

Organizzazione fisica e gestione delle interrogazioni

PROF. DIOMAIUTA CRESCENZO

Tecnologia delle basi di dati

I DBMS offrono i loro servizi in modo del tutto "trasparente":

- Finora abbiamo quindi trascurato degli aspetti realizzativi
- Il DBMS è stato considerato come una black box

Aprire la black box significa

- Capire il funzionamento in modo da avere un utilizzo migliore
- alcuni servizi sono offerti separatamente

DBMS: DataBase Management System

È un sistema (prodotto software) in grado di gestire collezioni di dati che siano (anche):

- **Grandi:** di dimensioni maggiori della memoria centrale dei sistemi di calcolo utilizzati
- **Persistenti:** periodo di vita indipendente dalle singole esecuzioni dei programmi che le utilizzano
- **Condivise:** utilizzate da applicazioni diverse

Garantiscono:

- **Affidabilità:** resistenza a malfunzionamenti hardware e software
- **Privatezza:** controllo degli accessi
- **Efficiente:** deve utilizzare al meglio le risorse di in termini di spazio e tempo del sistema
- **Efficace:** produttivo per i suoi utilizzatori

DBMS: DataBase Management System

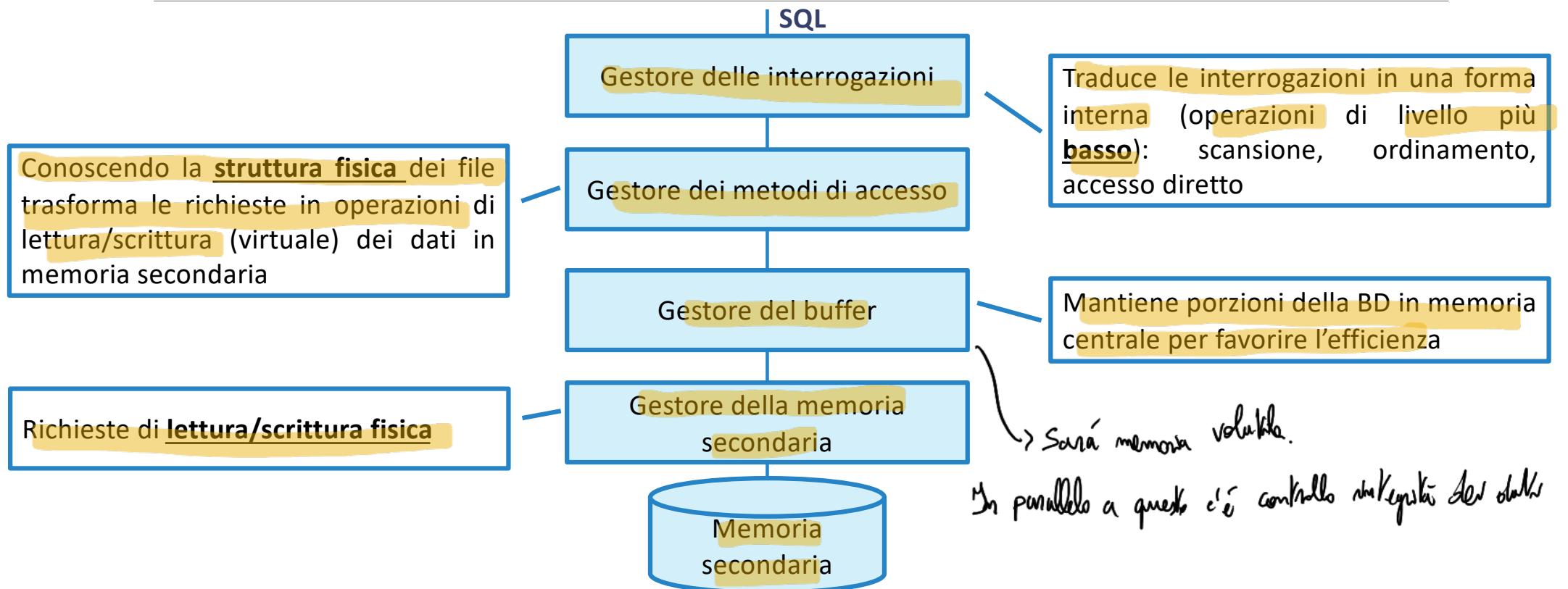
La proprietà della **persistenza** implica una gestione in **memoria secondaria**

La proprietà della **grandezza** implica una gestione sofisticata (ovvero non può essere caricato tutto in memoria principale e poi riscaricarla)

Interrogazione delle BD

- Gli utenti vedono il modello logico (*relazionale*)
- I dati si trovano in memoria secondaria
- Siccome la struttura logica non è efficiente nella memoria secondaria si utilizzano opportune strutture fisiche
- Siccome la memoria secondaria è molto più lenta della memoria principale bisogna limitare il più possibile gli accessi alla memoria secondaria

Gestore delle interrogazioni e negli accessi alla memoria secondaria



NOTA: operazioni possono anche essere simultanee.

Affidabilità delle BD

Le BD sono una risorsa per chi le possiede e le utilizza, bisogna conservarle anche in presenza di malfunzionamenti

Esempio:

- trasferimento di fondi da un conto corrente bancario ad un altro, con guasto del sistema a metà operazione
 - Le transazioni devono essere eseguite per intero o niente
 - Le transazioni dopo la loro conclusione sono definitive
- } garanzie ACID. compito del gestore dell'affidabilità

Garantire l'affidabilità nelle BD è impegnativo, a causa dei frequenti aggiornamenti e a causa della gestione del buffer

Condivisione delle BD

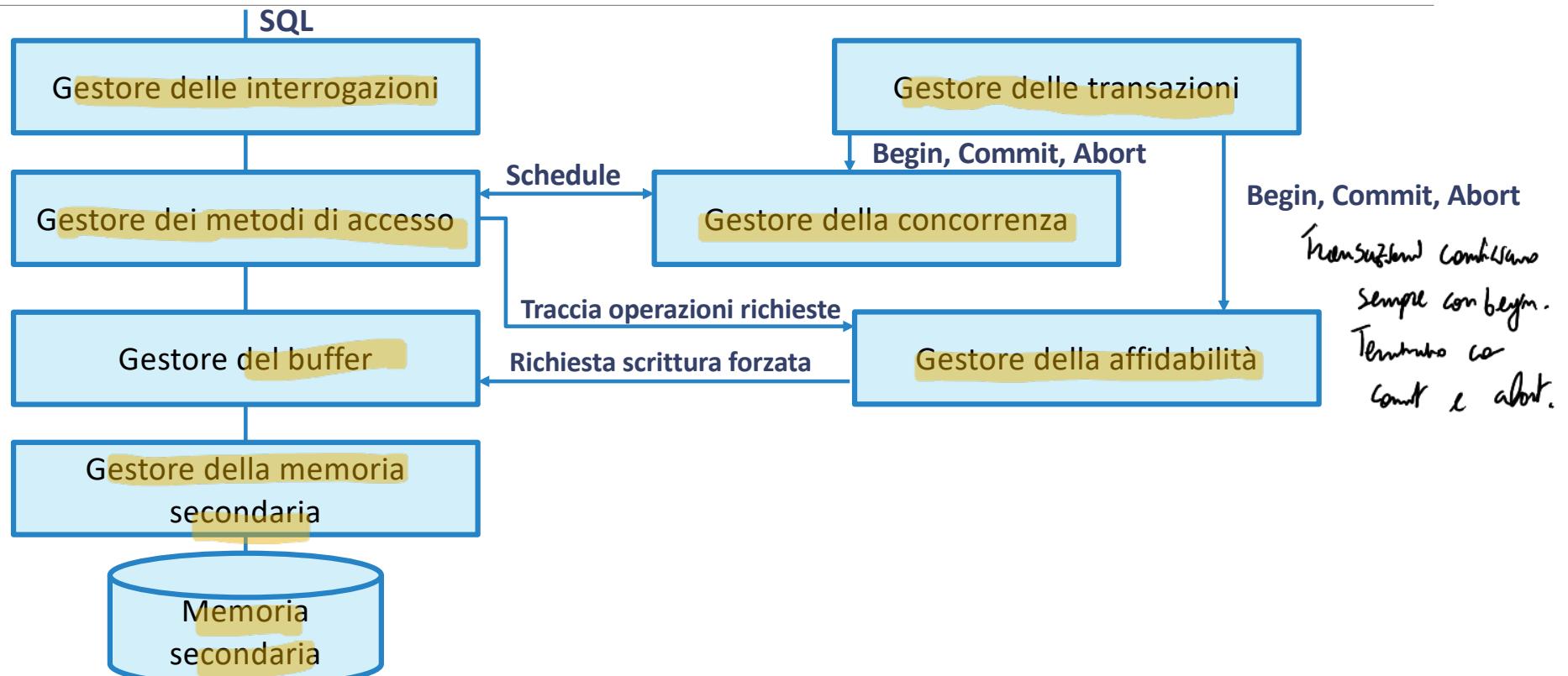
Una base di dati essendo una risorsa **integrata e condivisa** tra le diverse applicazioni utente che la utilizzano si ha che:

- Sono presenti attività diverse su dati in parte condivisi:
 - Necessità di meccanismi di **autorizzazione**
- Sono presenti attività multi - utente su dati condivisi:
 - Necessità di meccanismi di controllo della **concorrenza**

Operazioni su BD condivise

- Vediamo due esempi di operazioni condivise sulla BD:
 - due prelevamenti (quasi) contemporanei sullo stesso conto corrente
 - due prenotazioni (quasi) contemporanee di un posto su un treno
- Intuitivamente possiamo pensare che le transazioni sono corrette se eseguite in maniera **seriale**
- In molti sistemi reali la serialità delle transazioni penalizza troppo l'efficienza:
 - Bisogna utilizzare quindi un meccanismo di **controllo della concorrenza**

Gestore degli accessi e delle interrogazioni – Gestore delle transazioni



Memoria principale, secondaria e gestione dei buffer

I programmi fanno riferimento solo a dati presenti in **memoria principale**

Le BD non possono essere contenute nelle memorie principali e quindi bisogna inserirle nella memoria secondaria per i seguenti motivi:

- **dimensioni**
- **persistenza**

I dati presenti in memoria secondaria per poter essere utilizzati, devono prima essere trasferiti in memoria principale

Memoria principale, secondaria e gestione dei buffer

- I dispositivi di memoria secondaria sono organizzati in **blocchi** di dimensione di solito **fissa** (ordine di grandezza: alcuni KB) (4-8 kB)
- Le uniche operazioni possibili su un disco sono la **lettura** e la **scrittura** di un intero blocco (**pagina**), cioè di una stringa di byte;
- Inoltre se consideriamo i tempi di accesso in una memoria secondaria (si ha non meno di 16 ms):
 - tempo di posizionamento della testina (10-50ms)
 - tempo di latenza (5-10ms)
 - tempo di trasferimento (1-2ms)
- Il costo di un accesso a memoria secondaria è vari ordini di grandezza **maggiori** di quello per operazioni in memoria centrale
- Inoltre, accessi a blocchi adiacenti riduce il costo, poiché si ha che il tempo di posizionamento e di latenza sono nulli (**contiguità**)



Gestione dei buffer

L'interazione tra memoria centrale e memoria secondaria è realizzata nei DBMS attraverso una grande zona di memoria centrale detta **buffer**:

- area di memoria centrale, gestita dal DBMS (pre-allocata) e condivisa fra le transazioni
 - organizzato in **pagine** di dimensioni pari o multiple di quelle dei blocchi di memoria secondaria (1KB-100KB)
 - è importantissimo per via della grande differenza di tempo di accesso fra memoria centrale e memoria secondaria
 - Si occupa del **caricamento/scaricamento** delle pagine dalla memoria centrale alla memoria di massa
- Se aggiorno in memoria centrale devi aggiornare secondaria

Gestione dei buffer

Lo scopo del gestore del buffer è di ridurre il numero di accessi alla memoria secondaria

- In caso di lettura, se la pagina è già presente nel buffer, non è necessario accedere alla memoria secondaria
→ Sincrona (sullo) o asincrona?
- In caso di scrittura, il gestore del buffer può decidere di differire la scrittura fisica (ammesso che ciò sia compatibile con la gestione dell'affidabilità – vedremo più avanti)

Le politiche di gestione del buffer somigliano a quelle di gestione della memoria principale da parte dei sistemi operativi, obbedisce

- Al principio della località dei dati: è alta la probabilità di dover riutilizzare i dati attualmente in uso
- Legge empirica “80-20”: l'80% delle operazioni utilizza sempre lo stesso 20% dei dati

Gestione dei buffer

Il gestore del buffer non si limita solo a lettura/scrittura, ma anche ad informazioni sul riutilizzo delle pagine e sulla necessità di aggiornare immediatamente i dati in esso contenuti, è un modulo che possiede quindi:

- Un direttorio che per ogni pagina mantiene il file fisico e il numero del blocco corrispondente
- Per ogni pagina mantiene due variabili di stato:
 - un contatore che indica quanti programmi utilizzano la pagina
 - un bit di stato che indica se la pagina è stata modificata

Funzioni del gestore del buffer

Il gestore del buffer intuitivamente:

- Riceve richieste di lettura/scrittura di pagine
- Le esegue accedendo alla memoria secondaria solo se necessario, altrimenti utilizza il buffer (*Cerca prima in memoria principale, poi secondaria*)
- Esegue le primitive fix, unfix, setDirty e force
 - ↳ da basso livello

Funzioni del gestore del buffer

- **fix:** usata dalle transazioni per la richiesta di una pagina; richiede una lettura solo se la pagina non è nel buffer (incrementa il contatore associato alla pagina) *↳ lettura dalla memoria secondaria*
- **setDirty:** comunica al gestore del buffer che una pagina è stata modificata, modificando il relativo bit di stato (*la sua stat. Dovrò aggiornarla nella memoria secondaria*)
- **unfix:** indica che la transazione ha concluso l'utilizzo della pagina, viene decrementato il contatore di utilizzo della pagina
- **force:** trasferisce in modo sincrono una pagina dal buffer in memoria secondaria (su richiesta del gestore dell'affidabilità, non del gestore degli accessi) *quindi appena modifica scrivo lo modifico.*

Esecuzione della primitiva fix

- Si cerca la pagina nel buffer:
 - se è presente, restituisce l'indirizzo della pagina alla transazione richiedente
 - altrimenti, cerca una pagina libera nel buffer, ovvero con contatore a zero (secondo i criteri LRU, FIFO); *usata meno recentemente ecc. Ciononostante è ancora indirizzabile*
 - se la trova, restituisce l'indirizzo
 - Altrimenti (non esistono pagine libere), il buffer manager applica due alternative:
 - "steal": viene sottratta una pagina ad un'altra transazione, la pagina è detta "vittima", viene scritta in memoria secondaria (**flush**); viene letta la pagina di interesse dalla memoria secondaria e si restituisce l'indirizzo
 - "no-steal": la transazione viene sospesa in attesa della liberazione di pagine dal buffer

N.B.: Ogni accesso alla pagina implica l'incremento del relativo contatore

Modalità di scrittura in memoria secondaria

La scrittura in memoria secondaria può avvenire in due modalità differenti:

- in modo **sincrono** quando è richiesto esplicitamente dall'applicazione con una **force**
- in modo **asincrono** quando lo ritiene opportuno (o necessario); in particolare, può decidere di anticipare o posticipare scritture per coordinarle e/o sfruttare la disponibilità dei dispositivi

DBMS e File system

I DBMS utilizzano **alcune** delle funzionalità del File System, che è un modulo del Sistema Operativo che gestisce la memoria secondaria:

- per **creare ed eliminare file**
- per **leggere e scrivere singoli blocchi o sequenze di blocchi contigui.**

L'**organizzazione dei file**, sia relativamente alla struttura dei file e sia in termini di distribuzione dei record nei blocchi sia relativamente alla struttura all'interno dei singoli blocchi è **gestita direttamente dal DBMS**.

Pagine: moltissimo poco rispetto alla memoria, ma puntiamo con moltissimo logo da trattare

DBMS e File system

Il DBMS, in molti casi, **crea file di grandi dimensioni che utilizza per memorizzare diverse relazioni** (al limite, l'intera base di dati)

In **altri casi, vengono creati file in tempi successivi**:

➤ è possibile che un file contenga i dati di più relazioni e che le varie tuple di una relazione **siano in file diversi**.

In sostanza, il **DBMS gestisce i blocchi dei file allocati come se fossero un unico grande spazio di memoria secondaria e costruisce**, in tale spazio, le **strutture fisiche con cui implementa le relazioni**.

Nella maggior parte dei casi, **ogni blocco è dedicato a tuple di un'unica relazione**

Blocchi dei file a cui vale questo, così anche blocchi contigui significano che ho unica tabella.

Gestione delle tuple nelle pagine

In ogni pagina sono presenti informazioni utili e informazioni di controllo

- Utili: sono i dati veri e propri
- Controllo: consente di accedere all'informazione utile

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

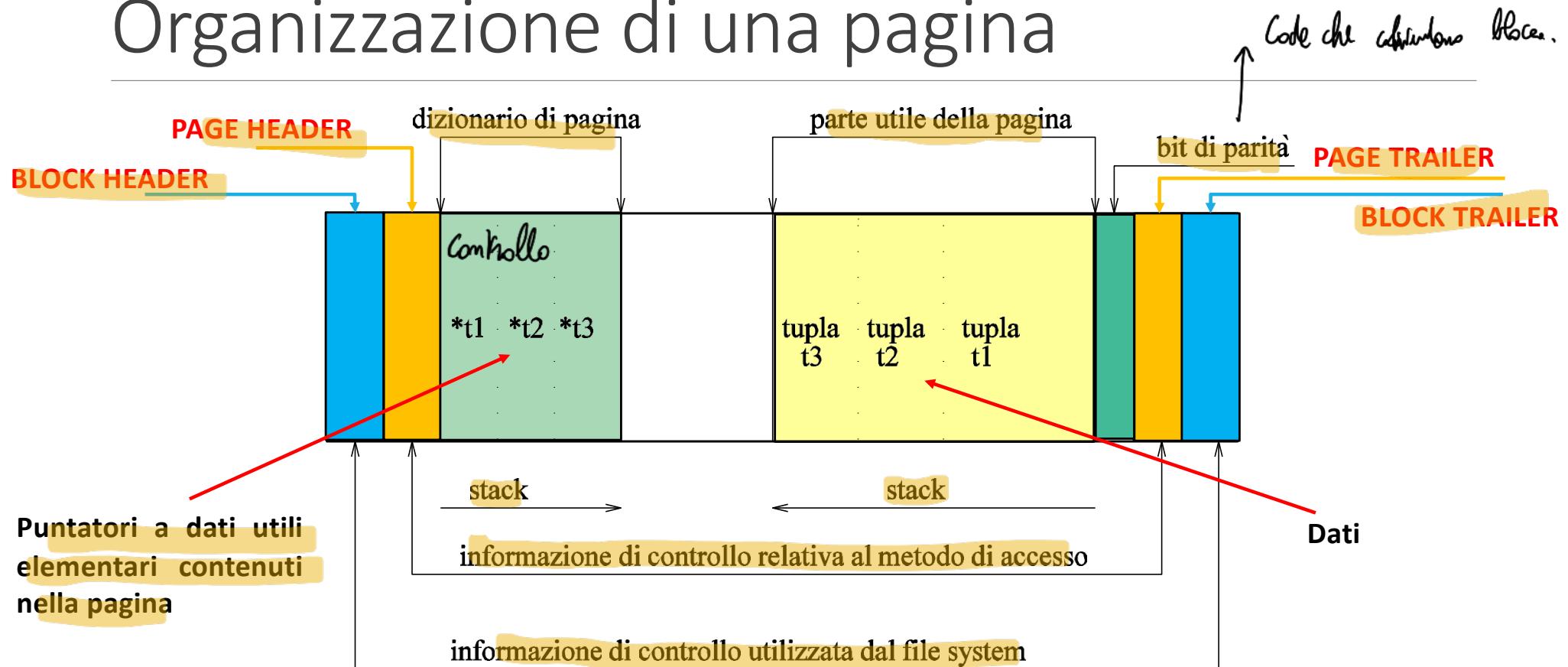
Inoltre:

- In alcuni casi la massima dimensione di una tupla è limitata al massimo spazio utile di una pagina
→ se ho tante colonne, per esempio
- In alcuni sistemi possono spezzare le tuple su più pagine (necessario per tuple grandi)
- In alcuni casi tutte le tuple hanno la stessa dimensione, semplificando così la struttura dei dizionari di pagina

↓
es. var char(50)

Yours: parte dell'interazione del blocco e della pagina. Dovendo chi pagina mantenere corrispondente con le tuple. So, gradi di puntatori dove sono i due utile della pagina.

Organizzazione di una pagina



Memorizzazione dei dati ed organizzazione dei file

- La struttura tipica dei dati per la memorizzazione delle informazioni in una BD è quella dei file di record
- L'unità di informazione letta/scritta dalla memoria di massa è detta pagina (tipicamente 4 o 8 KB)
- Dal punto di vista logico, un file è quindi una sequenza di record, dove ogni record è formato da uno o più campi identificati da un unico identificatore, che consente di identificare l'indirizzo fisico della pagina

Fattore di blocco

ID	Nome	Cognome	Stipendio
003	Antonio	De Simone	5000
001	Marco	Rossi	2000
005	Antonio	Di Girolamo	3000
004	Francesco	Di Caterino	1000

□ Assumiamo che ogni tupla in questa relazione sia mappata in un record di un file gestito dal sistema operativo.

□ Quando un utente richiede una tupla dal DBMS, quest'ultimo mappa un record logico in un record fisico, e lo associa ad una pagina in memoria primaria.

Il record fisico consiste di uno o più record logici ed il numero di record logici contenuti in un record fisico è detto fattore di blocco.

Quanti blocchi logici verso a mappare in un blocco fisico?

Blocco logico è riferito alla memoria principale, blocco fisico alla secondaria

Gestione delle tuple nelle pagine

I blocchi (componenti "fisici" di un file) e i record (componenti "logici") hanno dimensioni in generale diverse:

- la dimensione del blocco dipende dal file system
- la dimensione del record dipende dalle esigenze dell'applicazione, e può anche variare nell'ambito di un file

Il **fattore di blocco**: è il numero di record contenuti in un blocco

- L_R : dimensione di un record (per semplicità costante nel file: "record a lunghezza fissa")
- L_B : dimensione di un blocco
- se $L_B > L_R$, possiamo avere più record in un blocco:

$$\text{Fattore di blocco} = \frac{L_B}{L_R}$$

Sia su memoria principale che su
memoria secondaria

Primitive del gestore delle pagine

- Inserimento e aggiornamento:** comportano una **riorganizzazione** o meno delle pagine a seconda dello spazio disponibile
- Cancellazione:** viene effettuata sempre, senza riorganizzazione delle informazioni nella pagina
- Accesso ad una particolare tupla:** identificata tramite il valore della **chiave** (non necessariamente uso chiave primaria, se devo accedere a studente con cognome "A", vedo quello)
- Accesso ad un campo di una particolare tupla:** identificato in base all'**offset** (distanza di ciascuna tupla rispetto all'inizio della **parte utile**)

Organizzazione di file (che memorizzano fisicamente le mostre la tabella)

Esiste un criterio secondo il quale le tuple sono ordinate nell'ambito del file ai fini della gestione

- **Sequenziali**: disposizione consecutiva delle tuple in memoria di massa
- **Ad accesso calcolato (hash)**: tuple disposte in posizioni determinate dal risultato dell'esecuzione di un algoritmo
- **Ad albero**: posizionamento mediante l'uso di indici

Organizzazione di file

L'ordine con cui i record sono **memorizzati** e **acceduti** nel file dipende dalla struttura dei file in memoria secondaria. Esistono diverse modalità di organizzazione:

- **File non ordinati** (o file heap)
- **File ordinati** (o file sequenziali)
- **File ad accesso calcolato** (o hash file)
- **Ad albero**

File HEAP

Memorizzazione del record su file in modo del tutto **casuale**.

- Un **nuovo record** che viene inserito nel file, viene posizionato **nell'ultima pagina del file**
 - > se lo spazio è **insufficiente**, allora viene creata una nuova pagina e viene aggiunta in coda al file stesso
- le operazioni di **inserimento** nel file sono molto efficienti.
- L'unica possibilità di recupero dei record è quella basata su una **ricerca lineare**
- L'operazione di **cancellazione** del record richiede, al solito, la **ricerca nel file** del record da cancellare
 - > il record viene marcato come logicamente cancellato e non viene più usato
 - > **deterioramento progressivo** del file quando ci sono frequenti cancellazioni; necessità di **riorganizzazione** del file per recuperare gli spazi lasciati inutilizzati dalle cancellazioni.

File ORDINATI

I record di un file sono ordinati su uno o più campi

- L'operazione di ricerca di un record può fare uso di tecniche di **ricerca binaria** più efficiente della **ricerca lineare**.
- Le operazioni di inserimento e di cancellazione richiedono maggiori passi computazionali per mantenere l'ordine tra i record.
 - > L'inserimento richiede prima l'individuazione della posizione in cui inserire il record, quindi occorre fare spazio e solo a questo punto è possibile fare inserimenti. Se non c'è spazio nella pagina, occorrerà creare una nuova pagina e spostare uno o più record in essa.
 - > E' utile utilizzare un file temporaneo in cui effettuare inserimenti non ordinati ed opportuni. **algoritmi di merge-sort** possono essere usati ad intervalli regolari di tempo per riottenere un unico file ordinato.
 - > Per la cancellazione, occorrerà riorganizzare i file per rimuovere lo spazio lasciato libero.

NOTA: Con blocco faremo riferimento sempre alla stessa relazione. Faranno riferimento a chiave oppure su un campo particolare.

File HASH

La posizione del record nel file viene calcolata attraverso un'apposita funzione detta funzione di hash

- Tale funzione riceve in ingresso uno o più campi del record e restituisce una posizione. Se i campi di hash sono anche chiavi del file, si parla di **chiavi di hash**.
- La funzione di hash deve essere scelta in modo tale da distribuire i record nel file in modo uniforme, anche se apparentemente casuale.
 - > Una tecnica usuale che implementa la funzione di hash consiste nel convertire i valori delle chiavi di hash in numeri interi (esempio posizione alfabetica oppure codice ASCII) e nel sommare il numero ottenuto ad un opportuno offset (**tecnica di folding**) oppure effettuando una divisione in modulo per individuare la posizione (**tecnica della divisione in modulo**).

Dovendo un problema di classificazione, con dei buckets da cui vennero le mosse informazioni

Collisioni nei File di HASH

Non si può garantire ad ogni record un indirizzo univoco, poiché il numero dei possibili valori calcolati dei campi di hash è molto maggiore del numero di posizioni disponibile nel file.

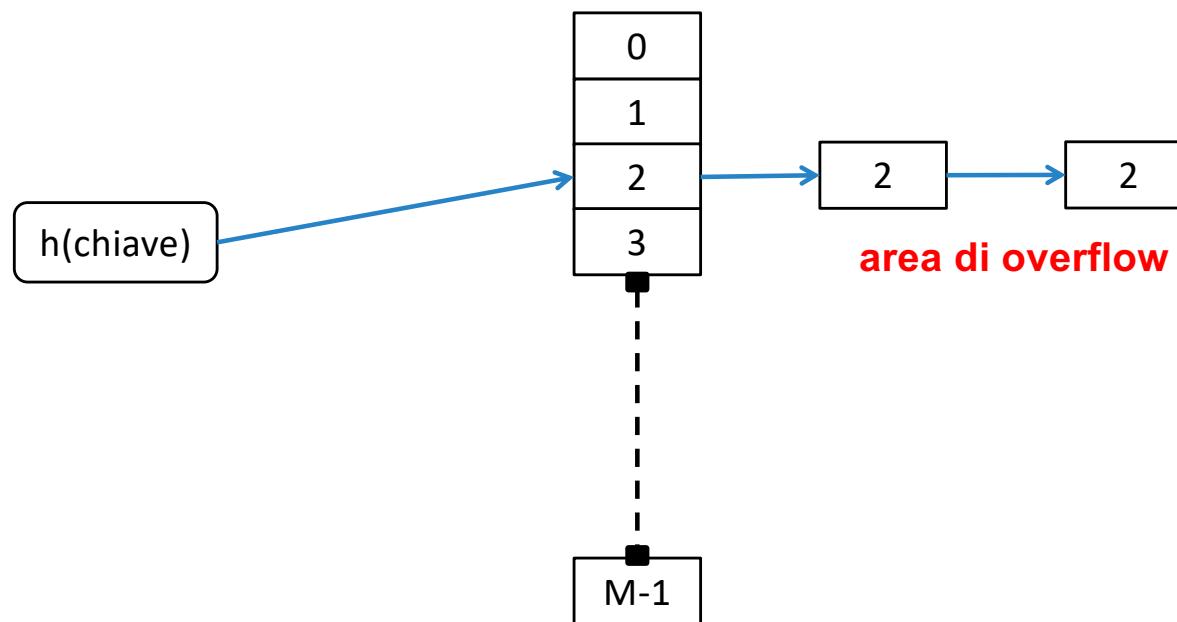
- Si associa, tipicamente, ad ogni indirizzo generato dalla funzione di hash un **bucket**, su memoria di massa contenente più record.
- All'interno del bucket i record sono memorizzati secondo l'ordine di arrivo.

In caso di record sinonimi (con lo stesso indirizzo di bucket calcolato dalla funzione hash) si parla di **collisioni**; in presenza di collisioni, se il bucket è **pieno** occorre inserire il record in una nuova posizione.

- Uso di un'area di **overflow**, in cui posizionare record **sinonimi**, ponendo i record in quest'area in modo libero (secondo l'ordine di arrivo) oppure collegandoli ad una lista circolare collegata ai vari bucket.

Dove si verificano collisioni crea area di overflow.

Collisioni nei File di HASH



Collisioni nei File di HASH

Esempio: funzione hash che calcola il resto della divisione per 50

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

Sequenza di arrivo

Matricola Matr. mod 50

060600	0
066301	1
205751	1
205802	2
200902	2
116202	2
200604	4
066005	5
116455	5
200205	5
201159	9
205610	10
201260	10
102360	10
205460	10
205912	12
205762	12
200464	14
205617	17
205667	17

Matricola Matr. mod 50

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

Scelta di Funzioni hash

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

Un file hash

Entra Matricola -> calcolo Matricola mod 5

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

file hash con fattore di blocco 10 (quante tuple entrano in un blocco)
5 blocchi con 10 posizioni

Indici di Accesso (Strutture ad albero)

- Un indice è una struttura dati che permette di organizzare in modo opportuno i record al fine di rendere **efficiente** il recupero dell'informazione, attraverso una **chiave di ricerca sull'indice**.

Indici di Accesso (Strutture ad albero)

Un **indice** è una **struttura dati ordinata** che consente di localizzare un particolare record in un file dati in modo veloce, diminuendo tempi di risposta di una query.

- Gli indici migliorano le prestazioni di un DBMS.
- Ad ogni file (anche non ordinato) viene associato **un file di indice** contenente la struttura dati dell'indice stesso
 - formata da coppie **chiave di ricerca e identificatore di un record**.
- Un indice **velocizza** le selezioni sui campi che compongono la chiave di ricerca per l'indice

Esempio

Possiamo memorizzare le informazioni in un file non ordinato mentre le posizioni si possono registrare in un file ordinato sul campo ID. Servono due file:

- Uno (disordinato) contenente i record dati identificati ognuno dal proprio identificatore (record identifier, rid);
- Uno ordinato contenente i record dell'indice ordinati sul campo **ID**
- Nel caso in cui si vogliano fare query sul campo stipendio, può essere utile aggiungere un altro file di indice, ordinato su stipendio (file indice ausiliario o secondario).

Esempio

ID	Nome	Cognome	Stipendio
003	Antonio	De Aurelio	5000
001	Marco	Marchi	2000
005	Antonio	Di Paola	3000
004	Francesco	Di Salet	1000

reference ad altri membri relativi
ma ordinato e non

ID	rid	rid	ID, Nome, Cognome, Stipendio
001	2	1	003, Antonio, De Aurelio, 5000
003	1	2	001, Marco, Marchi, 2000
004	4	3	005, Antonio, Di Paola, 3000
005	3	4	004, Francesco, Di Salet, 1000

indice

file di record

Data Entry

Il **data entry** è il **record** memorizzato in un file indice: **un indice è dunque una collezione di data entry**.

- Il **data entry** è costituito da un **intero record di dati**, con la sua **chiave di ricerca**: in questo caso, l'indice è un caso particolare di organizzazione di file;
- Il **data entry** è una coppia formata da (**chiave, rid**), come nell'esempio precedente;
 - In questo caso l'indice è completamente indipendente dall'organizzazione del file con i dati (**heap o ordinato**);
- Il **data entry** è una coppia (**chiave, lista di rid**), dove **lista di rid** è una lista di identificatori di record dati aventi un particolare valore della chiave di ricerca;
 - Anche in questo caso l'indice è indipendente dall'organizzazione del file, permettendo una migliore gestione dello spazio.

Tipi di Indice

Esistono diversi tipi di indici:

- ❖ **indice primario**: indice costruito su un file sequenziale ordinato su una chiave – (è un indice su di un insieme di campi che include la chiave primaria di una relazione)
- ❖ **indice secondario**: si tratta di un indice costruito su una chiave non primaria di una relazione (Su un attributo non chiave)
- ❖ **indice clustering**: si tratta di un indice costruito su un campo non chiave, di modo che ad ogni valore di tale campo corrispondono più record (**cluster di records**) (non ho molti record su molte righe ma ne posso avere da più)

[Sarà nella lista 8a]

Su un campo chiave più record (più strati).

(Record di memorizzazione)

Tipi di Indice

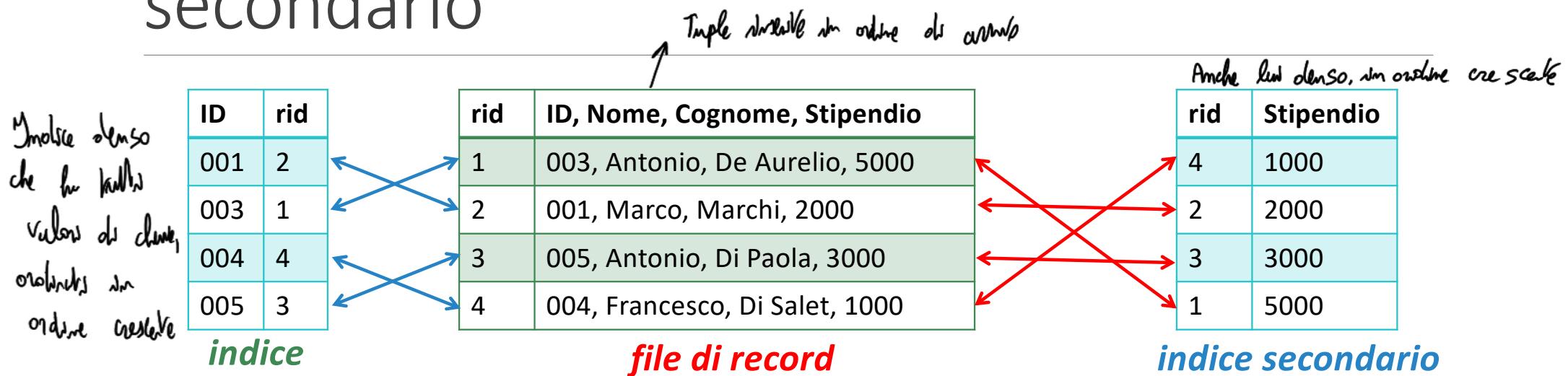
Un file può avere:

- ❖ **Un solo indice primario** Ho una sola chiave primaria
- ❖ **Un solo indice di clustering** (l'hw è opzionale)
- ❖ **Diversi indici secondari** (0...N)

Gli indici, inoltre, si distinguono inoltre in

- ✓ **indice sparso**: indice che ha un record di indice solamente per alcuni valori della chiave nel file
- ✓ **indice denso**: indice che ha un record di indice per ogni valore di chiave di ricerca nel file.

File di record di dati e di indice primario e secondario



Indice **primario**: su un campo sul cui ordinamento è basata la memorizzazione

Indice **secondario**: su un campo con ordinamento diverso da quello di memorizzazione

Struttura molto utilizzata: file ISAM

File Sequenziali Indicizzati (Indexed Sequential Files)

↳ file ordinato con indice definito su primary key.

Un file sequenziale indicizzato è un file ordinato con un indice primario.

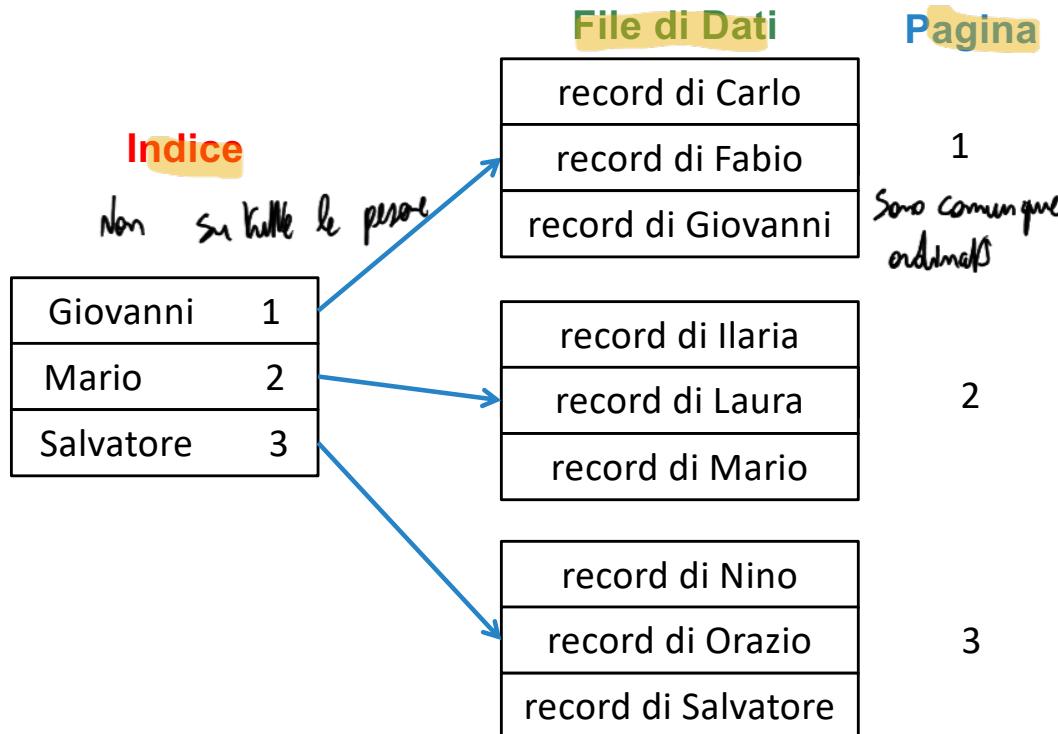
Il più famoso indice è quello definito da IBM con la struttura detta **ISAM** (Indexed Sequential Access Method), indice fortemente dipendente dall'hardware dello storage, la cui evoluzione hardware-independent ha dato luogo al Virtual Sequential Access Method (**VSAM**).

↳ usava memoria virtuale come sistema operativo

Un file sequenziale indicizzato è di solito organizzato in un'area di memorizzazione primaria, in uno o più indici separati, e in una area di overflow.

(indici separati & seconda livello x es)

Esempio di Indice Sparso



➤ Un file sequenziale indicizzato è organizzato, di solito, in un'area di memoria primaria, in uno o più indici separati e in un'area di workflow.

➤ Parte dell'indice primario è memorizzato in memoria principale, essendo ordinato, si può sfruttare per le ricerche algoritmi di ricerca binaria.

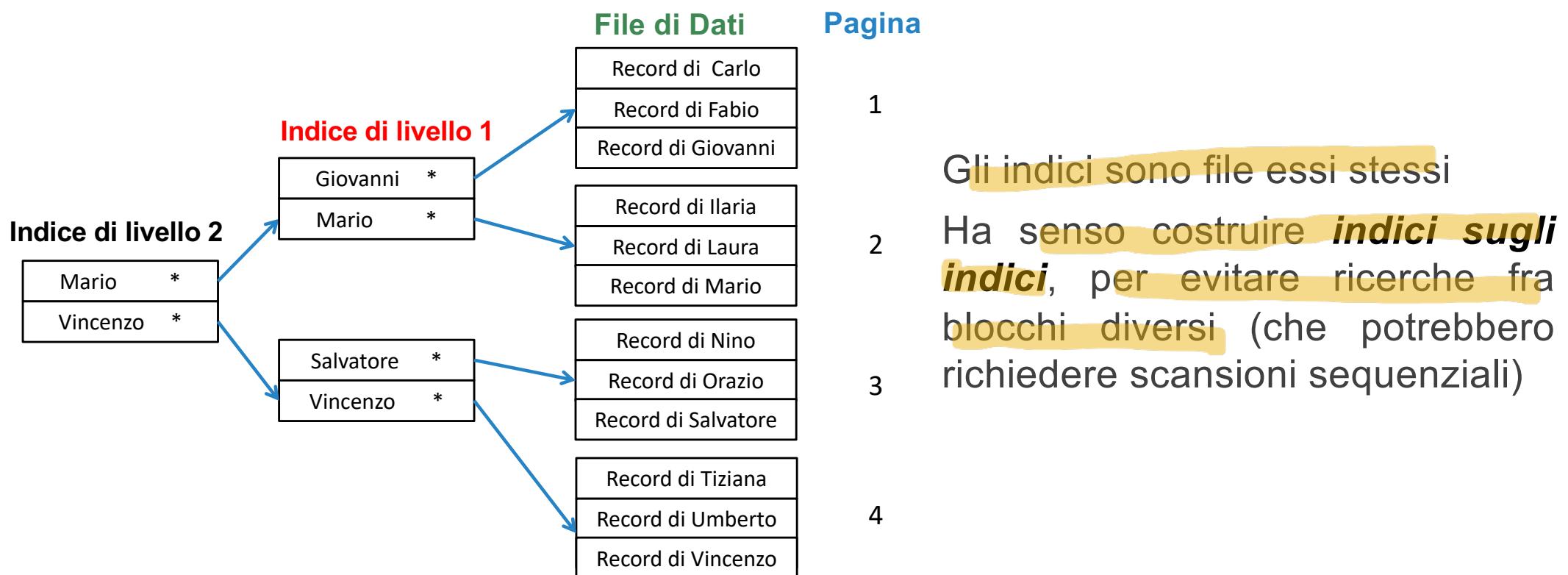
➤ Le continue operazioni di inserimento e di cancellazione rendono complesso il mantenimento dell'ordine nel file di dati e nel file di indice.

(posso uscire frammentazione)

Tutto ciò richiede ordinamento anche nel file di dati

Indici Multilivello

: cioè struttura di strutture, primi livelli sono struttura ad albero

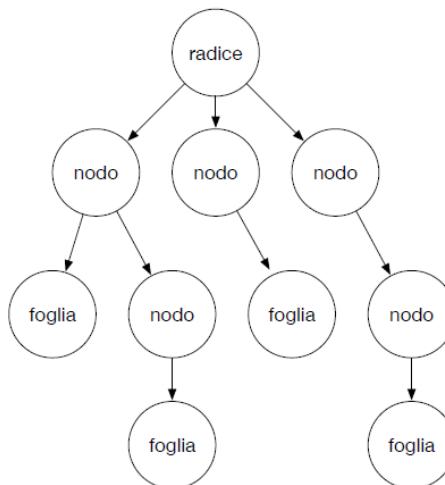


Indici B⁺ Tree

L'albero è una struttura dati gerarchica formata da nodi e archi.

Ogni nodo, eccetto il nodo radice, ha un unico nodo detto padre e zero o più nodi figli.

Un nodo che non ha nessun nodo figlio è detto anche nodo foglia



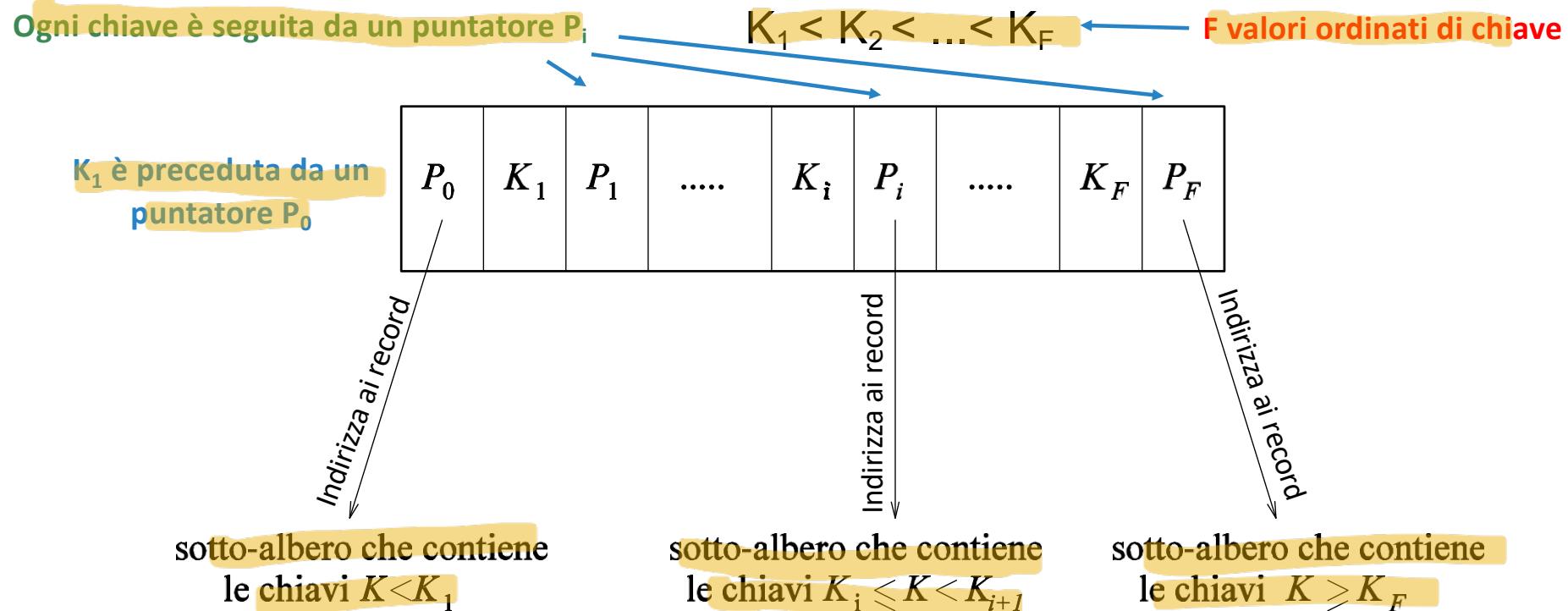
È la struttura dati più usata per gestire gli indici nei DBMS

Balanced Tree (o B-Tree)

- ❖ Un albero è detto **bilanciato** se la sua profondità (il massimo numero di livelli tra il nodo radice e una foglia) è la stessa per ogni nodo foglia
- ❖ In generale, mantenere un albero bilanciato è molto importante ai fini dell'efficienza di una ricerca
 - Ciò garantisce che nessun nodo si troverà a profondità troppo alta rispetto agli altri, richiedendo così molti accessi ai blocchi durante la ricerca stessa.
- ❖ È necessario avere algoritmi che, in fase di inserimento o di cancellazione di un nuovo record, aggiornano l'albero per mantenerlo bilanciato.

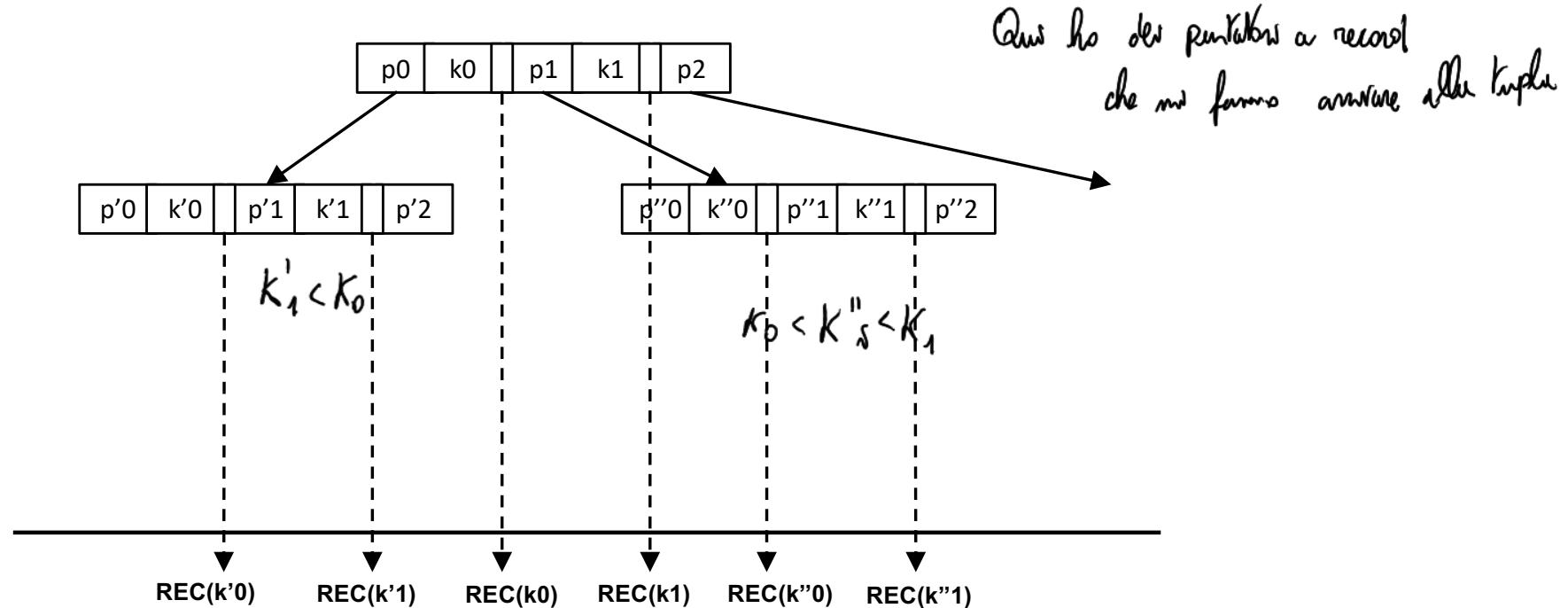
Valori di chiave ordinati in ordine crescente e ogni chiave è seguita da un puntatore. Chiave: punto per accedere al record. Puntatore è per navigare allbero.

Informazione contenuta in un nodo (pagina) di un albero B-tree



II B-Tree

- ❖ Un B-Tree di ordine p è definito nel seguente modo:



Definizione di un B-Tree

P è il numero dei puntatori
q è il numero di chiavi punz.

- ❖ Ogni nodo dell'albero contiene $\langle P_1, K_1, P_{K1}, P_2, K_2, P_{K2} \dots P_q \rangle$, con $q \leq p$, che sono i puntatori ai dati relativi alla chiave K_i , P_i è il puntatore ad un sottoalbero che contiene tutti i valori di chiave $K \leq K_i$, P_q punterà ad un sottoalbero con valori di chiavi $K > K_i$
- ❖ Ogni nodo può contenere al massimo p puntatori dell'albero;
- ❖ Ogni nodo, tranne la radice e i nodi foglia, deve contenere almeno $p/2$ puntatori dell'albero e il nodo radice contiene almeno 2 nodi figli;
- ❖ Tutti i nodi foglia sono allo stesso livello.

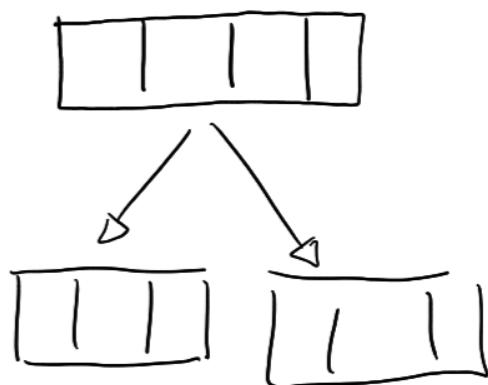
(radice è sempre un file (insieme di tutti gli alberi))

Costruzione del B-Tree (I)

- ❖ Un albero B inizia con un solo nodo radice a livello 0.
- ❖ Quando il nodo radice diventa completo, il nodo si divide (operazione di **split**) in due nodi di livello 1: il nodo radice conterrà solo il valore centrale del nodo, mentre il resto dei valori sarà equamente diviso tra i due nodi figli.
- ❖ Quando uno dei due nodi si riempie, tale nodo viene diviso in due nodi dello stesso livello e viene posto un puntatore al nodo padre ponendo in esso i valori di chiave centrale.
- ❖ Quando il nodo padre si riempie, viene diviso pure esso: se la divisione si propaga fino al nodo radice, e se anche la radice deve essere divisa, si crea un nuovo livello dell'albero.

Costruzione del B-Tree (II)

- ❖ Riguardo alla cancellazione, se la cancellazione di un valore fa sì che il nodo sia completo per meno della metà, allora il nodo viene fuso (operazione di **merge**) con uno dei suoi vicini: si noti che anche la cancellazione può propagarsi fino alla radice.



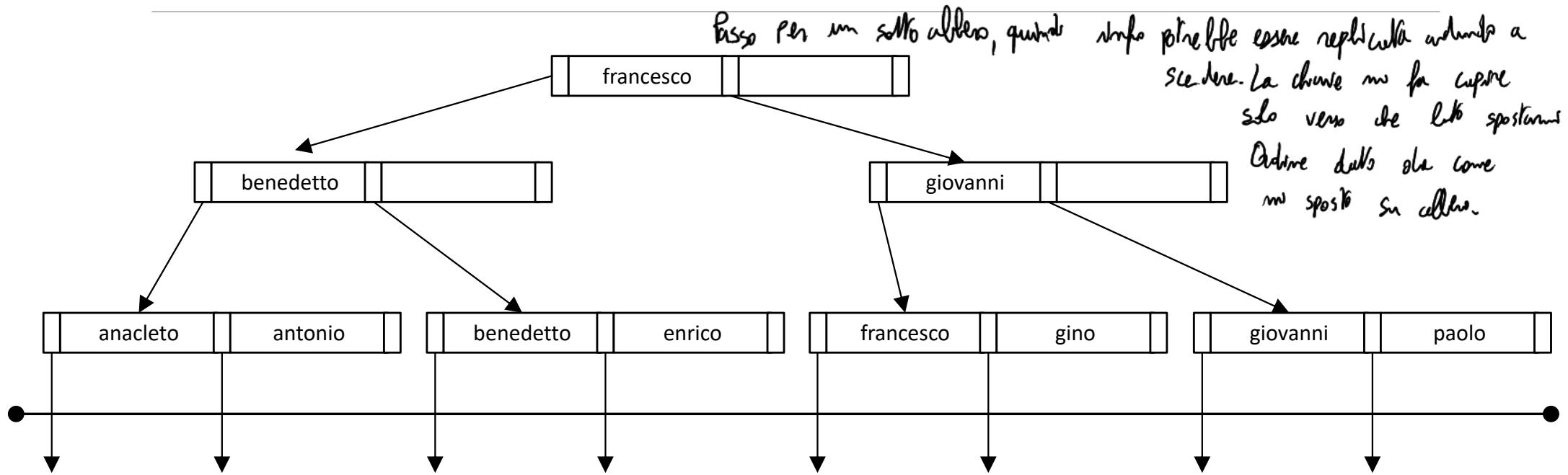
B⁺Tree

La maggior parte delle implementazioni commerciali utilizza una variante dell'albero B detta B⁺Tree.

- ❖ Nell'albero B⁺Tree, i puntatori ai dati sono memorizzati esclusivamente nei nodi foglia dell'albero. (non ho lista collegata)
- ❖ Tali nodi foglia, inoltre, sono collegati tra di loro (lista collegata).
- ❖ Un B⁺Tree richiede approssimativamente lo stesso tempo per accedere ad un qualunque record da esso indicizzato, e quindi assicura che vengono attraversati circa lo stesso numero di nodi prima di arrivare alla radice (il tempo di ricerca è dunque funzione della profondità dell'albero).

< >

Esempio di B⁺Tree

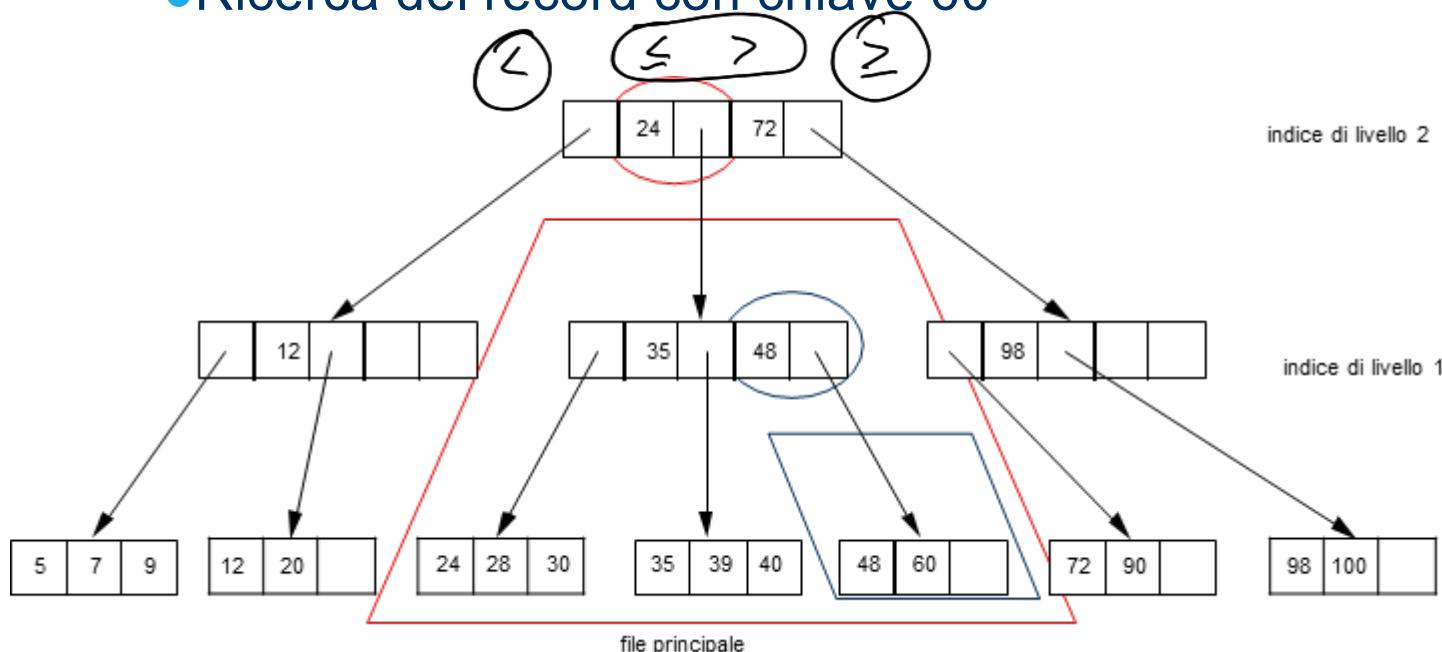


Puntatori ai dati non ordinati

Questa organizzazione può non essere totale, a differenza del prima (se x es. rimuovo valori da radice)

B-tree: Esempio di ricerca

• Ricerca del record con chiave 60

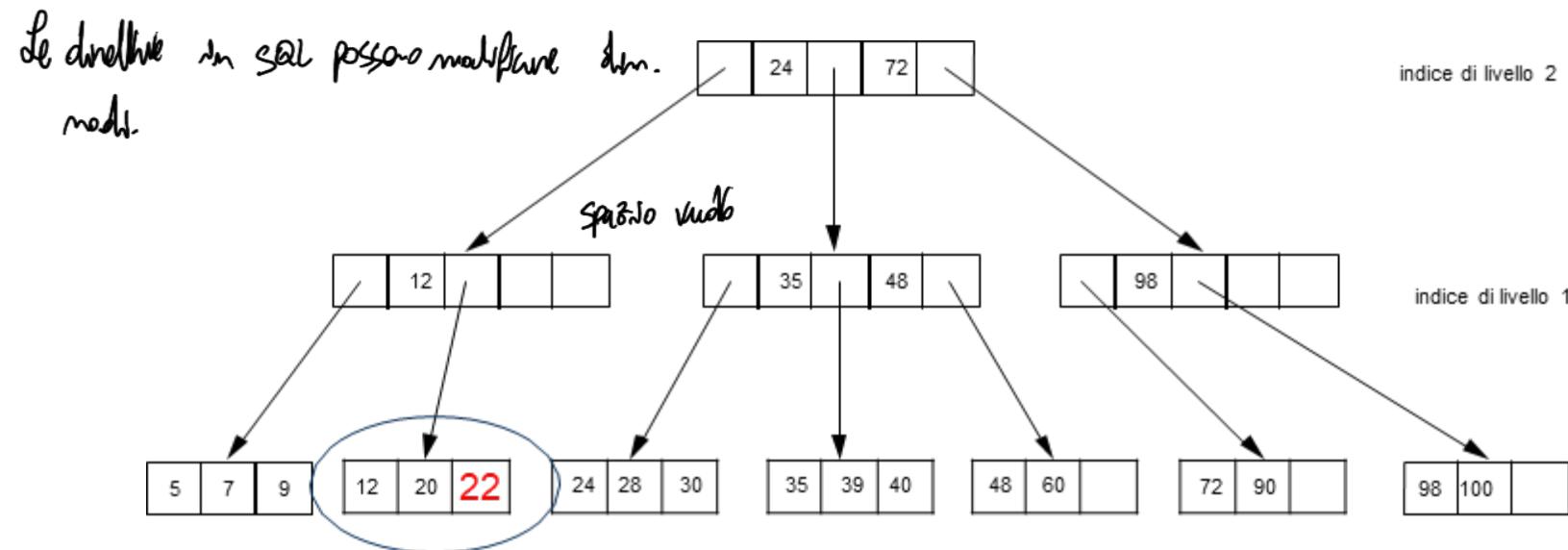


Se H è l'altezza dell'albero, nell'esempio $H=2$, sono necessari $H+1$ accessi per trovare il record

Più bilanciamento facess sempre $h+1$ accessi.

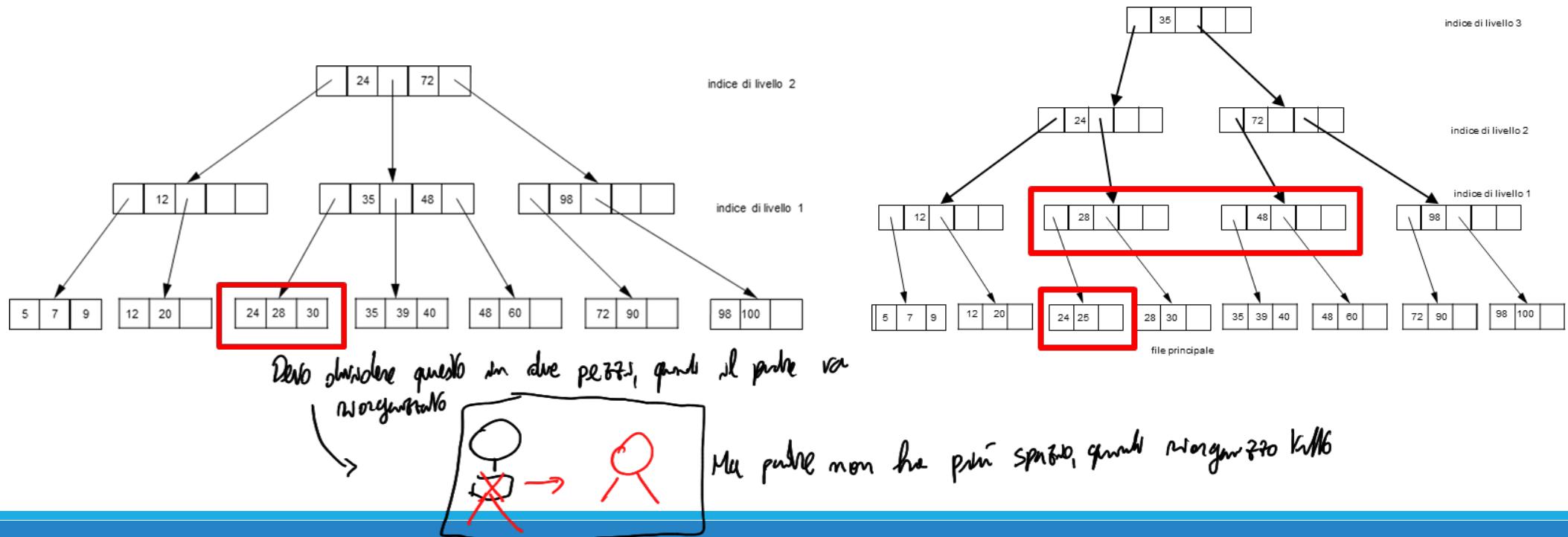
B-tree: Esempio di inserimento

- Inserimento del record con chiave di memorizzazione 22



B-tree: Esempio di inserimento

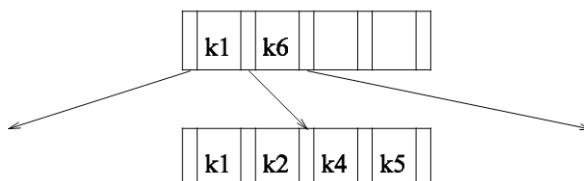
- Inserimento del record con chiave di memorizzazione 25



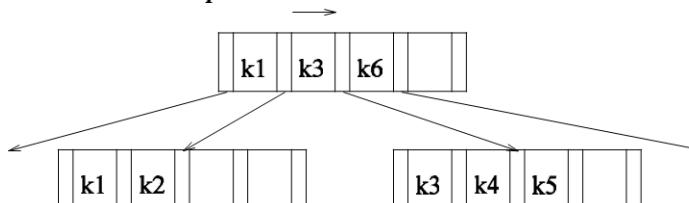
Split e merge: Operazioni su un albero B+

Quando la pagina della foglia non ha spazio disponibile, è necessario uno split, si suddivide l'informazione presente nella foglia (più quella che devo inserire) in due, allocando due foglie al posto di una. Questa operazione richiede una modifica della disposizione dei puntatori. **Aumento dei puntatori di una unità**

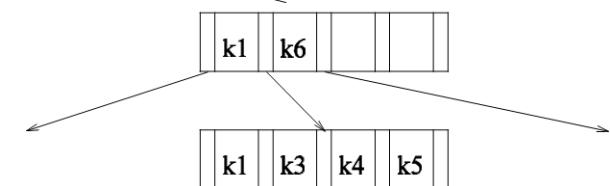
situazione iniziale



a. insert k3: split



b. delete k2: merge



Cancello K₂ e lo faccio con lo Spazio. Fatto.

Una cancellazione può essere sempre fatta in loco, marcando lo spazio precedentemente allocato ad una tupla come invalido. La modifica del valore di un campo chiave viene trattata come una cancellazione del suo valore iniziale seguito da un inserimento del suo valore finale

Vantaggi di un B⁺Tree

L'indice in questione è un indice denso: ogni record è indicizzato e questo fa sì che il file di dati non debba essere ordinato.

- Ciò che è costoso è, come per il B-Tree, l'inserimento e la cancellazione.
- Rispetto al B-Tree, il B⁺Tree permette di implementare in modo più efficienti le **range query**, ovvero le ricerche su intervalli di valori.

↳ es. cerca studenti con voti tra 25 e 30

Esistono in letteratura (ed in commercio) diverse varianti del B⁺Tree, in particolare il B^{*}Tree, che è un B⁺Tree con il vincolo che ogni nodo sia completo almeno per due terzi.

Alberi B e B+

- B+ tree:

- le chiavi compaiono tutte nelle foglie (e quindi quelle nei nodi intermedi sono comunque ripetute nelle foglie)

- le foglie sono collegate in una lista (*c'è un collegamento fra ogni due nodi foglia*)

- Ottimi per le ricerche su intervalli

- molto usati nei DBMS

- B tree:

- Le chiavi che compaiono nei nodi intermedi non sono ripetute nelle foglie

Esecuzione e ottimizzazione delle interrogazioni

Il **Query processor** (o **Ottimizzatore**): un **modulo cruciale** dell'architettura di un DBMS più importante nei sistemi attuali che in quelli "vecchi" (gerarchici e reticolari):

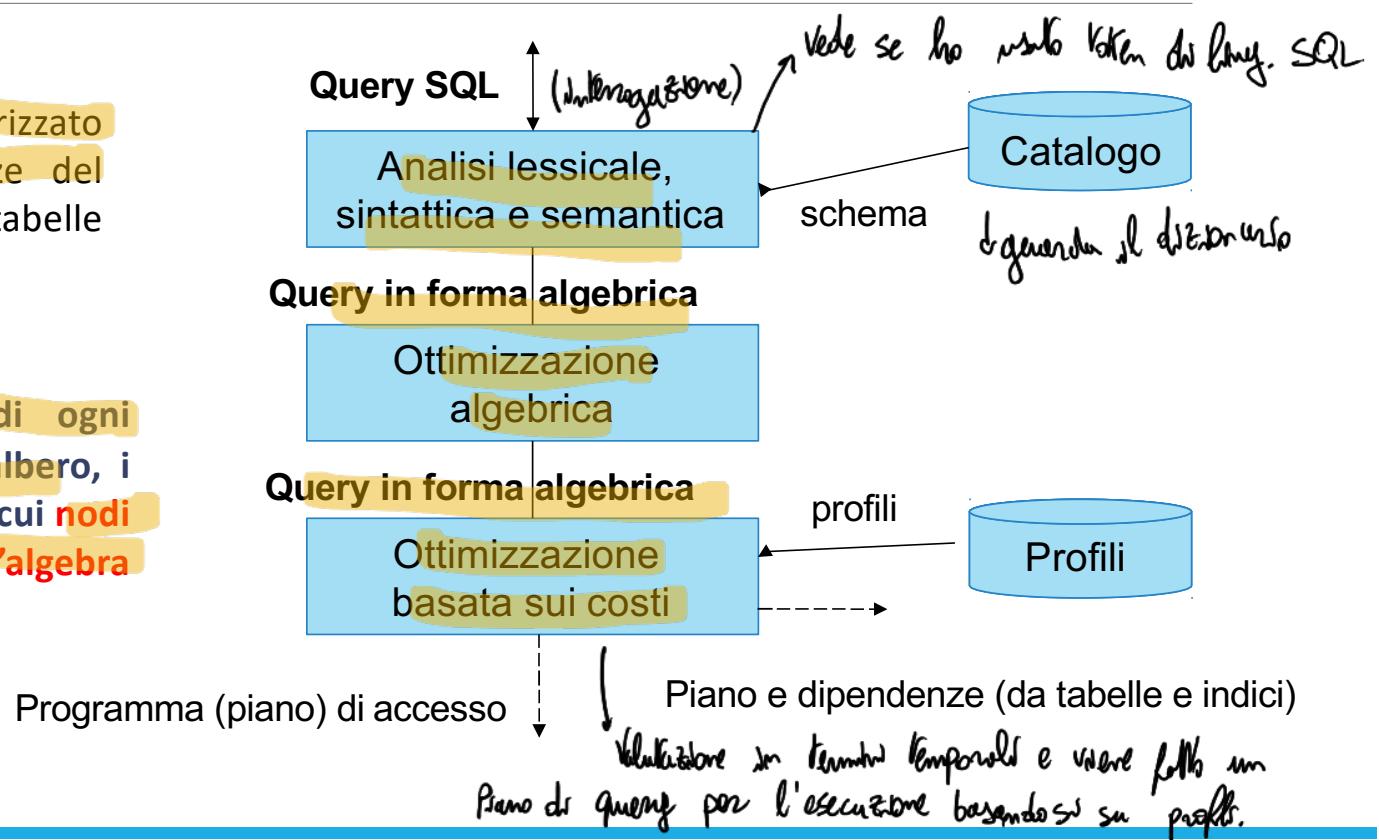
- le **interrogazioni sono espresse ad alto livello (indipendenza dei dati)**:
 - **insiemi di tuple**
 - **poca proceduralità**
- l'ottimizzatore **sceglie la strategia esecutiva «migliore»**, a partire dall'istruzione SQL

↓
modulo di algebra relazionale

Esecuzione delle interrogazioni

- Agisce a **tempo di compilazione**
- Il codice viene prodotto e memorizzato nella BD, insieme alle dipendenze del codice dalle particolari versioni di tabelle e indici della BD.

Il risultato è la rappresentazione di ogni interrogazione SQL sottoforma di un albero, i cui nodi foglia rappresentano tabelle e i cui nodi intermedi rappresentano operazioni dell'algebra relazionale



Ottimizzazione algebrica

- Il termine **ottimizzazione** ha lo scopo di trovare un'espressione che sia **equivalente** all'espressione data e possa essere eseguita in modo più efficiente
 - Due espressioni sono **equivalenti** se producono lo stesso risultato qualunque sia l'istanza attuale della base di dati
 - Ottimizzazione, in questo caso, è **improprio** poiché il processo utilizza tecniche **euristiche** (meccanismo basato su regole specifiche per produrre un piano di esecuzione efficiente)

I DBMS cercano di eseguire espressioni equivalenti a quelle date, ma meno "**costose**"

- **Euristica fondamentale:**
 - selezioni e proiezioni il prima possibile in modo da ridurre le dimensioni dei risultati intermedi
 - "push selections down"
 - "push projections down"

Esempio di ottimizzazione algebrica (I)

Impiegati(Matricola, Nome, Cognome, Dip)

Dipartimenti(Cod, Nome, Indirizzo, Città)

```
SELECT * FROM DIPARTIMENTI D JOIN IMPIEGATI I ON  
D.Cod=I.Dip WHERE I.Cognome='Rossi' ;
```

Se le tabelle sono ordinate quadrano.

Esempio di ottimizzazione algebrica (I)

$$\sigma_{I.Cognome='Rossi'}(D \triangleright\triangleleft_{D.Cod=I.Dip} I)$$

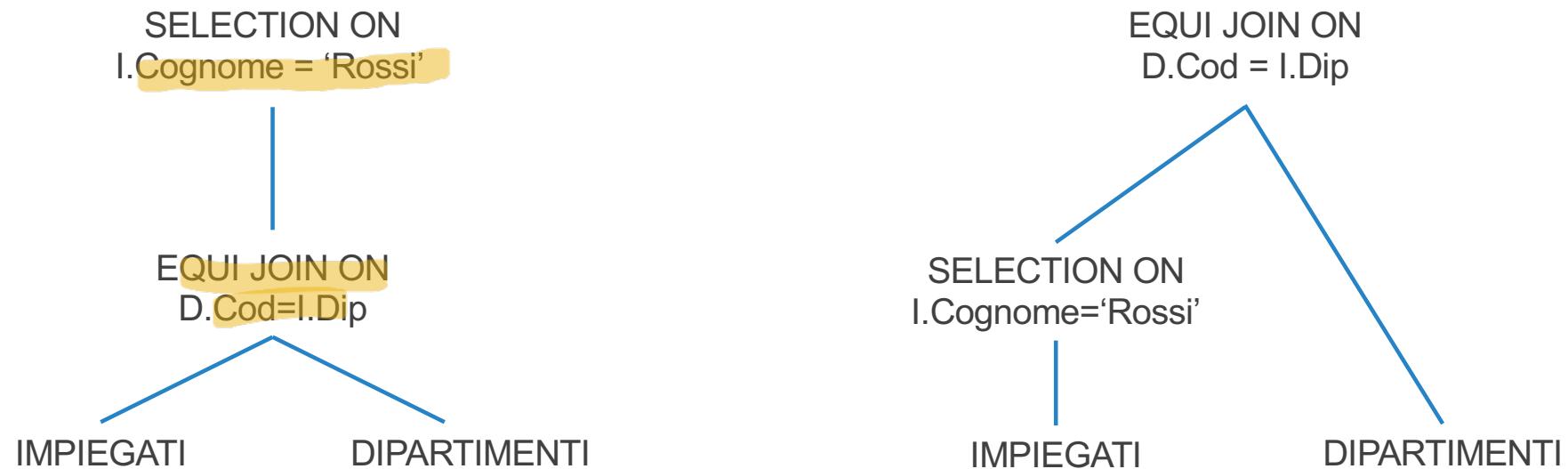
Se A e B due generiche relazione e F una condizione di selezione valida sugli attributi della relazione B allora:

$$\sigma_F(A \triangleright\triangleleft B) \equiv (A \triangleright\triangleleft \sigma_F(B))$$

Pushing selection down!

$$(D \triangleright\triangleleft_{D.Cod=I.Dip} \sigma_{I.Cognome='Rossi'}(I))$$

Rappresentazione con alberi di query



Alberi:

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

Una procedura euristica di ottimizzazione

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

Ottimizzazione basata sui costi

- Ogni DBMS in commercio possiede informazioni quantitative, dette **profili delle relazioni**:
 - La cardinalità di ciascuna tabella T (numero di tuple) : $\text{card}(T)$
 - La dimensione (in byte) delle tuple T : $\text{size}(T)$
 - La dimensione (in byte) di ciascun attributo A_j di T : $\text{size}(A_j, T)$
 - Il numero di valori distinti di ciascun attributo A_j di T : $\text{val}(A_j, T)$
 - Il valore minimo e massimo di ciascun attributo A_i di T : $\text{min}(A_i, T), \text{max}(A_i, T)$.
- I profili sono calcolati in base ai dati effettivamente memorizzati nelle tabelle, utilizzando primitive di sistema, ad esempio: “`update statistics`” (da effettuare periodicamente da addebito)
- Utilizzate nella fase finale dell'ottimizzazione, per stimare le dimensioni dei risultati intermedi.

Piani di Query

I profili sono utilizzati nell'ottimizzazione delle query per **stimare** le **dimensioni** dei risultati intermedi e scegliere, di volta in volta, quale sia la **migliore sequenza** possibile di operazioni di più basso livello

- Scansione
- Ordinamento
- Accesso diretto
- Join

Esistono due approcci:

Piano associato a query

- **Compile & Store**: il piano di esecuzione può essere memorizzato nel catalogo e richiamato più volte
- **Compile & Go**: determinare di volta in volta il piano di esecuzione delle query senza salvare niente nel catalogo

Scansione

→ di tuple su tabella

- Un'operazione di scansione (**scan**) esegue contestualmente varie operazioni di tipo **algebrico** ed **extra-algebrico**
 - Proiezione su una lista di attributi, senza eliminazione dei duplicati
 - Selezione su una o più condizioni
 - Inserimenti, cancellazioni e modifiche delle tuple durante una scansione.
- Primitive della scansione:
 - Open
 - Next
 - Read
 - Modify, Delete, Insert
 - Close

Ordinamento

- Non è una operazione algebrica in senso stretto ma è utile per:
 - La presentazione dei risultati
 - L'eliminazione dei duplicati
 - L'esecuzione più efficiente dei join
- Possono essere usati i ben noti algoritmi proposti in letteratura ma la complicazione è costituita dal fatto che i dati da ordinare potrebbero non andare tutti in memoria
 - Ordinamento con tecniche di merge-sort *↳ non sono presenti tutti nel buffer per esempio*
- La presenza di un indice adatto fa sì che l'ordinamento possa essere ottenuto semplicemente attraverso lo scan delle foglie dell'indice

Accesso diretto

- Si usa il termine **accesso diretto** quando è possibile **leggere** o **scrivere** un record senza dover necessariamente esaminare il file in modo sequenziale (disponibilità diretta dell'indirizzo del blocco).
- Può essere eseguito solo se le strutture fisiche lo permettono. Il predicato dell'interrogazione è valutabile tramite :
 - Indici**
 - Strutture hash**

Accesso diretto basato su indice

- Efficace per interrogazioni sulla chiave dell'indice
 - puntuali ($A_i = v$)
 - su intervallo ($v_1 \leq A_i \leq v_2$) *su range anche al arith*
- Se l'interrogazione presenta predici conguntivi
 - Cognome = 'Rossi' and nome = 'Mario'
 - si sceglie (eventualmente) il più selettivo per l'accesso diretto *Applico prima la costizione sul cognome*
 - si verifica l'altra condizione
- Se l'interrogazione presenta predici disgiuntivi:
 - Cognome = 'Rossi' or nome = 'Mario'
 - servono indici sui predici
 - È necessario eleminare i duplicati di quelle tuple che vengono ritrovate tramite più indici

Se man ho quale come fo?

Accesso diretto basato su hash

- Efficace per interrogazioni sulla chiave dell'indice
 - puntuali ($A_i = v$)
 - **NON SUPPORTATE** su intervallo ($v_1 < A_i < v_2$)
- Se l'interrogazione presenta predicati congiuntivi
 - Cognome = 'Rossi' and nome = 'Mario'
 - Hash su cognome implica l'accesso diretto
 - si verifica l'altra condizione
- Se l'interrogazione presenta predicati disgiuntivi
 - Cognome = 'Rossi' or nome = 'Mario'
 - **Impossibile**

Join

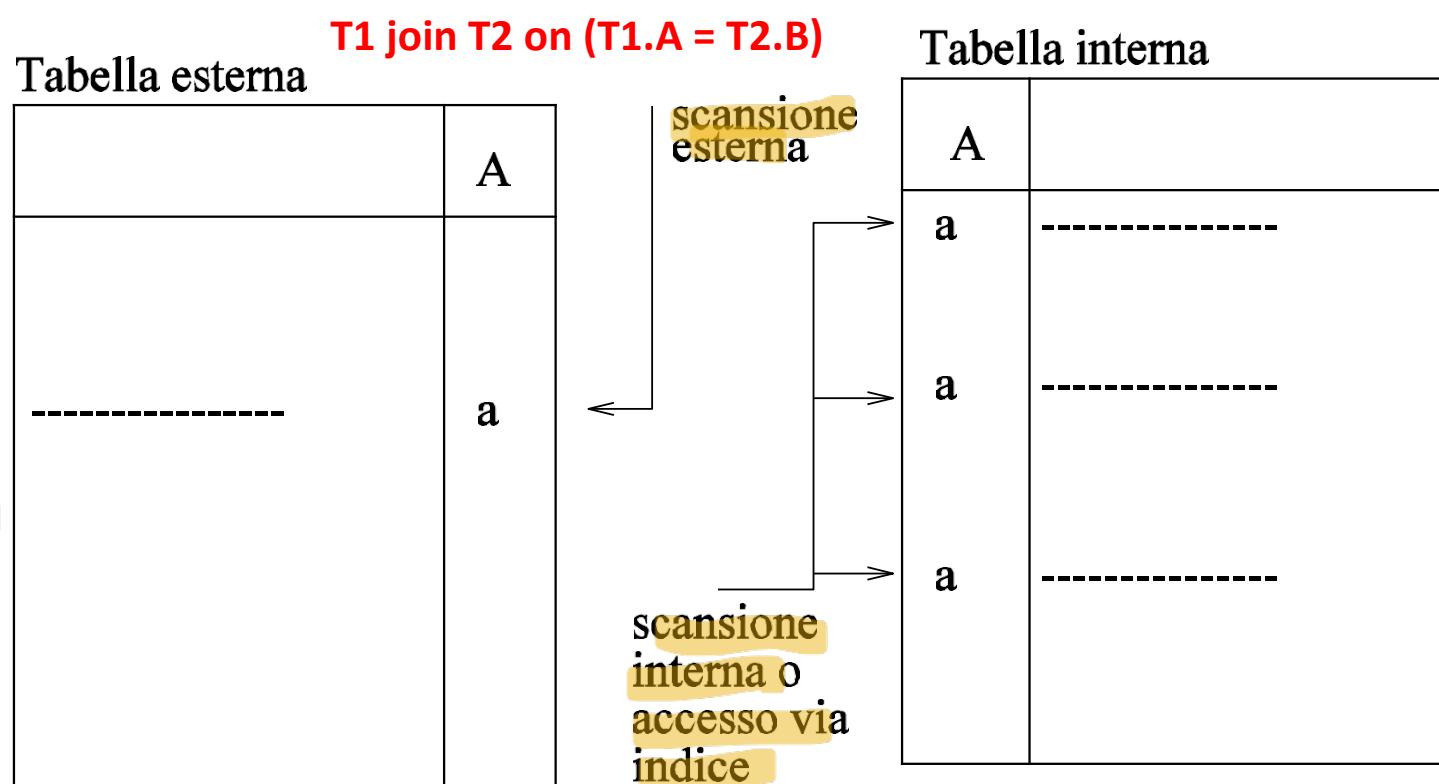
- È l'operazione più gravosa per un DBMS, poiché potrebbero esplodere il numero di tuple del risultato
- Esistono diverse tecniche per la realizzazione del join in un DBMS, le più note sono:
 - nested-loop*
 - merge-scan*
 - hash-based*
- Esse si basano sull'uso combinato di: Scansione, accesso via indice o via hash, ordinamento

Nested-loop (nidificata)

1. Si esegue una scansione sulla tabella esterna

2. Per ogni tupla ritrovata dalla scansione, si preleva il valore dell'attributo di join e si cercano le tuple della tabella interna che hanno lo stesso valore

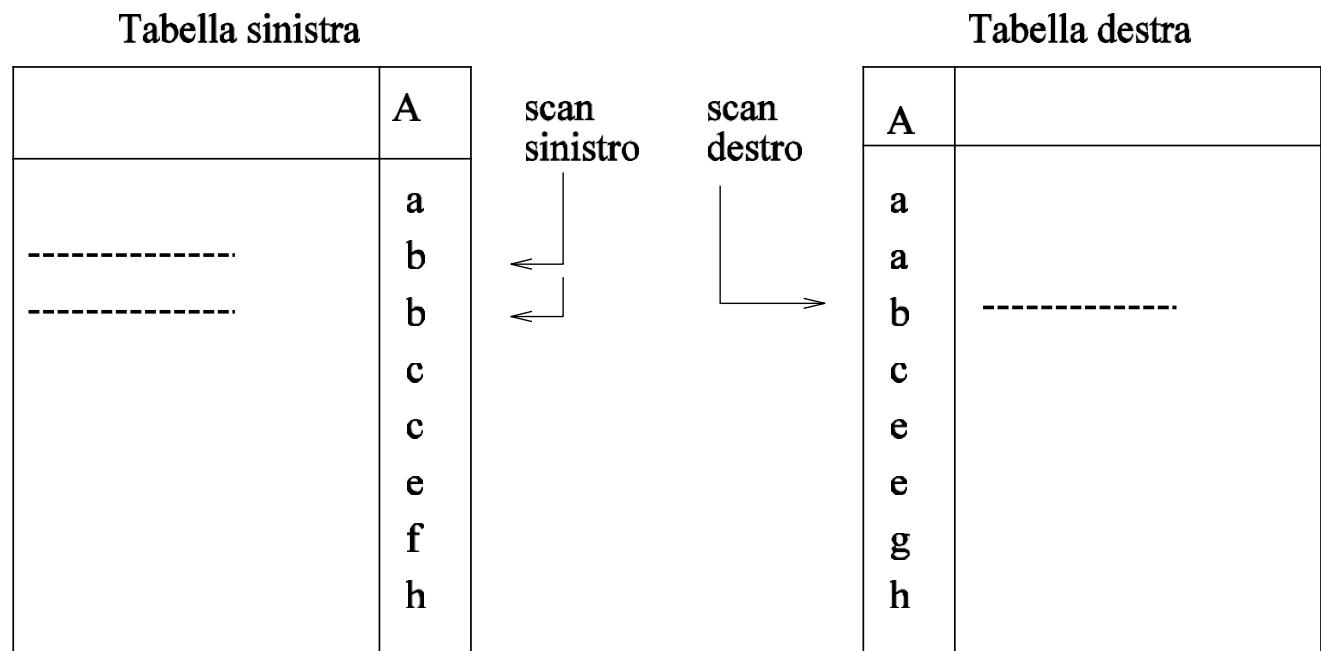
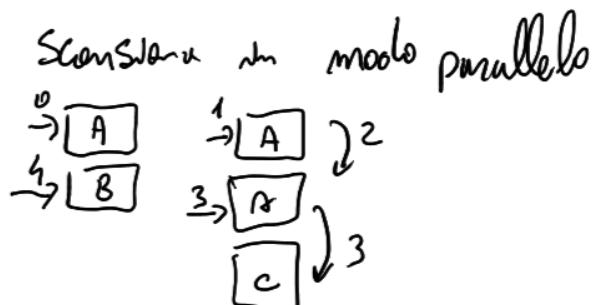
Nested-loop (scansione nidificata) nell'altra



Merge-scan

T1 join T2 on (T1.A = T2.B)

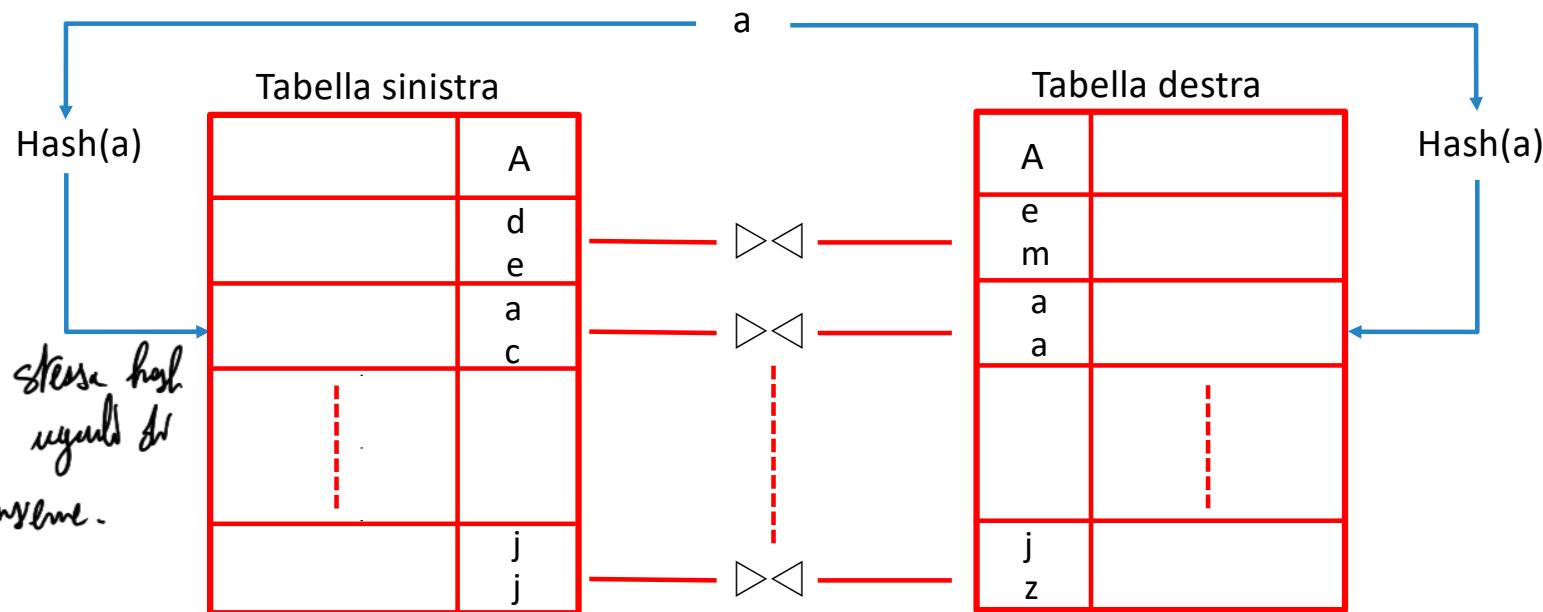
1. Esamina le tabelle secondo l'ordine degli attributi del join (efficiente per tabelle già ordinate)
2. Scansioni parallele su due tabelle, come negli algoritmi di fusione (merge)



Hash-based

Viene utilizzata una funzione hash "h" per generare copia delle due tabelle (o di loro parte);

Si generano B partizioni (con ugual valore dell'indice) su cui effettuare un join "semplice".



Ottimizzazione basata sui costi

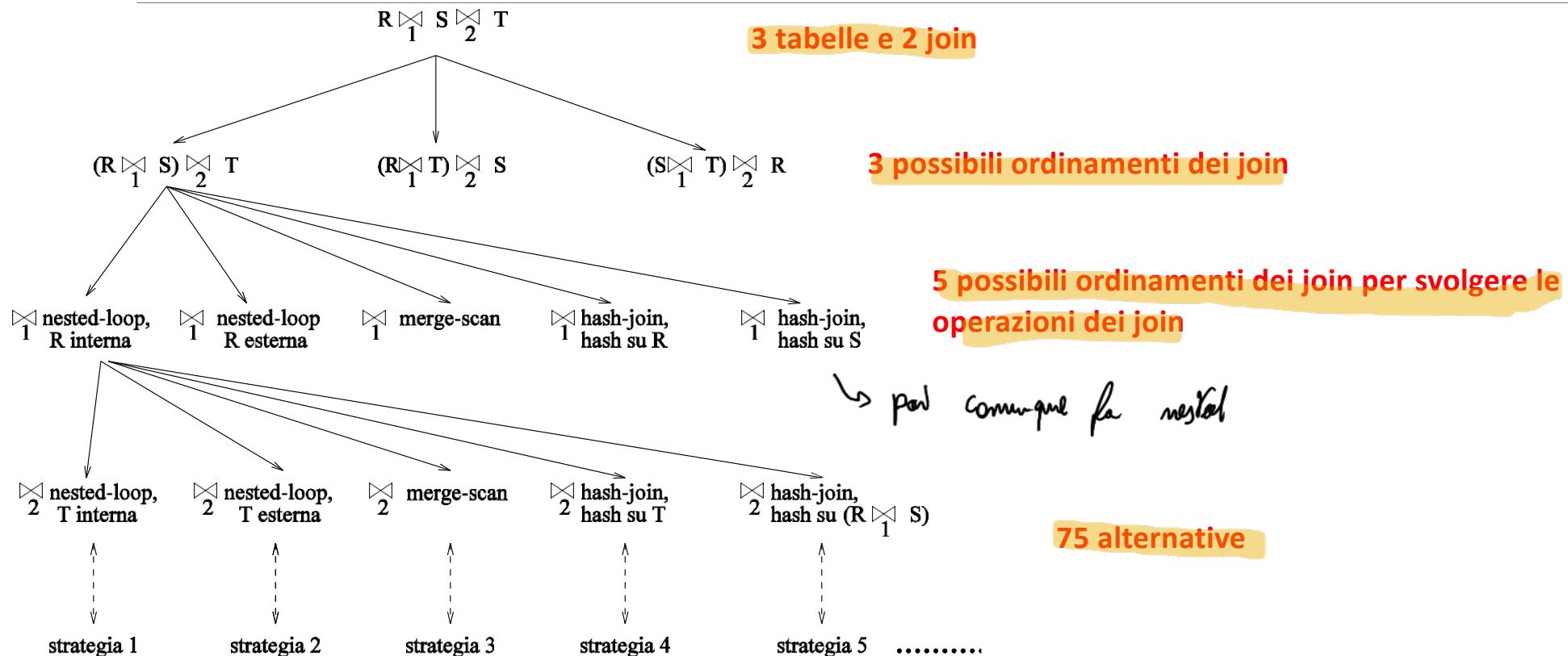
- È chiaro che **ogni tecnica ha un costo** che dipende dalla situazione iniziale e finale, per cui la scelta di quale tecnica utilizzare deve seguire delle linee ben precise.
- Bisogna definire un piano di accesso, per risolvere il problema legato alla complessità computazionale
 - Scegliere quali operazioni di accesso ai dati svolgere (indici, scansione, ecc)
 - Scegliere l'ordine delle operazioni da compiere
 - Se ci sono diverse alternative, scegliere quale alternativa associare all'operazione da eseguire (**ad esempio scegliere il metodo di join**)
 - Se richiesto un ordinamento, definire a quale livello della strategia ordinare i dati

Il processo di ottimizzazione

- Si costruisce un albero delle alternative ("piani di esecuzione")
- Si valuta il costo di ciascun piano
- Si sceglie il piano di costo minore
- L'ottimizzatore trova di solito una "buona" soluzione, non necessariamente l'"ottimo"

↓
Ma per risolvere
"molto più tempo"

Albero delle alternative (di decisione)



Funzionalità dei DBMS per il piano di esecuzione

- I DBMS di solito prevedono delle funzionalità che permettono di conoscere il piano di esecuzione scelto, a seguito di una interrogazione SQL, ovviamente in base ai profili contenuti nel catalogo
- EXPLAIN: per ottenere le informazioni su come viene ottenuto il risultato di una query (efficienza)
- EXPLAIN ANALYZE: per ottenere la stima del piano di esecuzione

Esempio

- Supponiamo di avere il seguente modello relazionale

Corsi(Codice, Corso)

Studenti(Matricola, Nome, Cognome, Indirizzo)

Carriere(MatStudente, CodiceCorso, Data, Voto)

punto dove come ha visto svolti

Voglio vedere zane che esamini sostituti

EXPLAIN SELECT *

FROM Studenti S JOIN Carriere C on S.Matricola = C.MatStudente INNER JOIN Corsi R on R.Codice=C.CodiceCorso
Where S.Cognome="Zara";

id	select_type	table	partitions	type	possible_keys	dipende se ho visto come			n. terna o estern	rows	filtered	Extra
						key	key_len	ref				
1	SIMPLE	S	NULL	ALL	PRIMARY	NULL	NULL	NULL	5	20.00	Using where	
1	SIMPLE	C	NULL	ref	PRIMARY,CodiceCorso	PRIMARY	4	my_itcarli.S.Matricola	1	100.00	NULL	
1	SIMPLE	R	NULL	eq_ref	PRIMARY	PRIMARY	42	my_itcarli.C.CodiceCorso	1	100.00	NULL	

↑

↑
N. partizioni

↓ ho visto tutti chiamati prima

↑ voglie risultante

Tipo selezione, tutti i join semplici

Esempio

```
EXPLAIN ANALYZE SELECT *
FROM Studenti S JOIN Carriere C on S.Matricola = C.MatStudente INNER JOIN Corsi R on R.Codice=C.CodiceCorso
Where S.Cognome="Zara";
```

```
EXPLAIN
-> Nested loop inner join (cost=1.71 rows=1) (actual time=0.035..0.043 rows=2 loops=1)
-> Nested loop inner join (cost=1.36 rows=1) (actual time=0.027..0.031 rows=2 loops=1)
  -> Filter: (S.Cognome = 'Zara') (cost=1.01 rows=1) (actual time=0.012..0.014 rows=1 loops=1)
    -> Table scan on S (cost=1.01 rows=5) (actual time=0.005..0.010 rows=5 loops=1)
    -> Index lookup on C using PRIMARY (MatStudente=S.Matricola) (cost=0.35 rows=1) (actual time=0.012..0.015 rows=2 loops=1)
-> Single-row index lookup on R using PRIMARY (Codice=C.CodiceCorso) (cost=0.35 rows=1) (actual time=0.005..0.005 rows=1 loops=2)
```

Progettazione fisica

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

Progettazione fisica nel modello relazionale

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

Definizione degli indici SQL

Non è standard, ma presente in forma simile nei vari DBMS

- `create [unique] index IndexName on TableName(AttributeList)`
- `drop index IndexName`

(indice è definibile su più attributi)

Strutture fisiche nei DBMS relazionali

Struttura primaria:

- disordinata (heap, "unclustered")
- ordinata ("clustered"), anche su una pseudochiave
- hash ("clustered"), anche su una pseudochiave, senza ordinamento
- clustering di più relazioni

Indici (densi/sparsi, semplici/composti):

- ISAM (statico), di solito su struttura ordinata
- B-tree (dinamico)

Strutture fisiche in alcuni DBMS

Oracle:

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