $B(T_5)$: $UNDO = \{T_2, T_3, T_5\}$ e $REDO = \{T_4\}$
 - $C(T_5)$: $UNDO = \{T_2, T_3\}$ e $REDO = \{T_4, T_5\}$

Scarselli Franco

Sistemi per basi di dati 2006-2007

132

Un esempio

$B(T_1)$ $B(T_2)$ $U(T_2, O_1, B_1, A_1)$ $I(T_1, O_2, A_2)$ $B(T_3)$ $C(T_1)$ $B(T_4)$
 $U(T_3, O_2, B_3, A_3)$ $U(T_4, O_3, B_4, A_4)$ $C(T_2, T_3, T_4)$ $C(T_4)$ $B(T_5)$ $U(T_3, O_3, B_5, A_5)$
 $U(T_5, O_4, B_6, A_6)$ $D(T_3, O_5, B_7)$ $C(T_5)$ $I(T_2, O_6, A_8)$ **FAIL**

- UNDO
 - T_2 : Delete(O_6) T_3 : Insert($O_5=B_7$); $O_3=B_5$; $O_2 = B_3$ T_2 : $O_1 = B_1$
- REDO
 - T_4 : $O_3 = A_4$ T_5 : $O_4 = A_6$

Cold restart

- Il guasto consiste in un deterioramento di una parte del database
- I passi del cold restart sono
 - Si ripristina la parte deteriorata dall'ultimo **backup** (dump)
 - Si accede al record di dump più recente nel log
 - Si ripercorre in avanti il log ripetendo tutte le azioni per ripristinare la parte deteriorata
 - Si esegue una ripresa a caldo

Backup

I backup

- servono in caso di guasti con **perdita dei dati**
- riguardano
 - **i dati**: si copia il contenuto dell'archivio (dopo si puo' tagliare il log)
 - **il log**: permette di ricostruire lo stato della base di dati ad un qualsiasi istante
- sono di due tipi
 - **completi**:
 - si copia tutto il contenuto dell'archivio
 - **il backup e' piu' lento**
 - **incrementali**:
 - si copia solo i cambiamenti dall'ultimo backup
 - per ripristino occorre prima ripristinare un backup completo e poi i backup incrementali successivi
 - **il ripristino è piu' lento**
- Tipicamente si alternano backup completi a backup incrementali:
(es. completo ogni settimana, incrementale ogni giorno)

Scarselli Franco

Sistemi per basi di dati 2006-2007

135

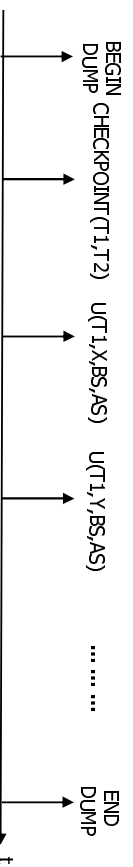
Backup on line

La soluzione piu' semplice

- disconnettere tutti gli utenti e fare il backup
- deve esistere un periodo sufficientemente lungo durante il quale il DBMS puo' essere fermato (ad es., la notte)

Backup con transazioni attive

- durante il backup
 - si scrive sul log il momento di inizio
 - si fa un checkpoint
 - si copiano i dati
 - si scrive sul log il momento di fine del backup
 - si copia il log



Scarselli Franco

Sistemi per basi di dati 2006-2007

136

Backup on line II

- durante il ripristino
 - si ricopia dal backup i dati andati persi
 - si applica una ripartenza a caldo, come se ci fosse stato un fallimento all'istante della fine del backup
 - l'archivio si ritrova nello stato del momento in cui è finito il backup (per metterlo nello stato del momento in cui è iniziato il backup occorre non usare i commit fra l'inizio e la fine del backup)

B(T1) B(T2) U(T2,O1,B1,A1) I(T1,O2,A2) C(T1) B(T3) U(T3,O3,B4,A4)
BEGIN_DUMP CK(T2,T3) C(T3) B(T4) U(T4,O4,B6,A6) C(T4) I(T2,O6,A8) END_DUMP

Log

UNDO = {T2} e REDO = {T3, T4} ←

Ripristino

Controllo di concorrenza

- Accesso al DBMS da parte di più utenti
- Il carico si misura in **tps** (transactions per second)
 - 10-10² tps in sistemi bancari
 - 10³ tps sistemi di prenotazione dei voli, gestori delle carte di credito
- La concorrenza permette un uso efficiente del DBMS massimizzando il numero di transazioni eseguite al secondo e minimizzando i tempi di risposta
- Si considerano le **operazioni di ingresso/uscita di basso livello**
 - trasferimenti fra la memoria di massa e la memoria centrale di blocchi di dati (pagine) - letture/scritture
- Il controllore della concorrenza è uno **scheduler** determina se le richieste possono essere soddisfatte

Scarselli Franco

Sistemi per basi di dati 2006-2007

139

Architettura

gestore delle concorrenza (scheduler)

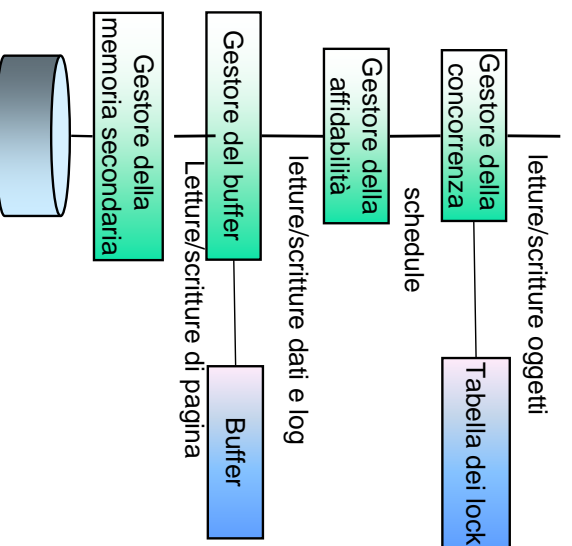
- garantisce che le transazioni concorrenti non interferiscano. Può usare una tabella dei lock

gestore dell'affidabilità

- gestisce le strategie di log e ripristino

gestore del buffer

- gestisce i trasferimenti da disco a memoria primaria



Nota che tali moduli non corrispondono necessariamente a moduli di un DBMS reale

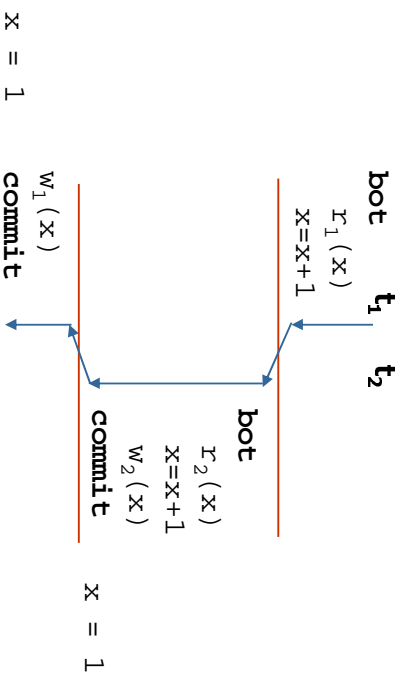
Scarselli Franco

Sistemi per basi di dati 2006-2007

140

Perdita di aggiornamento

- Si considerano due transazioni che operano sullo stesso dato
 - $t_1 : r(x), x=x+1, w(x)$
 - $t_2 : r(x), x=x+1, w(x)$
- se inizialmente $x=0$, dopo l'esecuzione seriale $x=2$



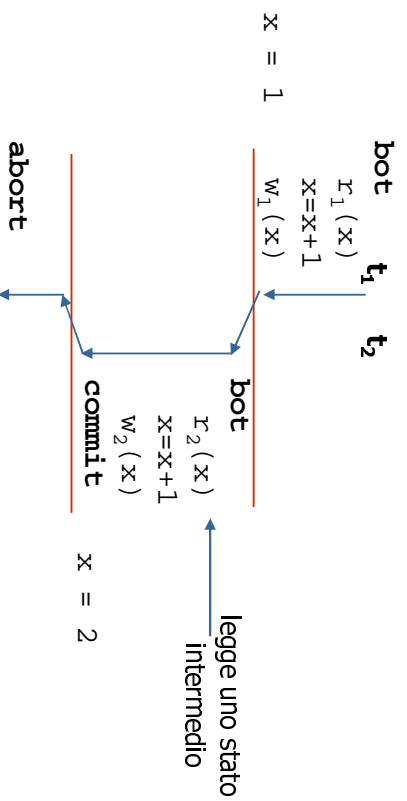
Scarselli Franco

Sistemi per basi di dati 2006-2007

141

Lecture sporche

- Si considerano le due transazioni t_1 e t_2 con fallimento di t_1



- Per il fallimento di t_1 , dovrebbe essere $x=1$ alla fine
- L'abort di t_1 dovrebbe causare il fallimento di t_2 (effetto domino)

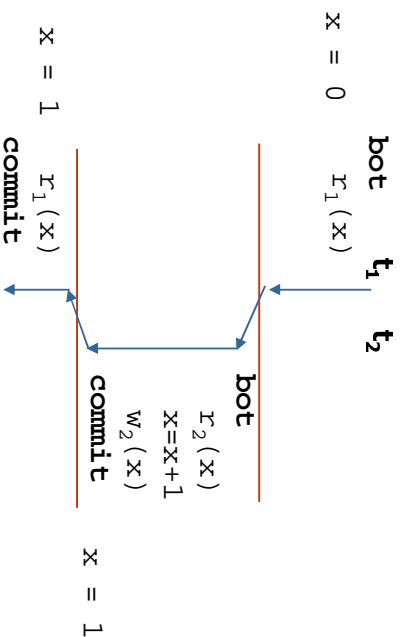
Scarselli Franco

Sistemi per basi di dati 2006-2007

142

Lecture inconsistenti

- La transazione t_1 ripete la lettura di x in istanti successivi



- All'interno della stessa transazione il valore letto non deve risentire dell'effetto di altre transazioni

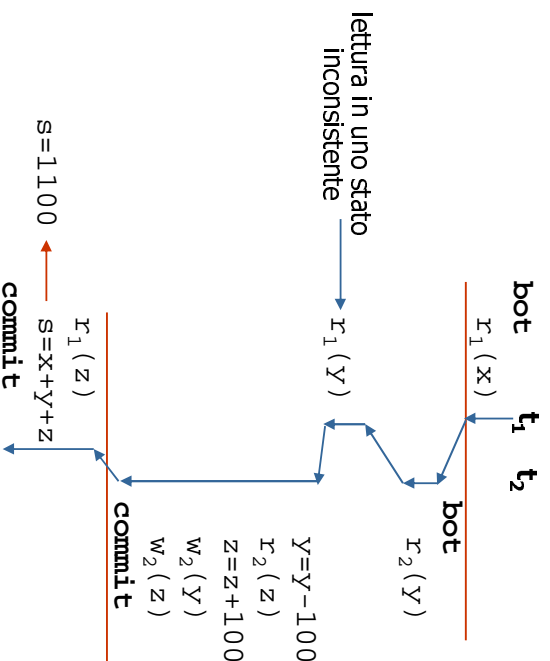
Scarselli Franco

Sistemi per basi di dati 2006-2007

143

Aggiornamento fantasma

- Si suppone che esista il vincolo di integrità $x+y+z=1000$



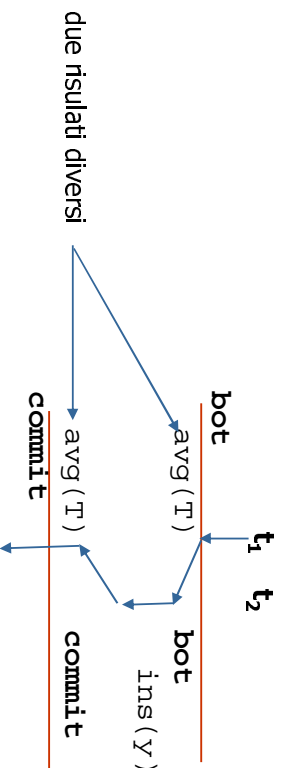
Scarselli Franco

Sistemi per basi di dati 2006-2007

144

Inserimento fantasma

- Una transizione calcola due volte una valore aggregato $avg(T)$ mentre l'altra inserisce un nuovo dato $ins(Y)$



- Esiste anche la rimozione fantasma

Scarselli Franco

Sistemi per basi di dati 2006-2007

145

Teoria della concorrenza

- Transazione** = sequenza di azioni di scrittura o lettura
- Ogni transazione ha un unico identificatore assegnato
- Ogni transazione è compresa fra i comandi **begin transaction** e **end transaction** (omessi)
- Un esempio è



- Il controllo di concorrenza accetta/rifiuta esecuzioni concorrenti durante l'evoluzione delle transizioni (senza saperne l'esito)

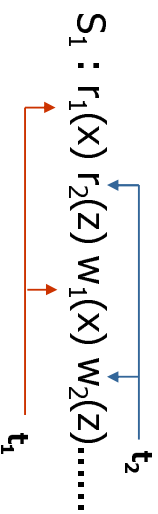
Scarselli Franco

Sistemi per basi di dati 2006-2007

146

Schedule

- Uno schedule è una sequenza di operazioni di lettura/scrittura corrispondente a transazioni concorrenti



- In teoria, si considerano solo transazioni che terminano con un commit (**commit-proiezioni**)
 - In pratica, il controllore della concorrenza deve occuparsi anche delle transizioni che terminano con un abort
- Obiettivo del controllo di concorrenza è di riuscire a generare solo schedule che non causano anomalie

Scarselli Franco

Sistemi per basi di dati 2006-2007

147

Schedule seriali e serializzabili

- Uno **schedule seriale** è uno schedule in cui tutte le operazioni di ogni transazione compaiono in sequenza

$$S_2 : \underbrace{r_0(X) \ r_0(Y) \ w_0(X)}_{t_0} \underbrace{r_1(Y) \ r_1(X) \ w_1(Y)}_{t_1} \underbrace{r_2(X) \ r_2(Y) \ r_2(Z) \ w_2(Z)}_{t_2}$$

- le transazioni godono della proprietà dell'isolamento
- Uno schedule è **serializzabile** se produce lo stesso risultato di uno schedule seriale delle stesse transazioni
 - l'utente non può distinguere fra schedule seriali e schedule serializzabili
- gli schedule serializzabili
 - si comportano come schedule seriali
 - sono piu' efficienti degli schedule seriali

Scarselli Franco

Sistemi per basi di dati 2006-2007

148

Schedule seriali e serializzabili II

La teoria

- si introduce la nozione di equivalenza di schedule
 - **view-equivalenza**, **conflict-equivalenza**, **locking a due fasi**, controllo basato su **timestamp**
- ciascuna equivalenza identifica una classe di schedule accettabili

La pratica

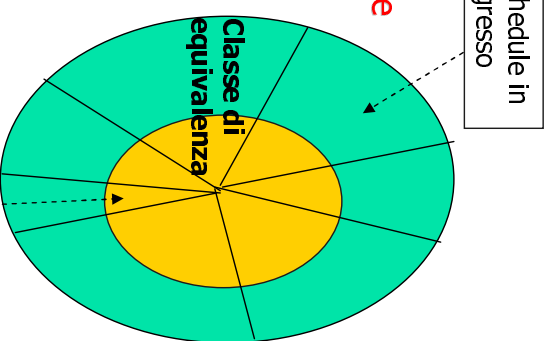
- uno scheduler trasforma la sequenza di transizioni in ingresso in schedule serializzabili equivalenti
 - fa in modo che le transizione serializzabili non si verifichino oppure
 - uccide le transazioni non serializzabili

- Il progetto dello scheduler deve bilanciare l'efficienza delle transazioni e l'efficienza quella dello scheduler

Scarselli Franco

Sistemi per basi di dati 2006-2007

149



View-equivalenza

Definizioni

- $r_i(x)$ **legge-da** $w_j(x)$ ($\text{legge}(r_i(x), w_j(x))$)
se $w_j(x)$ precede $r_i(x)$ e non esiste $w_k(x)$ compreso fra $w_j(x)$ e $r_i(x)$
..... $w_j(x)$ $r_i(x)$

- $w_i(x)$ è una **scrittura finale** se è l'ultima scrittura di x nello schedule
- Due schedule sono **view-equivalenti** ($S_i \approx_v S_j$) se su di essi sono definite le stesse relazioni legge-da e le stesse scritture finali
- Uno schedule è **view-serIALIZZABILE** se è view-equivalente ad un generico schedule seriale
- L'insieme degli schedule view-serIALIZZABILI è detto **VSR**

Scarselli Franco

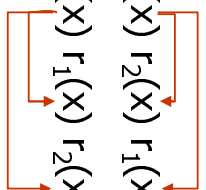
Sistemi per basi di dati 2006-2007

150

Un esempio: schedule view-equivalenti

$S_3 : w_0(x) \ r_2(x) \ r_1(x) \ w_2(x) \ w_2(z)$

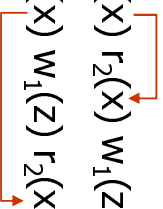
$S_4 : w_0(x) \ r_1(x) \ r_2(x) \ w_2(x) \ w_2(z)$



S_4 è uno schedule seriale; S_3 è serializzabile

$S_5 : w_0(x) \ r_1(x) \ w_1(x) \ r_2(x) \ w_1(z)$

$S_6 : w_0(x) \ r_1(x) \ w_1(x) \ w_1(z) \ r_2(x)$



S_6 è uno schedule seriale; S_5 è serializzabile

Schedule non serializzabili

- Perdita di aggiornamento

$S_7 : r_1(x) \ r_2(x) \ w_2(x) \ w_1(x)$
 $r_1(x) \ w_1(x) \ r_2(x) \ w_2(x)$
 $r_2(x) \ w_2(x) \ r_1(x) \ w_1(x)$


- Lettura inconsistente

$S_8 : r_1(x) \ r_2(x) \ w_2(x) \ r_1(x)$
 $r_1(x) \ r_1(x) \ r_2(x) \ r_2(x) \ w_2(x)$
 $r_2(x) \ w_2(x) \ r_1(x) \ r_1(x)$

Schedule seriali disponibili

- Aggiornamento fantasma

$S_9 : r_1(x) \ r_1(y) \ r_2(z) \ r_2(y) \ w_2(y) \ w_2(z) \ r_1(z)$



Complessità

- Decidere se due schedule sono view-equivalenti ha complessità **lineare**
 - E' efficiente confrontare fra loro due schedule
- Decidere se un generico schedule è view-serializzabile è un problema **NP-completo**
 - Occorre confrontare lo schedule con tutti gli schedule seriali ottenuti permutando le transazioni
 - **Non si può usare questa nozione per verificare la serializzabilità**

Scarselli Franco

Sistemi per basi di dati 2006-2007

153



Conflitti

- L'azione a_i è in **conflitto** con l'azione a_j ($i \neq j$) se operano sullo stesso oggetto ed almeno una è un write
 - Conflitti lettura-scrittura (rw o wr)
 - Conflitti scrittura-scrittura (ww)
- Intuitivamente:
 - due azioni sono in conflitto se non si può cambiare l'ordine senza effetti (Si considerano solo le due azioni)

Esempi: si può scambiare l'ordine delle azioni senza effetti

- $r_i(x) \dots r_j(y)$ non è mai un conflitto anche se $x=y$
Le letture non modificano i valori
- $r_i(x) \dots w_j(y) \dots r_i(x)$ o $w_j(y) \dots w_i(x)$ non è un conflitto se $x \neq y$
La modifica su y non cambia la lettura/scrittura su x

Scarselli Franco

Sistemi per basi di dati 2006-2007

154



Conflitti

Esempi: **non si può scambiare l'ordine delle seguenti azioni**

- $w_i(x) \dots w_j(x)$ è un conflitto
Il valore di x dopo l'esecuzione è definito da t_j . Scambiando l'ordine sarebbe invece definito da t_i
- $w_i(x) \dots r_j(x)$ è un conflitto
Il valore di x che deve essere letto è quello scritto da t_j
- $r_i(x) \dots w_j(x)$ è un conflitto
Il valore di x che deve essere letto è quello precedente alla scrittura da parte di t_j

Scarselli Franco

Sistemi per basi di dati 2006-2007

155



Conflict-equivalenza

- Si possono fare scambi che non cambiano i conflitti presenti
- Due schedule sono **conflict-equivalenti** ($S_i \approx_c S_j$) se
 - i due schedule presentano le stesse operazioni
 - ogni coppia di operazioni in conflitto è nello stesso ordine in entrambi gli schedule
- Uno schedule è **conflict-serIALIZZABILE** se è conflict-equivalente ad un generico schedule seriale
- Se due azioni sono in conflitto le corrispondenti transazioni nello schedule serializzato devono essere nello stesso ordine
- L'insieme degli schedule conflict-serIALIZZABILI è detto CSR
- Risulta $CSR \subset VSR$

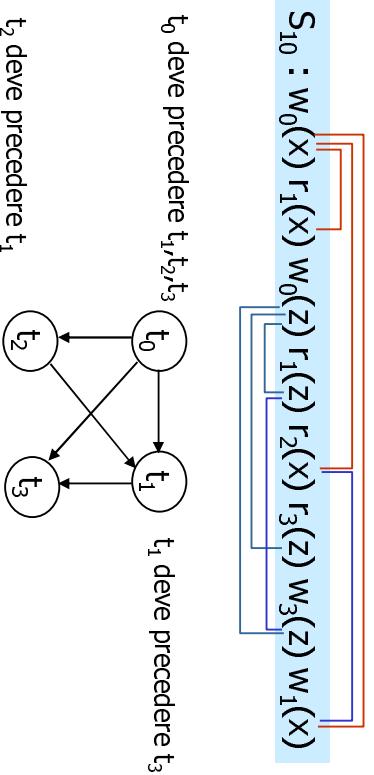
Scarselli Franco

Sistemi per basi di dati 2006-2007

156

Verifica della conflict-serializzabilità

- Si costruisce il grafo dei conflitti (o di precedenza)
 - ogni nodo corrisponde ad una transazione
 - Un arco fra t_i e t_j indica che c'è almeno un conflitto fra un'azione a_i e un'azione a_j tali che a_i precede a_j



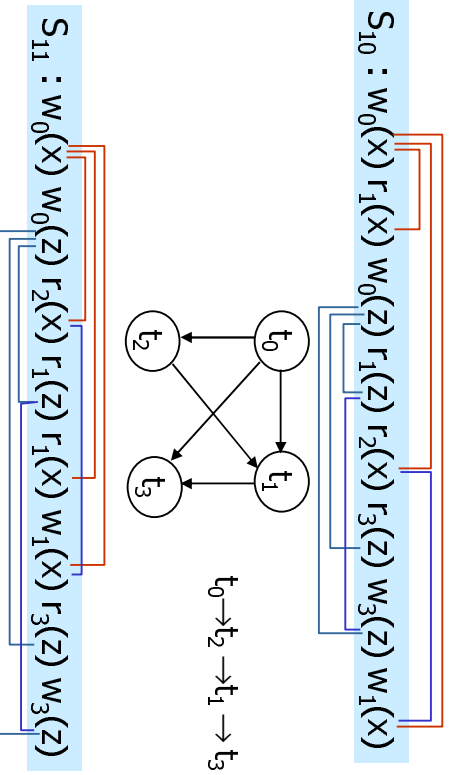
Scarselli Franco

Sistemi per basi di dati 2006-2007

157

Conflict-serializzabilità

- Uno schedule è CSR se e solo se il grafo è aciclico
- La serializzazione è data da un ordinamento topologico dei nodi
- La complessità è lineare nel numero dei nodi del grafo (transazioni)



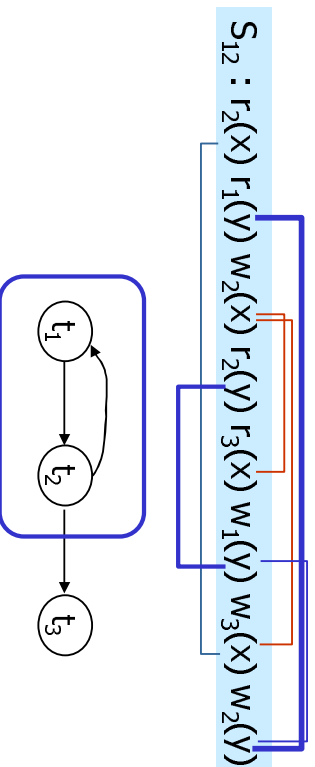
Scarselli Franco

Sistemi per basi di dati 2006-2007

158

Conflict-serializzabilità

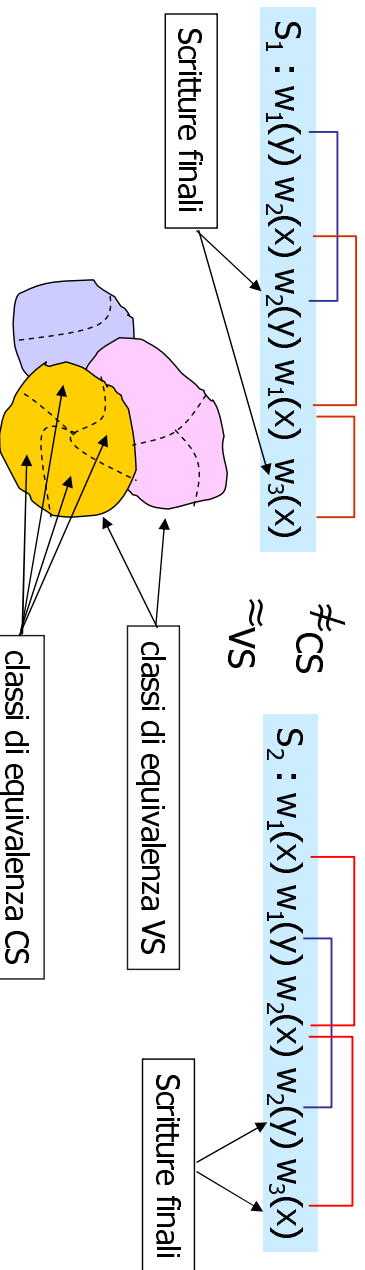
- Un esempio di schedule non conflict-serializzabile



- Non si può decidere se mettere prima t_1 o t_2
- Questo mostra perché lo schedule non è conflict-serializzabile se ci sono cicli nel grafo delle precedenze

Conflict e view serializzabilità

- Uno schedule conflict-serializzabile e' view-serializzabile ($CSR \subset VSR$)
- Non è vero il viceversa: ci sono schedule view-serializzabili che non sono conflict-serializzabili
 - le classi di equivalenza definiti dalla view-serializzabilità' sono piu' piccole di quelle definite dal conflict-serilizabilità'



Implementazione dello scheduler

Esistono due approcci diversi all'implementazione dello scheduler

- Approcci basati sul **controllo delle transazioni**
 - si controlla le transazioni evitando **il verificarsi delle anomalie**: lo scheduler genera solo schedule serializzabili
 - tipicamente questi approcci sono basati su locking
- Approcci **ottimistici**
 - assumono che **tutto andrà bene**, se poi si verificano anomalie, lo scheduler interviene
 - l'intervento consiste nell'uccidere la transazione colpevole dell'anomalia

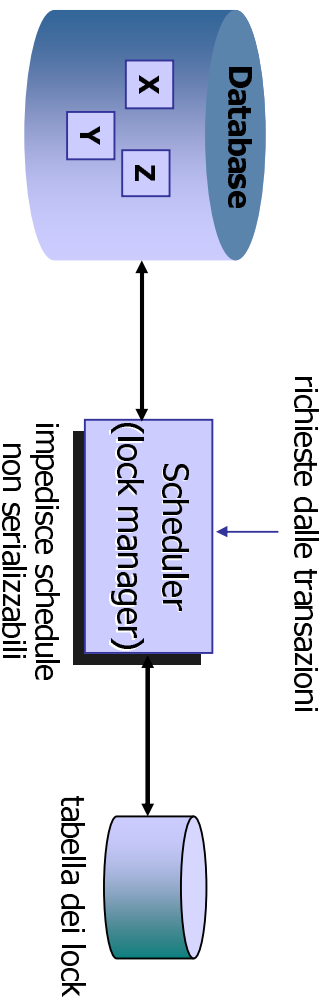
Scarselli Franco

Sistemi per basi di dati 2006-2007

161

Locks

- L'uso del locking è molto comune nei DBMS commerciali
- Le operazioni di lettura e scrittura sono protette dalle primitive
 - **r_lock** - read lock
 - **w_lock** - write lock
 - **unlock**



Scarselli Franco

Sistemi per basi di dati 2006-2007

162

Tipi di lock

- Ogni lock è applicato ad uno specifico dato del database
- I vincoli da rispettare sono
 - Ogni lettura è preceduta da un **r_lock** e seguita da un **unlock**
Il lock è **condiviso** dato che sullo stesso dato possono essere attivi più lock di questo tipo (tipo S)
 - Ogni scrittura è preceduta da un **w_lock** e seguita da un **unlock**
Il lock è **esclusivo** perché non può coesistere con altri lock sullo stesso dato (tipo X)
- Una transazione che rispetta queste regole si dice **ben formata rispetto al locking**
- In genere le richieste di lock e unlock sono inserite in modo trasparente

Scarselli Franco

Sistemi per basi di dati 2006-2007

163

Richieste di lock

- Quando la richiesta di lock è concessa, la transazione **acquisisce** la corrispondente risorsa
- Con l'esecuzione di unlock la risorsa è **rilasciata**
- Quando la richiesta di lock non viene concessa la transazione richiedente viene messa in **stato di attesa**

Richiesta	Stato della risorsa		
	libero	S	X
S	OK/S	OK/S	NO/X
X	OK/X	NO/X	NO/X
Unlock	Error	OK/dipende	OK/libero

Si: la risorsa viene concessa

No: la transazione è bloccata

Tabella di compatibilità → libero se non ci sono più richieste S

Scarselli Franco

Sistemi per basi di dati 2006-2007

164

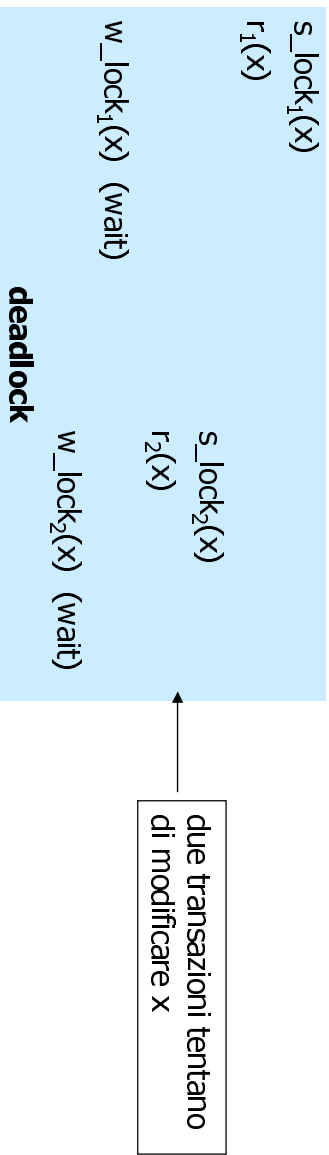
Aumentare il livello di lock

Se si vuol prima leggere e poi scrivere un oggetto

- si acquisisce prima un lock condiviso poi si incrementa a lock esclusivo

$s_lock(x)$ $r(x)$ $w_lock(x)$ $w(x)$ $unlock(x)$

- questa strategia aumenta la probabilita' di deadlock



Lock di tipo update

- $u_lock(x)$: lock di tipo update (tipo U)

- fornisce il privilegio solo di lettura su x alla transazione t_i
- può essere trasformato in un lock esclusivo (write)
- una volta che è attivo su x non si possono aggiungere ulteriori lock attivi (le transazioni corrispondenti si mettono in wait)

Richiesta	Stato		
	S	X	U
S	OK/S	No	No
X	No	No	No
U	OK/U	No	No

Si: la risorsa viene concessa

No: la transazione è bloccata

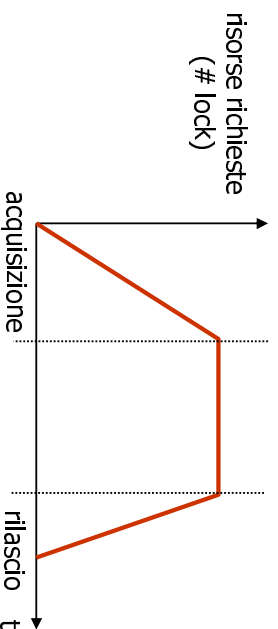
Tabella di compatibilità

Locking a due fasi

- Per garantire che le transazioni seguano uno schedule serializzabile si introduce il **two phase locking** (2PL)

Una transazione dopo aver rilasciato un lock non può acquisirne altri

- Perché 2 fasi
 - prima si acquisiscono i lock per le risorse necessarie (fase crescente)
 - poi si rilasciano i lock acquisiti (fase calante)



Scarselli Franco

Sistemi per basi di dati 2006-2007

167

2PL e CSR

- Ogni schedule che rispetta il protocollo 2PL è anche serializzabile rispetto alla conflict-equivalenza **2PL \subset CSR**

- Dimostrazione per assurdo

Se $S \in 2PL$ e $S \notin CSR$, il grafo dei conflitti è ciclico

$$t_1 \rightarrow t_2 \rightarrow \dots, t_n \rightarrow \dots t_1$$

- esiste una risorsa **R1** su cui t_1 e t_2 operano entrambe in modo conflittuale
- t_2 può proseguire solo quando t_1 rilascia il lock sulla risorsa
- iterando per ogni i.....
- esiste una risorsa **Rn** su cui t_n e t_1 operano entrambe in modo conflittuale
- Affinché t_1 possa procedere è necessario che acquisisca il lock sulla risorsa **Rn** rilasciata da t_n
- La transazione t_1 rilascia un risorsa (**R1**) prima di acquisirne un'altra (**Rn**), ovvero lo schedule non può essere 2PL

Scarselli Franco

Sistemi per basi di dati 2006-2007

168

CSR e 2PL

- Uno schedule che è conflict-serializzabile non è detto che sia 2PL

$S_{13} : r_1(x) \ w_1(x) \ r_2(x) \ w_2(x) \ r_3(y) \ w_1(y)$

$t_3 \rightarrow t_1 \rightarrow t_2$

$S_{14} : r_3(y) \ r_1(x) \ w_1(x) \ w_1(y) \ r_2(x) \ w_2(x)$

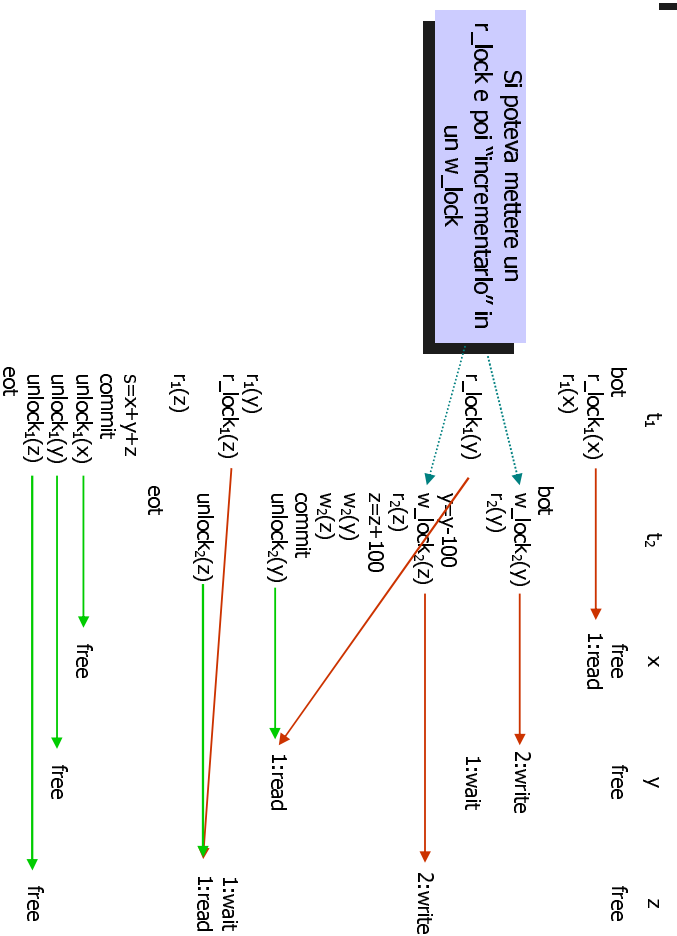
- Non è 2PL perché in S_{13} la transazione t_1 deve liberare il lock su x e poi richiedere il lock su y

Scarselli Franco

Sistemi per basi di dati 2006-2007

169

Modifica fantasma: soluzione



Scarselli Franco

Sistemi per basi di dati 2006-2007

170



Locking a 2 fasi “stretto”

- Si rimuove l'ipotesi che tutte le transazioni vadano a buon fine (commit-proiezione)

I lock di una transazione possono essere rilasciati solo dopo aver effettuato correttamente le operazioni di commit/abort

- I lock vengono rilasciati solo al termine della transazione dopo che ogni dato è in uno stato consistente
- E' utilizzato nei DBMS commerciali
- Elimina l'anomalia della lettura sporca (dovuta alla presenza degli abort)

Scarselli Franco

Sistemi per basi di dati 2006-2007

171



Gestione dei lock

- Le operazioni di gestione dei lock hanno in genere il seguente prototipo

- `r_lock(T,x,errcode,timeout)`
 - `w_lock(T,x,errcode,timeout)`
 - `unlock(T,x)`

- **T**: identificatore della transazione
- **x**: risorsa su cui si richiede/rilascia il lock
- **errcode**: codice che indica l'esito della richiesta (0 richiesta eseguita correttamente, diverso da 0 in caso di errore)
- **timeout**: tempo massimo che si attende per ottenere il lock sulla risorsa (se scade errcode assume il valore opportuno)

Scarselli Franco

Sistemi per basi di dati 2006-2007

172

Gestione dei lock

- Se la richiesta può essere soddisfatta
 - il lock manager cambia lo stato della risorsa nelle tabelle dei lock
 - viene restituito il controllo al processo che ha effettuato la richiesta
- Se la richiesta non può essere soddisfatta
 - il processo richiedente viene inserito in una coda associata alla risorsa
 - il processo richiedente viene sospeso
 - quando la risorsa viene rilasciata si controlla se ci sono processi in attesa e nel caso si concede la risorsa al processo in testa alla coda
- Se scatta un timeout
 - la transizione fallisce e si esegue un rollback
 - si prova a richiedere nuovamente il lock

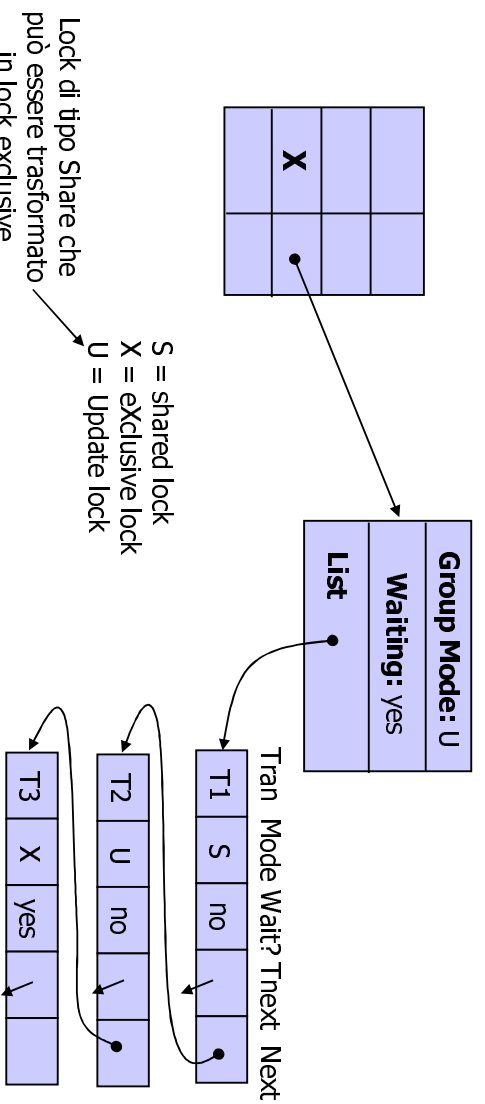
Scarselli Franco

Sistemi per basi di dati 2006-2007

173

Tabelle dei lock

- Sono mantenute in memoria principale per motivi di efficienza
- Ad ogni oggetto è associato lo stato



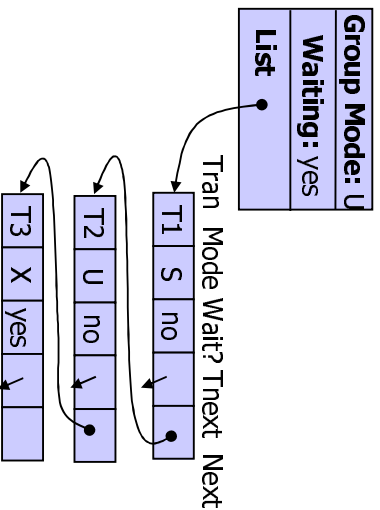
Scarselli Franco

Sistemi per basi di dati 2006-2007

174



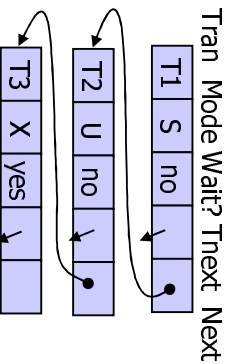
Tabelle dei lock



- Il **group mode** individua la condizione più stringente attiva sulla risorsa
 - S se sono attivi solo shared lock
 - U se c'è un update lock e forse uno o più shared lock
 - X se c'è un exclusive lock
- Il bit **Waiting** indica se c'è almeno una transazione in attesa sulla risorsa
- **List** contiene tutte le transazioni che al momento possiedono il lock o che sono in attesa di ottenerlo



Tabelle dei lock



- **Tran** è l'identificatore della transazione che detiene o attende un lock
- **Mode** indica il tipo di lock (S X U)
- **Wait?** è un flag che indica se la transazione detiene o attende un lock
- **Tnext** permette di collegare i lock relativi alla stessa transazione. Questa lista può essere usata quando la transazione esegue un commit o un abort per rilasciare tutti i lock

Granularita' di locking

Quali sono gli oggetti su cui si deve porre un lock ?

- L'intera tabella
 - Per eliminare il problema dell'inserimento fantasma
 - Per un'interrogazione con operatore aggregato
- Alcuni blocchi
 - operazioni di modifica su un range (record consecutivi)
- Le tuple
 - la modifica di una tupla
- Il campo
 - La modifica di uno o pochi campi

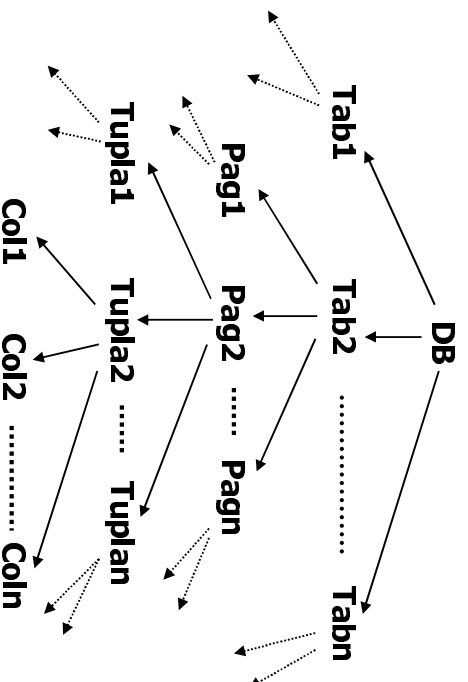
Scarselli Franco

Sistemi per basi di dati 2006-2007

177

Granularita' di locking II

- E' possibile specificare i lock a livelli diversi (**granularità dei lock**)
 - tabelle, pagine, tuple, campi di singole tuple



Scarselli Franco

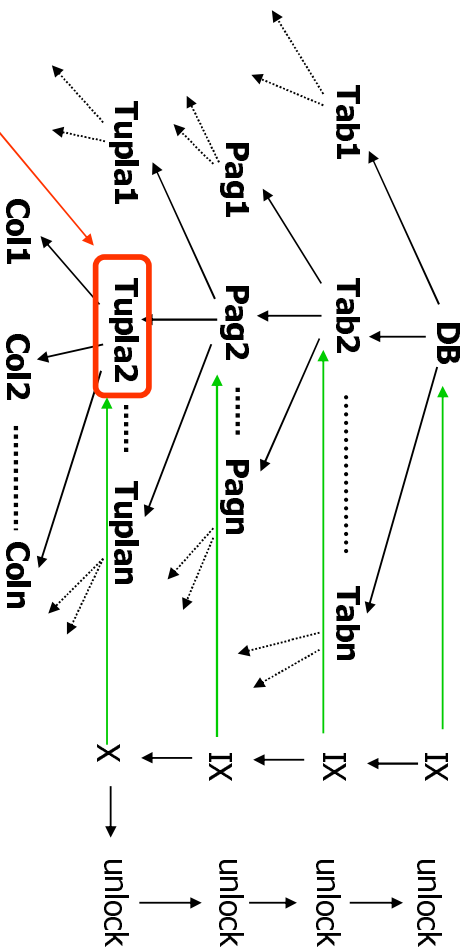
Sistemi per basi di dati 2006-2007

178

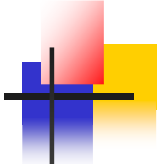
Primitive per il lock gerarchico

- Si aggiungono lock specifici oltre ai lock S e X
 - **IS : Intentional Shared lock**
Esprime l'intenzione di bloccare in modo condiviso uno dei nodi discendenti del nodo corrente
 - **IX : Intentional exclusive lock**
Esprime l'intenzione di bloccare in modo esclusivo uno dei nodi discendenti del nodo corrente
 - **SIX : Shared Intentional-exclusive lock**
Blocca il nodo corrente ed esprime l'intenzione di bloccare in modo esclusivo uno dei nodi discendenti del nodo corrente

Lock gerarchico



Lock X sulla tupla



Protocollo di lock gerarchico

- Si richiedono i lock a partire dalla radice scendendo lungo l'albero
- Si rilasciano i lock a partire dal nodo verso la radice
- Per richiedere un lock S o IS su un nodo si deve possedere un lock IS o IX sul padre
- Per poter richiedere un lock IX, X o SIX su un nodo si deve già possedere un lock SIX o IX sul padre
- E' definita la tabella di compatibilità per stabilire se accettare la richiesta di lock

Scarselli Franco

Sistemi per basi di dati 2006-2007

181

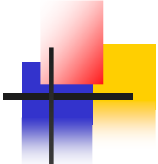


Tabella di compatibilità

Richiesta	Stato risorsa				
	IS	IX	S	SIX	X
IS	Si	Si	Si	Si	No
IX	Si	Si	No	No	No
S	Si	No	Si	No	No
SIX	Si	No	No	No	No
X	No	No	No	No	No

Scarselli Franco

Sistemi per basi di dati 2006-2007

182

Lock e B tree

Problema

- si parte dalla radice acquisendo i lock di tutti i nodi incontrati fino ai dati che vogliamo modificare/leggere
- il locking a due fasi prevede che un lock sia rilasciato solo dopo il commit
- quindi la root **rimane bloccata**
 - con gli inserimenti tutto l'albero rimane completamente bloccato (concorrenza inesistente)

Soluzione

- definire un nuovo algoritmo di lock per gli alberi che
 - rilasci i lock non appena possibile
 - garantisca comunque la serializzabilit 

Scarselli Franco

Sistemi per basi di dati 2006-2007

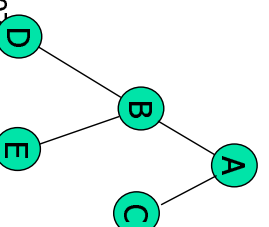
183

Lock e B tree II

Algoritmo

- Inizialmente, la transizione acquisisce un **lock sulla radice**
- Successivamente, puo' essere acquisito un lock su un qualsiasi nodo solo se la transizione possiede un lock sul padre
- Un lock puo' essere rilasciato in qualsiasi momento
- Un lock rilasciato non puo' essere riacquisito

$w_lock_1(A), r_1(A)$ $w_lock_1(C), w_1(C)$ $unlock_1(C)$	$s_lock_2(A) \text{ (wait)}$
$w_lock_1(B), r_1(B)$ $unlock_1(A)$	$s_lock_2(C)$ $unlock_2(A)$ $r_2(C)$ $unlock_2(C)$
$w_lock_1(E)$ $unlock_1(B)$ $w_1(E)$ $unlock_1(E)$	



Scarselli Franco

Sistemi per basi di dati 2006-2007

184

Lock e B tree III

Osservazioni

- Un lock condiviso su un nodo **n** puo' essere rilasciato
 - **non appena** si acquisisce il lock sul figlio **f**
- Un lock esclusivo su un nodo **n** puo' essere rilasciato
 - quando si acquisisce il lock sul figlio **f**
 - se **non ci sono rischi che la modifica si propaghi**
 - modifica di inserimento ed f e' pieno
 - modifica di eliminazione ed f e' al limite della minima occupazione

Serializzabilita'

- Si puo' dimostrare che usando il protocollo descritto si generano schedule conflict-serIALIZZABILI
- Le transizioni possono essere ordinate sulla base all'ordine di acquisizione del lock sulla radice

Scarselli Franco

Sistemi per basi di dati 2006-2007

185

Deadlocks

- E' possibile che si verifichi uno stallo (**deadlock**)

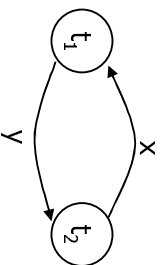
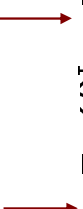
- si verifica in caso di **schedule non serializzabile secondo 2PL**

$t_1 : r_1(x) w_1(y)$

$t_2 : r_2(y) w_2(x)$

- Possibile schedule con 2PL

$r_lock_1(x) \ r_lock_2(y) \ r_1(x) \ r_2(y) \ w_lock_1(y) \ w_lock_2(x)$



t_1 si blocca aspettando
che si liberi y bloccato
da t_2

t_2 si blocca aspettando
che si liberi x bloccato
da t_1

Scarselli Franco

Sistemi per basi di dati 2006-2007

186



Deadlocks II

La probabilità di un deadlock

Numero transazioni: t

Numero di oggetti disponibili: n

Numero medio di oggetti bloccati per transazione: m

La probabilità che una transazione T_1 stia attendendo una risorsa bloccata da un'altra transazione T_2 è m/n

La probabilità di un deadlock causato da T_1 e T_2 (deadlock di lunghezza due) è m^2/n^2

La probabilità che esista un deadlock di lunghezza due è $O(t^2m^2/n^2)$

- Poiché normalmente $n \gg mt$, la probabilità di avere un deadlock è bassa
- Si tratta di una sottostima a causa della **località** delle transazioni
- In pratica, i **deadlock** sono rari ma possibili

Scarselli Franco

Sistemi per basi di dati 2006-2007

187



Soluzioni al deadlock

Tre soluzioni

■ Timeout

- Le transazioni rimangono in attesa di una risorsa per un tempo prefissato
- Una risorsa in deadlock viene comunque liberata dopo il timeout
- Rimane difficile determinare il timeout ottimale
- Il timeout è comunque utile (una transazione blocca una risorsa a lungo)

■ Prevenzione (deadlock prevention)

- Se una transazione può causare un deadlock, allora viene uccisa
- si fa in modo da non generare mai deadlock

■ Rilevamento (deadlock detection)

- Si controlla periodicamente il sistema alla ricerca di situazioni di stallo e si uccide una delle transazioni coinvolte nel deadlock

Scarselli Franco

Sistemi per basi di dati 2006-2007

188

Rilevamento e prevenzione attraverso grafo dei conflitti

Strategia

- Si costruisce il grafo dei conflitti
- Se è ciclico allora c'è un deadlock

Osservazioni

- Il grafo può essere aggiornato tutte le volte che una transazione richiede/rilascia una risorsa
 - in questo caso si tratta di **prevenzione dei deadlock**
- Il grafo può essere aggiornato in corrispondenza dei checkpoint o a scadenze predefinite
 - in questo caso si tratta di **rilevazione dei deadlock**
 - la transazione da uccidere può essere scelta in base ad un timestamp o scegliendo quella che ha fatto meno lavoro (accessi)

Scarselli Franco

Sistemi per basi di dati 2006-2007

189

Prevenzione attraverso ordinamento

La strategia

- Si usa 2PL
- Si ordinano gli oggetti del database
 - es. in base al loro indirizzo
- Una transazione deve richiedere le risorse rispettando l'ordinamento
 - es. se una transazione richiede in sequenza R_1, R_2, \dots, R_n , deve essere verificato che $R_1 < R_2 < \dots < R_n$

Osservazioni

- In questo modo **non si verificano stalli**
- spesso non è possibile conoscere in anticipo quali risorse serviranno

T_1 ha bloccato le risorse R_1, R_2, \dots, R_n T_2 ha bloccato le risorse S_1, S_2, \dots, S_m
 T_2 è in attesa della risorsa R_i di T_1

T_1 non può essere in attesa di S_j di T_2 altrimenti $R_i \leq R_n \leq S_j \leq S_n < R_i$

Scarselli Franco

Sistemi per basi di dati 2006-2007

190

Un esempio

- Si suppone che gli oggetti A,B,C,D siano ordinati alfabeticamente
- T1: r(A), w(B)
- T2: r(C), w(A)
- T3: r(B), w(C)
- T4: r(D), w(A)

l=lock
u=unlock

T1	T2	T3	T4
l(A), r(A)			
	l(A) (wait)		
		l(B), r(B)	
			l(A) (wait)
		l(C), w(C)	
		u(B), u(C)	
l(B), w(B)			
u(A), u(B)			
	l(A), l(C)		
	r(C), w(A)		
	u(C), u(A)		
			l(A), l(D)
			r(D), w(A)
			u(D), u(A)

Scarselli Franco

Sistemi per basi di dati 2006-2007

191

Prevenzione attraverso timestamp

- Ad ogni transazione si assegna un **timestamp** (inizio della transazione)
- Quando una transazione T richiede una risorsa bloccata da S si confrontano i due timestamp h_t e h_s
- **Politica non interrompente**
 - Se $h_t < h_s$, T può attendere il rilascio della risorsa
 - Se $h_t > h_s$, T viene uccisa
- **Politica interrompente**
 - Se $h_t < h_s$, S viene forzata a rilasciare la risorsa
 - Se $h_t > h_s$, T può attendere il rilascio della risorsa
- Le transazioni uccise sono rilanciate con lo stesso timestamp che avevano all'inizio per evitare problemi di **starvation**

Scarselli Franco

Sistemi per basi di dati 2006-2007

192

Esempi

T1: $r(A), w(B)$; T2: $r(C), w(A)$; T3: $r(B), w(C)$; T4: $r(D), w(A)$

T1	T2	T3	T4
$I(A), r(A)$			
	$I(A)$ (uccisa)		
		$I(B), r(B)$	
			$I(A)$ (uccisa)
		$I(C), w(C)$	
		$u(B), u(C)$	
$I(B), w(B)$			
$u(A), u(B)$			
	$I(A)$ (wait)		$I(A), I(D)$
			$r(D), w(A)$
			$u(D), u(A)$
	$I(A), I(C)$		
	$r(C), w(A)$		
	$u(C), u(A)$		

Politica non interrompente

Scarselli Franco

Sistemi per basi di dati 2006-2007

193

T1	T2	T3	T4
$I(A), r(A)$			
	$I(A)$ (wait)		
		$I(B), r(B)$	
			$I(A)$ (wait)
		(uccisa)	
$I(B), w(B)$			
$u(A), u(B)$			
	$I(A), I(C)$		
	$r(C), w(A)$		
	$u(C), u(A)$		
			$I(A), I(D)$
			$r(D), w(A)$
			$u(D), u(A)$
		$I(B), r(B)$	
		$I(C), w(C)$	
		$u(B), u(C)$	

Politica interrompente

Prevenzione attraverso timestamp II

Perche' funziona ?

- Si supponga che esiste un ciclo $T_1, T_2, \dots, T_n, T_1$ nel grafo dei conflitti
- Con la politica non interrompente le transizioni piu' vecchie aspettano quelle piu' giovani: $h_1 < h_2 < \dots < h_n < h_1$
- Con la politica interrompente le transizioni piu' giovani aspettano quelle piu' vecchie: $h_1 > h_2 > \dots > h_n > h_1$

Vantaggi e svantaggi (assumendo che molti lock siano acquisiti all'inizio)

- Politica non interrompente
 - Si uccide le transazioni che hanno fatto meno lavoro
- Politica interrompente
 - Si uccide meno transazioni

Scarselli Franco

Sistemi per basi di dati 2006-2007

194



Un paragone fra i meccanismi per risolvere i deadlock

Il grafo dei conflitti

- E' il metodo **comunemente** usato dai DBMS commerciali
- riduce al minimo i roll back
- è dispendioso e complicato: occorre mantenere il grafo dei conflitti

Prevenzione attraverso ordinamento

- richiede di conoscere in anticipo gli oggetti da bloccare

Prevenzione attraverso timestamp

- può succedere che transazioni che non causeranno conflitti vengano uccise e devano essere rilanciate
- ha un ruolo più importante in un ambiente distribuito, dove il mantenimento del grafo dei conflitti è più gravoso

Scarselli Franco

Sistemi per basi di dati 2006-2007

195



Riassumendo

- La teoria suggerisce le proprietà che uno scheduler deve avere perché non possieda anomalie: deve essere serializzabile
- La view-serializzabilità fornisce **schedule più efficienti**, ma implicerebbe uno **scheduler inefficiente**
- 2PL permette di realizzare uno **scheduler efficiente**
- Con i **lock** si implementa 2PL, quindi si garantiscono schedule senza anomalie (isolamento delle transazioni)
- L'uso dei lock può introdurre deadlock (si verificano quando lo schedule è non serializzabile secondo 2PL)
- Occorre adottare strategie di **prevenzione/riconoscimento** dei deadlock

Scarselli Franco

Sistemi per basi di dati 2006-2007

196

Altri approcci al controllo della concorrenza

Il meccanismo dei lock assume

- se non si controlla le transazioni si verificano anomalie

Gli **approcci ottimistici** assumono

- tutto andrà bene, altrimenti interveniamo
- l'intervento consiste nell'uccidere la transazione colpevole dell'anomalia
- **basati su timestamp**
 - si verifica che lo schedule sia equivalente ad uno seriale usando dei timestamp per le transazione e le operazioni di lettura e scrittura
- **basati su validazione**
 - si esaminano i timestamp prima che la transazione faccia il commit

Scarselli Franco

Sistemi per basi di dati 2006-2007

197

Controllo della concorrenza basato su timestamp

Si assegnano

- per ogni transizione T un timestamp **TS(T)**
- per ogni oggetto X
 1. Il maggiore timestamp fra quelli delle transazioni che hanno letto X: **RT(X)** (**tempo di lettura**)
 2. Il maggiore timestamp fra quelli delle transazioni che hanno scritto X: **WT(X)** (**tempo di scrittura**)
 3. Un booleano che è vero se la transazione in (2) ha fatto commit: **c(X)** (**bit di commit**)

L'idea di base

- TS(T), RT(X), WT(X) servono a riconoscere gli schedule non serializzabili
- c(X) serve a riconoscere le letture sporche

Scarselli Franco

Sistemi per basi di dati 2006-2007

198

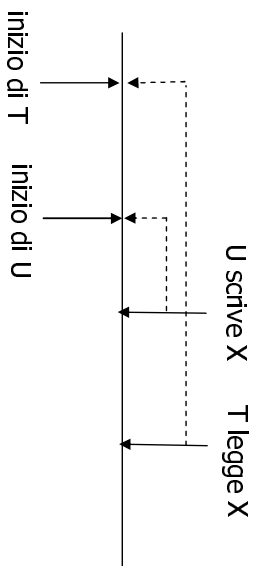
Comportamenti fisicamente non realizzabili

Si assume che tutta la transazione abbia luogo **all'istante iniziale**

I comportamenti **fisicamente non realizzabili**

- sono gli schedule che non si comportano come uno schedule seriale
- lo schedule seriale è quello in cui le transazioni vengono eseguite all'istante $TS(T)$

Lettura in ritardo

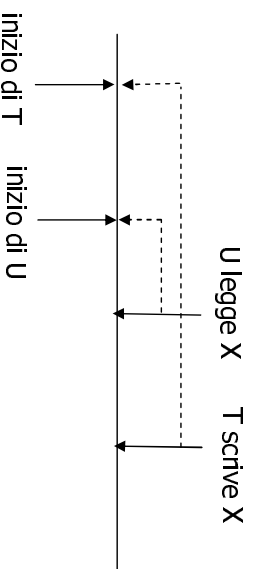


Scarselli Franco

Sistemi per basi di dati 2006-2007

199

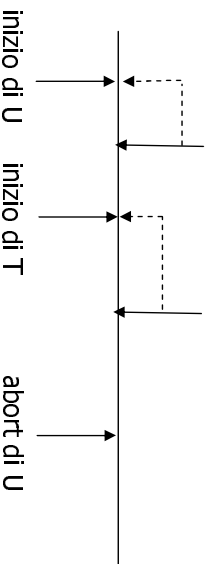
Scrittura in ritardo



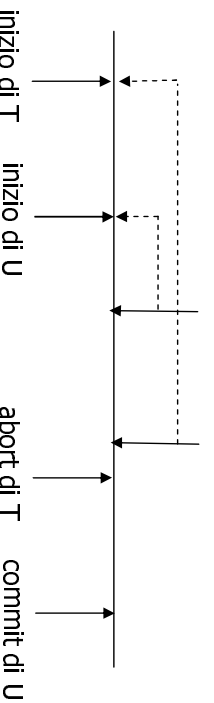
Lecture sporche

Sono problemi dovuti alla presenza di ABORT in alcune transazioni

U scrive X T legge X



U scrive X T scrive X



Una scrittura seguita da un'altra scrittura potrebbe essere non eseguita. Se la seconda scrittura viene abortita questo genera una anomalia.

Scarselli Franco

Sistemi per basi di dati 2006-2007

200



L'algoritmo

Se lo scheduler riceve una richiesta di lettura $r_T(X)$

- se $TS(T) \geq WT(X)$ la lettura è fisicamente realizzabile
 - se $c(X)$ è vero, esegui la richiesta. Aggiorna $RT(X)$
 - se $c(X)$ è falso, ritarda T fino a quando la transazione che ha scritto X abortisce o fa commit
- se $TS(T) < WT(X)$ la lettura non è fisicamente realizzabile
 - Abortisci e rilancia T



L'algoritmo II

Se lo scheduler riceve una richiesta di scrittura $w_T(X)$

- se $TS(T) \geq RT(X)$ e $TS(T) \geq WT(X)$ la scrittura è fisicamente realizzabile
 - esegui la scrittura
 - aggiorna $WT(X)$ e assegna $c(X) = \text{falso}$
- se $TS(T) \geq RT(X)$ e $TS(T) < WT(X)$ la scrittura è fisicamente realizzabile ma X contiene un nuovo valore scritto da una transazione U
 - se $c(X)$ è vero, non fare niente
 - se $c(X)$ è falso, allora occorre uccidere la transazione
- se $TS(T) < RT(X)$ la scrittura non è fisicamente realizzabile
 - Abortisci e rilancia T

L'algoritmo III

Se lo scheduler riceve una richiesta commit da una transazione T

- per tutti gli oggetti X modificati da T
 - assegna c(X)=true
 - se ci sono transazioni in attesa, attivale

Se lo scheduler riceve una richiesta abort da una transazione T

- le transazioni in attesa devono ripetere il loro tentativo di scrittura e lettura e vedere cosa succede

Un esempio

T ₁	T ₂	T ₃	A	B	C
200	150	175	RT=0 WT=0 C=true	RT=0 WT=0 C=true	RT=0 WT=0 C=true
r(B)			RT=200		
	r(A)		RT=150		
		r(C)			RT=175
w(B)				WT=200 C=false	
w(A)			WT=200 C=false		
Commit			C=true	C=true	
	w(C) -> SA				
		w(A)			
		Commit			

oggetti

Abort forzato dallo scheduler

Si va avanti senza fare niente

Confronto fra locking e controllo basato su timestamp

Pro e contro

- Il controllo basato su timestamp
 - è migliore nel caso la maggior parte delle transazioni sia in **sola lettura**
 - se ci sono molte transazioni in scrittura provoca **numerosi rollback**
- Il controllo basato su locking
 - spesso **ritarda** senza motivo l'esecuzione delle transazioni
 - è adottato dalla **maggior parte dei DBMS** commerciali
- Una soluzione **intermedia** in sistemi commerciali
 - dividere le transazioni fra quelle sola lettura e lettura e scrittura
 - quelle in sola lettura usano il controllo basato su timestamp, le altre il controllo basato su locking

Scarselli Franco

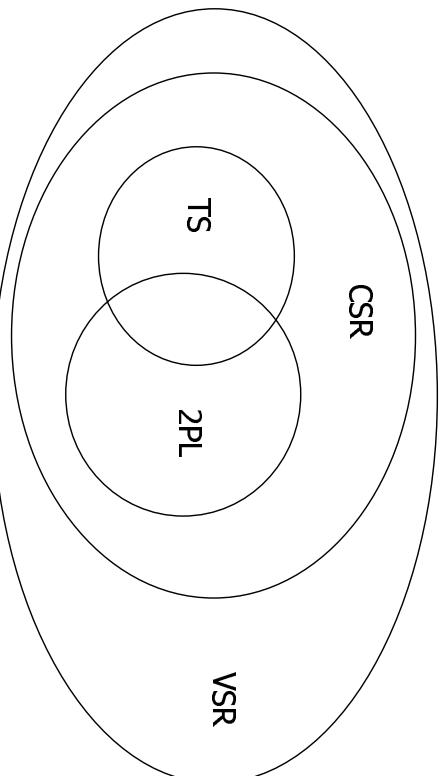
Sistemi per basi di dati 2006-2007

205

Confronto fra locking e controllo basato su timestamp II

Da un punto di vista teorico

- Il controllo basato su concorrenza genera una classe di schedule TS contenuti in CSR
- 2PL e TS hanno un'intersezione, ma non vale nessuna inclusione



Scarselli Franco

Sistemi per basi di dati 2006-2007

206



Controllo di concorrenza basato su validazione

Controllo **basato su validazione**

- E' controllo di concorrenza **ottimistico**
- Lo scheduler mantiene una lista delle operazioni di ciascuna transazione
- Subito prima di procedere al commit verifica che non esistano comportamenti non realizzabili fisicamente

L'algoritmo prevede **tre fasi** per ogni transazione

- **Lettura**: Le transazioni leggono dal database e scrivono nel loro spazio di lavoro i loro risultati
- **Validazione**: Lo scheduler valida le transazioni comparando le operazioni fatte con quelle delle altre transazioni. Se la validazione fallisce la transazione viene abortita.
- **Scrittura**: la transazione scrive i dati sul database

Scarselli Franco

Sistemi per basi di dati 2006-2007

207



La validazione

Per la validazione lo scheduler deve memorizzare **tre insieme**

- **START**: l'insieme delle transazioni che sono iniziate, ma non hanno terminato la validazione
 - **VAL**: l'insieme delle transazioni che sono state validate ma che non hanno concluso la scrittura
 - **FIN**: l'insieme delle transazioni che hanno finito la scrittura
- e **tre timestamp** per ogni transizione T
- **START(T)**: il momento in cui T è iniziata
 - **VAL(T)**: il momento in cui T è stato validato (**indica anche l'ordine seriale assunto**)
 - **FIN(T)**: il momento in cui T ha finito

I tre valori sono necessari fino a quando esiste U per la quale $START(U) \leq FIN(T)$

Scarselli Franco

Sistemi per basi di dati 2006-2007

208

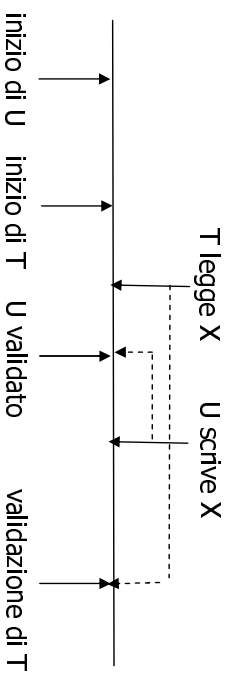
La validazione II

Si assume che tutta la transazione abbia luogo al momento della validazione

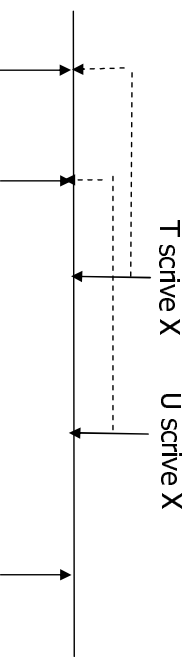
Per la validazione di T si controlla che

- $RS(T) \cap WS(U) = \emptyset$, per ogni U per il quale $FIN(U) > START(T)$
- $WS(T) \cap WS(U) = \emptyset$, per ogni U per il quale $FIN(U) > VAL(T)$

Letture/scrittura
T ha letto il valore di X prima che U finisse di scrivere



Scrittura/scrittura
T potrebbe scrivere prima di U



Scarselli Franco

U validato
Sistemi per basi di dati 2006-2007

fine di U 209

Un esempio

Validazione di U

- **ok:** nessun'altra transazione validata

Validazione di T

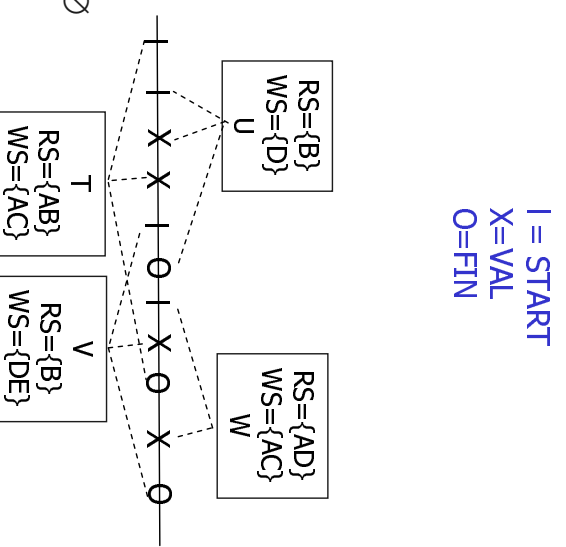
- **ok:** $WS(T) \cap WS(U) = \emptyset$, $RS(T) \cap WS(U) = \emptyset$

Validazione di V

- **ok:** $RS(V) \cap WS(U) = \emptyset$
 $WS(V) \cap WS(T) = \emptyset$, $RS(V) \cap WS(T) = \emptyset$

Validazione di W

- **no:** $RS(W) \cap WS(V) = \{D\}$, $WS(W) \cap WS(V) = \emptyset$
 $RS(W) \cap WS(T) = \{A\}$



Scarselli Franco

Sistemi per basi di dati 2006-2007

210



Confronto fra i metodi di controllo della concorrenza

Spazio occupato dai i tre metodi è simile

- **Locking:**
 - la tabella di lock e' proporzionale al numero di richieste di lock in corso
- **Timestamp:**
 - 2 timestamp per ogni oggetto acceduto dalle transazioni e 1 timestamp per ogni transazione
 - Non è sufficiente tenere conto solo delle transazioni in corso, ma servono anche quelle recenti
- **Validazione:**
 - spazio per gli insiemi RS, WS di oggetti acceduti dalle transazioni e 3 timestamp per ogni transazione
 - pero' deve tenersi in un area locale le modifiche fatte da ogni transazione

Scarselli Franco

Sistemi per basi di dati 2006-2007

211



Confronto fra i metodi di controllo della concorrenza II

Osservazioni

- Il locking ritarda le transazioni, ma evita i rollback
- Il timestamp e la validazione sono migliori in caso di bassa interferenza fra le transazioni
- quando si deve uccidere una transazione
 - il timestamp identifica il problema prima
 - la validazione aspetta il completamento delle transazioni
- nel caso di transazioni che possono causare un abort
 - la validazione permette di continuare l'esecuzione
 - il timestamp la blocca

Scarselli Franco

Sistemi per basi di dati 2006-2007

212