

# *File e Indici*



- ❖ Dati di un DBMS memorizzati come *record*
- ❖ Un *file* è una collezione di record
- ❖ *Organizzazione del file*: metodo per registrare un *file* su un dispositivo di memorizzazione esterno
  - Un *record ID (rid)* è sufficiente per localizzare fisicamente il *record*
  - Gli *indici* ci permettono di trovare i *rid* dei *record* con valori dati nei campi della chiave di ricerca dell'indice



## *Dati su dispositivi di memorizzazione esterni*

- ❖ Dischi: si può leggere qualunque *pagina* (4KB o 8 KB) a costo fisso
  - Ma leggere diverse pagine consecutive è molto più economico che leggerle in ordine casuale
- ❖ Nastri: si possono leggere le pagine solo in sequenza
  - Più economici dei dischi; usati per memorizzare archivi
- ❖ Architettura:
  - *gestore di buffer* carica i dati in memoria per elaborarli e scrive su disco per memorizzarli
  - *strato di gestione dei file* e quello *degli indici* eseguono le chiamate al gestore di buffer



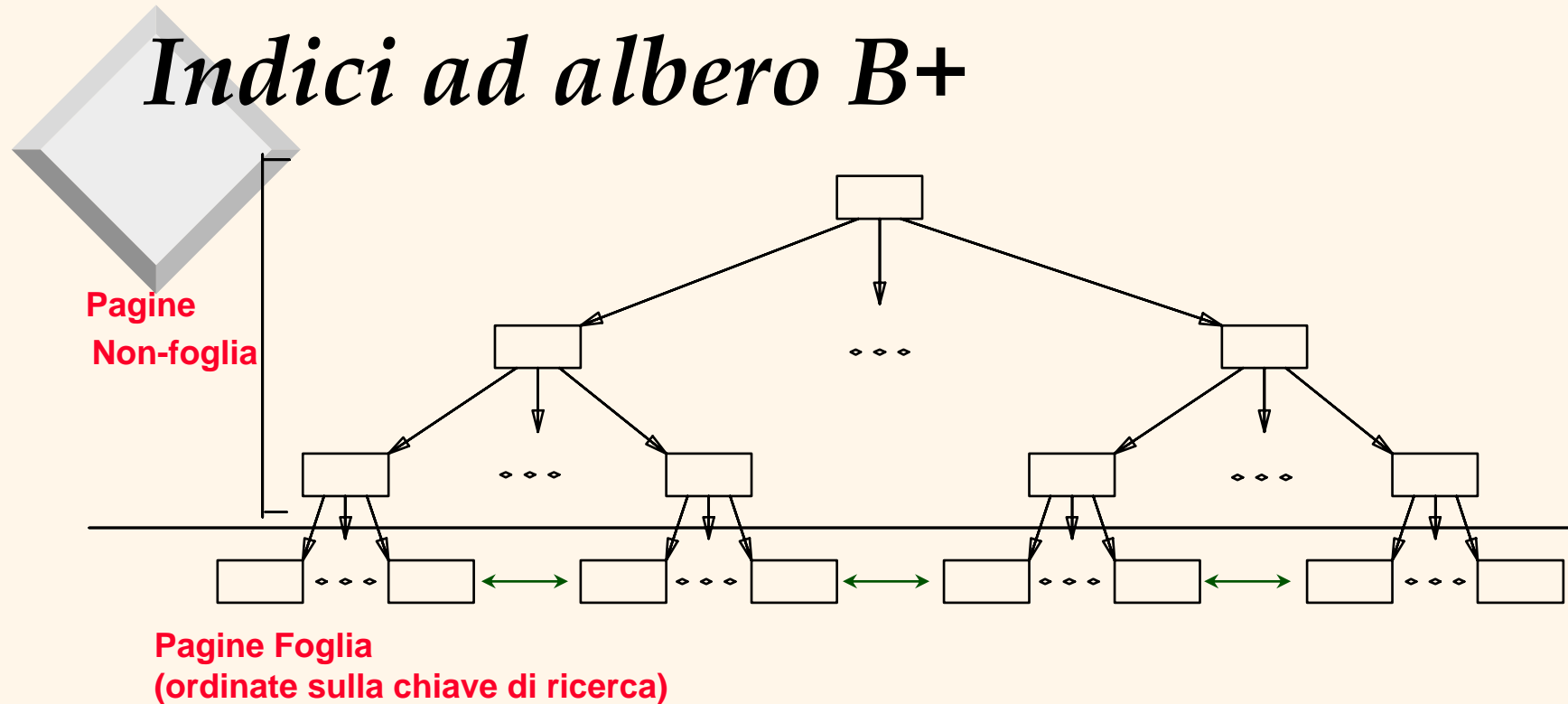
# *Organizzazioni alternative dei file*

- ❖ *Varie alternative, ciascuna ideale per qualche situazione, e non molto buona in altre :*
  - *file heap* (ordine casuale): va bene quando l'accesso tipico è una scansione di *file* per leggere tutti i *record*
  - *file ordinato*: ottimale quando i *record* devono essere letti in qualche ordine, o quando si ha bisogno solo di un "intervallo" di *record*
  - **indici**: strutture di dati per organizzare i *record* (in alberi o tramite hashing) di modo da ottimizzare operazioni di lettura
    - ◆ come i *file* ordinati, essi velocizzano le ricerche di sottoinsiemi di *record*, basate su valori in certi campi ("chiavi di ricerca")
    - ◆ gli aggiornamenti sono molto più veloci che nei *file* ordinati

# Indici

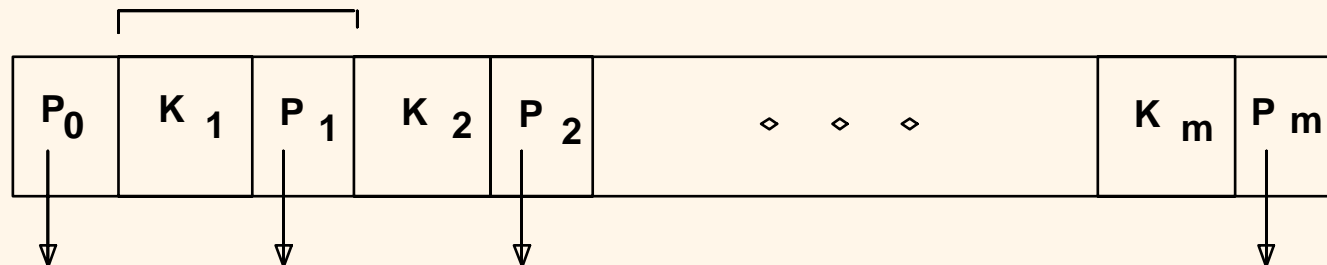
- ❖ Un indice su un *file* velocizza le selezioni sui campi che compongono la chiave di ricerca per l'indice
  - Qualunque sottoinsieme dei campi di una relazione può essere la chiave di ricerca per un indice sulla relazione
  - La chiave di ricerca non è la chiave (insieme minimo di campi che identificano univocamente un *record* in una relazione)
- ❖ Un indice contiene una collezione di *data entry*, e supporta il reperimento efficiente di tutte le *data entry*  $k^*$  con un dato valore  $k$  della chiave
  - Avendo la *data entry*  $k^*$  possiamo trovare i *record* con chiave  $k$  in al più un I/O di disco (fra poco i dettagli...)

# Indici ad albero B+

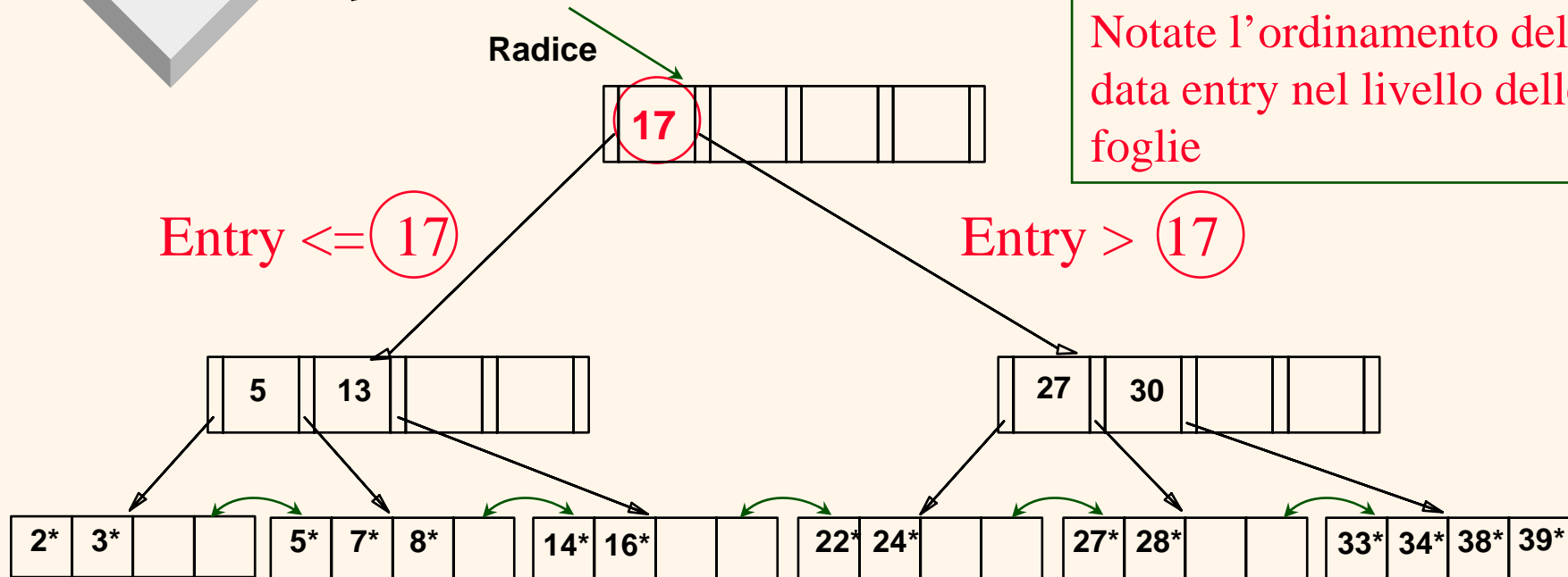


- ❖ Le pagine foglia contengono *data entry*, e sono collegate (prec & succ)
- ❖ Le pagine non-foglia contengono *index entry*, usate solo per guidare le ricerche

## Entry dell'indice



# Esempio di albero B+




- ❖ Trovare  $28^*$ ?  $29^*$ ? Tutte  $>15^*$  e  $<30^*$
- ❖ Inserimento/cancellazione: trovare la *data entry* nella foglia, poi cambiarla. A volte bisogna aggiustare i nodi genitori
  - E qualche volta la modifica si propaga a tutto l'albero

# *Indici basati su hash*

- ❖ Buoni per selezioni basate su uguaglianze (ovvero record che hanno un dato valore nella chiave di ricerca)
- ❖ L'indice è una collezione di *bucket*
  - *Bucket* = pagina *primaria* più zero o più pagine addizionali
  - I *bucket* contengono le *data entry*
- ❖ Funzione *hash*  $h$ :
  - $h(\text{chaive di ricerca } r) = \textit{bucket}$  cui appartiene (la *data entry* per) il *record*  $r$ .  $h$  si applica ai campi della chiave di ricerca di  $r$
  - In questo schema non c'è bisogno di *index entry*

# Alternative per le data entry $k^*$ nell'indice

- ❖ In una *data entry*  $k^*$  possiamo memorizzare:
  1. *record* di dati con valore di chiave  $K$ , oppure
  2. Coppia  $\langle k, rid \rangle$  dove *rid* è l'identificativo di un *record* con valore della chiave di ricerca  $k$ , oppure
  3.  $\langle k, \text{lista dei } rid \text{ dei } record \text{ di dati con chiave di ricerca } k \rangle$
- ❖ la scelta dell'alternativa per le *data entry* è ortogonale alla tecnica di indicizzazione usata per localizzare le *data entry* con un dato valore di chiave  $K$ 
  - esempi di tecnica di indicizzazione: alberi B+, strutture basate su *hash*
  - tipicamente, l'indice contiene informazioni aggiuntive che guidano le ricerche verso le *data entry* desiderate



# *Alternative per le data entry (segue)*

## ❖ Alternativa 1:

- se si usa questa alternativa, si ha una sorta di organizzazione di *file* indicizzata
- al più un indice su una data collezione di *record* di dati può usare l'Alternativa 1 (altrimenti, i *record* di dati sono duplicati, il che porta a memorizzazione ridondante e potenziale inconsistenza)
- se i *record* di dati sono molto grandi, il numero di pagine contenente le *data entry* è alto. Tipicamente, ciò implica che la dimensione delle informazioni aggiuntive nell'indice è anch'essa molto grande



# *Alternative per le data entry (segue)*

## ❖ *Alternative 2 e 3:*

- Le *data entry* tipicamente sono molto più piccole dei *record* di dati. Quindi, meglio dell'Alternativa 1 con *record* di dati grandi, specialmente se le chiavi di ricerca sono piccole (la porzione della struttura ad indice usata per dirigere la ricerca, che dipende dalla dimensione delle *data entry*, è molto più piccola rispetto all'Alternativa 1)
- L'alternativa 3 è più compatta dell'Alternativa 2, ma porta a *data entry* di dimensioni variabili; anche se le chiavi di ricerca sono di lunghezza fissa

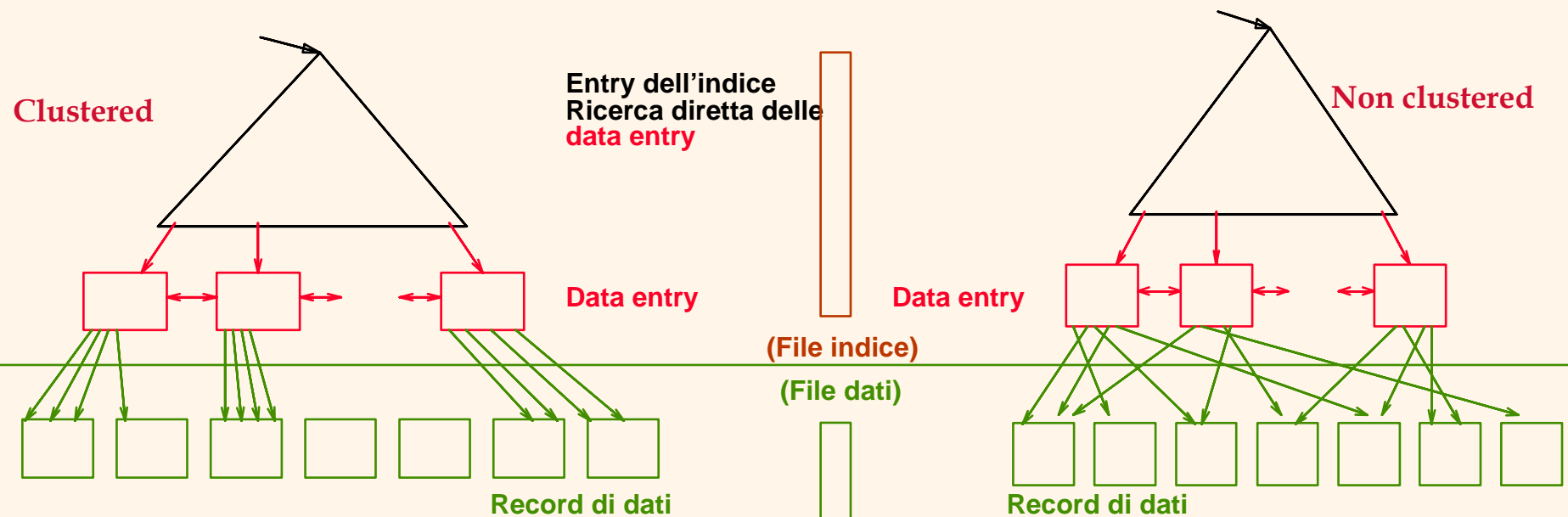


# *Classificazione degli indici*

- ❖ **Primario o secondario:** se la chiave di ricerca contiene la chiave primaria, allora è chiamato indice primario
  - **Indice univoco:** la chiave di ricerca contiene una chiave candidata
- ❖ ***Clustered* e non *clustered*:** se l'ordine dei *record* di dati è lo stesso, o "quasi", delle *data entry*, allora è chiamato indice *clustered*
  - L'Alternativa 1 implica un indice *clustered*; nella pratica, anche gli indici *clustered* implicano l'Alternativa 1 (poiché i *file* ordinati sono rari)
  - Un *file* può essere *clustered* su al più una chiave di ricerca
  - Il costo della lettura dei *record* di dati usando l'indice varia *fortemente* in base al fatto che l'indice sia *clustered* oppure no!

# Indici clustered e non clustered

- ❖ Supponiamo che venga usata l'Alternativa 2 per le *data entry*, e che i *record* di dati siano memorizzati in un *file heap*
  - Per costruire un indice *clustered*, prima ordiniamo il *file heap* (con dello spazio libero su ciascuna pagina per gli inserimenti futuri)
  - Potrebbero essere necessarie delle pagine



# *Modello di costo per la nostra analisi*

Tralasciamo i costi della CPU, per semplicità:

- ❖ B: il numero di pagine di dati
- ❖ R: il numero di *record* per pagina
- ❖ D: tempo (medio) per leggere o scrivere una pagina su disco
- ❖ La misura del numero di I/O di pagina trascurava il guadagno dovuto al pre-caricamento di una sequenza di pagine; quindi, anche il costo di I/O è solo approssimato
- ❖ Analisi del caso medio; basata su diverse assunzioni semplificatrici

✉ *Abbastanza buona per mostrare la tendenza generale !*

# Confronto tra le organizzazioni di file

- ❖ *File heap* (ordine casuale; inserimento alla fine del *file*)
- ❖ *File* ordinati su  $\langle \text{età}, \text{sal} \rangle$
- ❖ *File clustered* ad albero B+, Alternativa (1), chiave di ricerca  $\langle \text{età}, \text{sal} \rangle$
- ❖ *File heap* con indice *hash non clustered* su chiave di ricerca  $\langle \text{età}, \text{sal} \rangle$

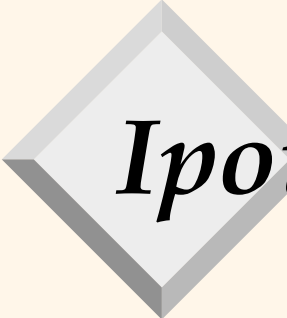


# *Operazioni da confrontare*

- ❖ Scansione: lettura di tutti i *record* dal disco
- ❖ Ricerca con selezione di uguaglianza
- ❖ Ricerca con selezione di intervallo
- ❖ Inserimento di un *record*
- ❖ Cancellazione di un *record*

# *Ipotesi nella nostra analisi*

- ❖ *File heap:*
  - selezione di uguaglianza sulla chiave; un solo risultato
- ❖ *File ordinati:*
  - *file* compattati dopo le cancellazioni
- ❖ *Indici:*
  - Alternative (2), (3): dimensione delle *data entry* = 10% della dimensione dei *record*
  - Hash: niente *bucket* aggiuntivi
    - ◆ 80% di occupazione delle pagine => dimensione del *file* = 1.25 volte la dimensione dei dati
  - Albero: 67% di occupazione (valore tipico)
    - ◆ Implica una dimensione di *file* = 1.5 volte la dimensione dei dati



## *Ipotesi (segue)*

### ❖ Scansioni:

- i livelli foglia di un indice ad albero sono collegati
- le *data entry* dell'indice più il *file* vero e proprio vengono scansionati per indici non *clustered*

### ❖ Ricerche con selezione di intervallo:

- usiamo gli indici ad albero per restringere l'insieme di *record* di dati restituito, ma tralasciamo gli indici *hash*

# Cost of Operations


	(a) Scan	(b) Equality	(c) Range	(d) Insert	(e) Delete
(1) Heap					
(2) Sorted					
(3) Clustered					
(4) Unclustered Tree index					
(5) Unclustered Hash index					

✉ *Diverse ipotesi alla base di queste (grossolane) stime!*

# Costo delle operazioni

	(a) Scan	(b) Equality	(c) Range	(d) Insert	(e) Delete
(1) Heap	BD	0.5BD	BD	2D	Search +D
(2) Sorted	BD	$D \log_2 B$	$D(\log_2 B + \text{\# pgs with match recs})$	Search + BD	Search +BD
(3) Clustered	1.5BD	$D \log_F 1.5B$	$D(\log_F 1.5B + \text{\# pgs w. match recs})$	Search + D	Search +D
(4) Unclust. Tree index	$BD(R+0.15)$	$D(1 + \log_F 0.15B)$	$D(\log_F 0.15B + \text{\# pgs w. match recs})$	Search + 2D	Search + 2D
(5) Unclust. Hash index	$BD(R+0.125)$	2D	BD	Search + 2D	Search + 2D

- B: il numero di pagine di dati; R: il numero di *record* per pagina; D: tempo (medio) per leggere o scrivere una pagina su disco
- Heap: + a,d - b,c,e
- Sorted: + a,b,c -d,e
- Clustered: + a,b,c =de
- Tree unclustered: +b,c,d,e - a,c(multi recs)
- Hash unclustered: +b,c,d,e - a,c(multi recs)



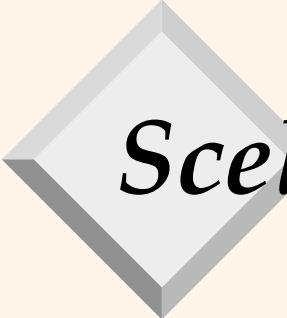
# *Comprendere il carico di lavoro*

- ❖ Per ciascuna interrogazione nel carico di lavoro:
  - A quali relazioni accede?
  - Quali attributi vengono restituiti?
  - Quali attributi sono coinvolti nelle condizioni di selezione/join? Quanto selettive sono, probabilmente, tali condizioni?
- ❖ Per ciascun aggiornamento nel carico di lavoro:
  - Quali attributi sono coinvolti nelle condizioni di selezione/join? Quanto selettive sono, probabilmente, tali condizioni?
  - Il tipo di aggiornamento (INSERT/DELETE/UPDATE) e gli attributi che vengono modificati



## *Scelta degli indici*

- ❖ Quali indici dovremmo creare?
  - Quali relazioni dovrebbero avere indici? Quali campi dovrebbero costituire la chiave di ricerca? Dovremmo creare più di un indice?
- ❖ Per ciascun indice, che tipo di indice dovremmo usare?
  - *Clustered? Hash/albero?*



## *Scelta degli indici (segue)*

- ❖ Un approccio: consideriamo a turno le interrogazioni più importanti. Consideriamo il miglior piano usando gli indici correnti, e vediamo se è possibile un piano migliore con un indice aggiuntivo. Se sì, creiamolo.
  - Ovviamente, ciò implica la comprensione di come un DBMS valuta le interrogazioni e crea i piani di valutazione delle interrogazioni!
  - Per ora discutiamo semplici interrogazioni di una tabella
- ❖ Prima di creare un indice, dobbiamo anche considerare l'impatto sugli aggiornamenti nel carico di lavoro!
  - Compromesso: gli indici possono velocizzare le interrogazioni, rallentare gli aggiornamenti. Anche richiedere spazio su disco

# Linee guida per la selezione degli indici

- ❖ Gli attributi nella clausola WHERE sono candidati come chiavi dell'indice
  - Una condizione di corrispondenza esatta suggerisce un indice *hash*
  - Una interrogazione con selezione di intervallo suggerisce un indice ad albero
    - ◆ L'uso dei *cluster* è utile specialmente per le interrogazioni con selezione di intervallo; può anche essere di aiuto per le interrogazioni con selezione di uguaglianza se ci sono molti duplicati
- ❖ Chiavi di ricerca multi-attributo dovrebbero essere prese in considerazione quando una clausola WHERE contiene diverse condizioni
  - L'ordine degli attributi è importante per le interrogazioni con selezione di intervallo
  - Tali indici possono a volte consentire strategie basate soltanto sull'indice per le interrogazioni importanti
    - ◆ Per le strategie basate soltanto sull'indice l'essere *clustered* non è importante!
- ❖ Provare a scegliere gli indici che supportano quante più interrogazioni possibile. Poiché solo un indice per ogni relazione può essere *clustered*, sceglierlo in base alle interrogazioni importanti che più traggono beneficio dal *clustering*

# Esempi di indici clustered

- ❖ Un indice ad albero B+ su I.età può essere usato per selezionare le tuple
  - Quanto è selettiva la condizione?
  - L'indice è *clustered*?
- ❖ Consideriamo l'interrogazione con GROUP BY
  - Se molte tuple hanno I.età > 10, usare l'indice su I.età e ordinare le tuple restituite può essere costoso
  - L'indice *clustered* I.rnum può essere migliore!
- ❖ Interrogazioni con selezione di uguaglianza e duplicati
  - Il clustering su I.hobby aiuta!

```
SELECT I.rnum
FROM Imp I
WHERE I.età > 40
```

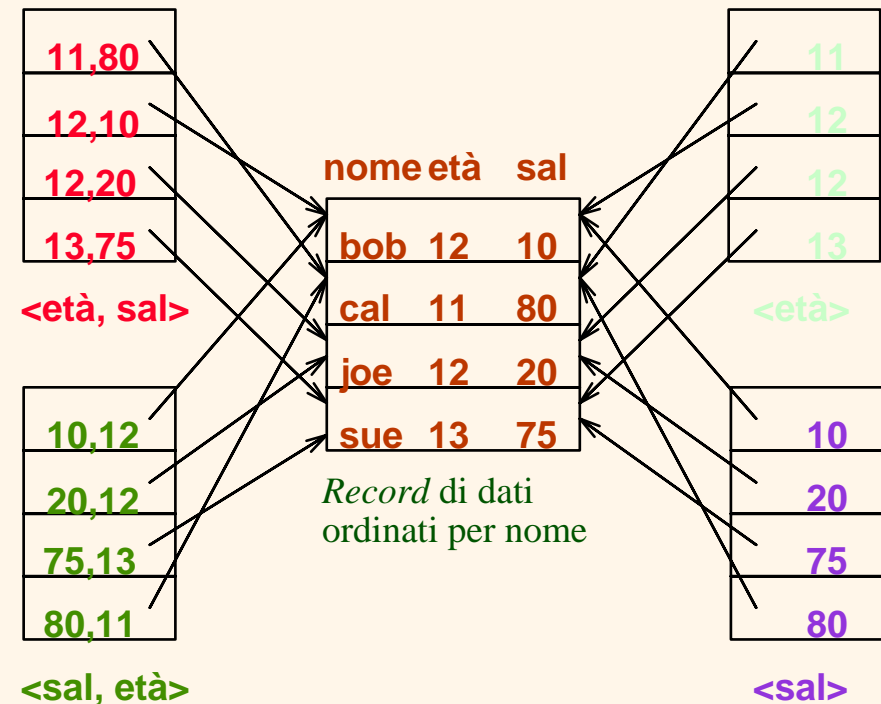
```
SELECT I.rnum, COUNT(*)
FROM Imp I
WHERE I.età > 10
GROUP BY I.rnum
```

```
SELECT I.rnum
FROM Imp I
WHERE
I.hobby='Francobolli'
```

# Indici con chiavi di ricerca composite

- ❖ Chiavi di ricerca composite: ricerca su una combinazione di campi
  - Interrogazioni con selezione di uguaglianza: il valore di ogni campo è uguale a un valore costante. Ad esempio, con riferimento all'indice <sal, età>:
    - ◆ età = 20 and sal = 75
  - Interrogazioni con selezione di intervallo: il valore di alcuni campi non è una costante. Ad esempio:
    - ◆ età = 20 and sal > 10
  
- ❖ le *data entry* nell'indice sono ordinate per chiave di ricerca per supportare le interrogazioni con selezione di intervallo
  - ordine lessicografico, oppure
  - ordine spaziale

Esempi di indice con chiave composta che usa l'ordine lessicografico



Data entry nell'indice ordinate per <sal, età>

Data entry ordinate per <sal>

# Chiavi di ricerca composite

- ❖ Per trovare i *record* di Imp con età = 30 AND sal = 4000, un indice su <età, sal> sarebbe meglio di un indice su età o di un indice su sal
  - La scelta della chiave dell'indice è ortogonale al *clustering* etc.
- ❖ Se la condizione è  $20 < \text{età} < 30$  AND  $3000 < \text{sal} < 5000$ 
  - È meglio un indice *clustered* ad albero su <età, sal> o su <sal, età>
- ❖ Se la condizione è età = 30 AND  $3000 < \text{sal} < 5000$ 
  - Un indice *clustered* <età, sal> è molto meglio di un indice su <sal, età>
- ❖ Gli indici composti sono più grandi, e aggiornati più spesso

# Piani basati solo sull'indice

- Se è disponibile un indice opportuno, a molte interrogazioni si può rispondere senza leggere alcuna tupla da alcuna delle relazioni coinvolte

<I.rnum>

```
SELECT I.rnum, COUNT(*)  
FROM Imp I  
GROUP BY I.rnum
```

<I.rnum, I.sal>  
Indice ad albero!

```
SELECT I.rnum, MIN(I.sal)  
FROM Imp I  
GROUP BY I.rnum
```

<I.età, I.sal>  
oppure  
<I.sal, I.età>  
Indice ad albero!

```
SELECT AVG(I.sal)  
FROM Imp I  
WHERE I.età = 25 AND  
E.sal BETWEEN 3000 AND 5000
```

## *Piani basati solo sull'indice (segue)*

- ❖ Piani basati solo sull'indice sono possibili se la chiave è  $\langle \text{rnum}, \text{età} \rangle$  o se abbiamo un indice ad albero con chiave  $\langle \text{età}, \text{rnum} \rangle$ 
  - Quale è migliore?
  - Che succede se consideriamo la seconda interrogazione?

```
SELECT I.rnum, COUNT(*)  
FROM Imp I  
WHERE I.età = 30  
GROUP BY I.rnum
```

```
SELECT I.rnum, COUNT(*)  
FROM Imp I  
WHERE I.età > 30  
GROUP BY I.rnum
```

## *Piani basati solo sull'indice (segue)*

- Si possono trovare piani basati solo sull'indice anche per interrogazione che coinvolgono più di una tabella: ne parleremo più avanti

<I.rnum>

```
SELECT R.mgr
FROM Rep R, Imp I
WHERE R.rnum = I.rnum
```

<I.rnum, I.iid>

```
SELECT R.mgr, I.iid
FROM Rep R, Imp I
WHERE R.rnum = I.rnum
```



# Sommarrio

- ❖ Esistono molte organizzazioni di *file* alternative, ciascuna appropriata in certe situazioni
- ❖ Se le interrogazioni con selezione di intervallo sono frequenti, l'ordinamento del *file* o la costruzione di un indice sono importanti
  - Indici basati su *hash* sono buoni solo per ricerche con selezione di uguaglianza
  - *File* ordinati e indici basati su alberi sono migliori per ricerche con selezione di intervallo; buoni anche per ricerche con selezione di uguaglianza (nella pratica i *file* sono raramente mantenuti ordinati: è meglio un indice ad albero B+)
- ❖ Un indice è una collezione di *data entry* più un modo per trovare rapidamente le *data entry* quando vengono dati dei valori per la chiave.



## Sommario (segue)

- ❖ Le *data entry* possono in realtà essere *record* di dati, coppie <chiave, *rid*> oppure coppie <chiave, lista-di-*rid*>
  - La scelta è ortogonale alla tecnica di indicizzazione usata per localizzare le *data entry* con un dato valore per la chiave
- ❖ Si possono avere diversi indici su un dato *file* di *record* di dati, ciascuno con una diversa chiave di ricerca
- ❖ Gli indici possono essere classificati in *clustered* e non *clustered*, primari e secondari. Le differenze hanno conseguenze importanti sull'utilità e sulle prestazioni

# Sommario (segue)

- ❖ Capire la natura del carico di lavoro per l'applicazione, e gli obiettivi prestazionali, è essenziale per sviluppare un buon progetto
  - Quali sono le interrogazioni e gli aggiornamenti importanti? Quali attributi/relazioni sono coinvolti?
- ❖ Gli indici devono essere scelti per velocizzare le interrogazioni importanti (e magari qualche aggiornamento!)
  - Il mantenimento degli indici incide sugli aggiornamenti dei campi chiave
  - Scegliete indici che possono aiutare molte interrogazioni, se possibile
  - Costruite indici per supportare strategie basate solo sull'indice
  - L'uso del *clustering* è una decisione importante; soltanto un indice per ogni relazione può essere *clustered*!
  - L'ordine dei campi nella chiave di un indice composito può essere importante