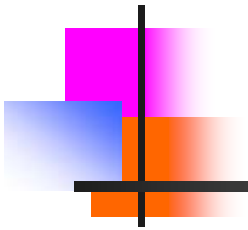


Indici

Sistemi Informativi L-B

Home Page del corso:
http://www-db.deis.unibo.it/courses/SIL-B/

Versione elettronica: Indici.pdf

Perché gli indici

- Le organizzazioni dei file viste (heap, sequenziale, hash) non permettono di ottenere prestazioni soddisfacenti se si eseguono ricerche usando valori di attributi che non sono usati per determinare il criterio di allocazione dei record nel file
- Anche nel caso di file sequenziali, una ricerca per valore di chiave richiede comunque un numero di operazioni di I/O pari a $\lceil \log_2 NP \rceil$, e quindi elevato per file di grandi dimensioni

- Per ovviare a questi limiti si creano degli indici che forniscono "cammini di accesso" alternativi ai dati

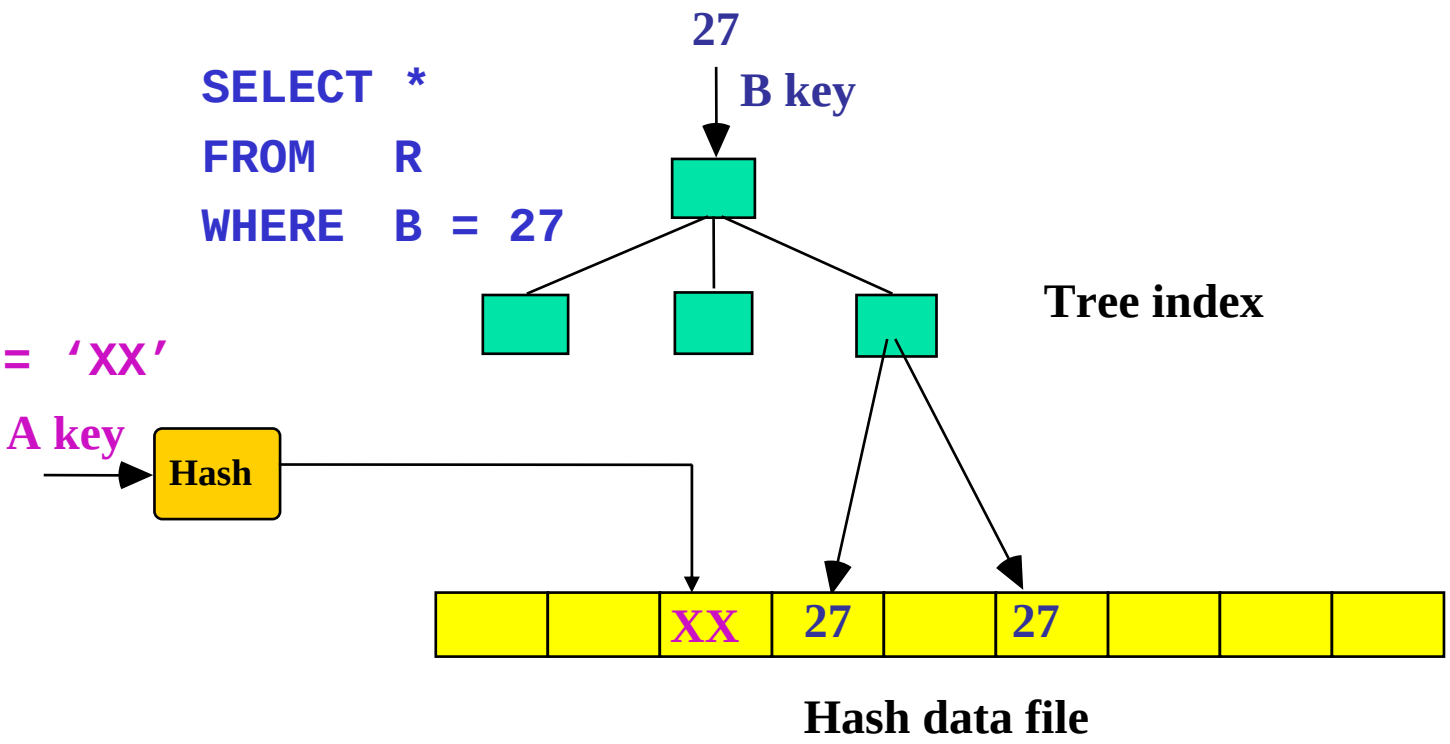
Cammini di accesso

- La costruzione di indici su una relazione mette a disposizione modalità alternative (cammini di accesso) per localizzare velocemente i dati di interesse

NB Si usa comunemente il termine (valore di) chiave (di ricerca) per indicare il valore di un campo usato per selezionare i record (es. B è una chiave)

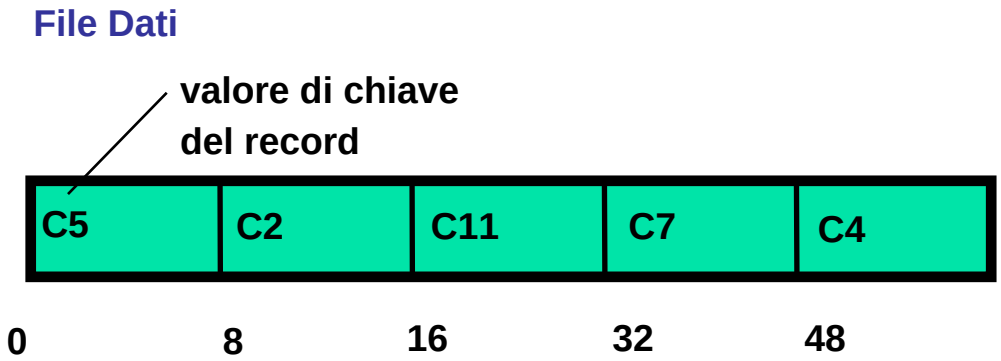
```
SELECT *
FROM   R
WHERE  B = 27
```

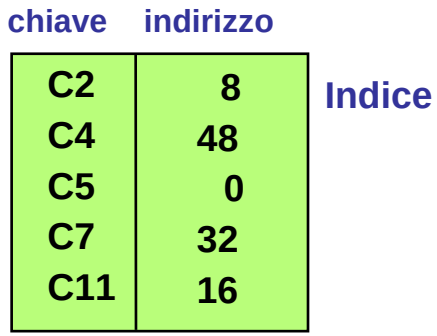
```
SELECT *
FROM   R
WHERE  A = 'XX'
```



Indici: principio di base

- Logicamente, un indice può essere visto come un insieme di coppie del tipo (k_i,p_i) dove:
  - k_i è un valore di chiave del campo su cui l'indice è costruito
  - p_i è un puntatore al record (eventualmente il solo) con valore di chiave k_i. Nei DBMS è quindi un RID o, al limite, un PID
- Il vantaggio di usare un indice nasce dal fatto che la chiave è solo parte dell'informazione contenuta in un record. Pertanto, l'indice occupa uno spazio minore rispetto al file dati
- I diversi indici differiscono essenzialmente nel modo con cui organizzano l'insieme di coppie (k_i,p_i)

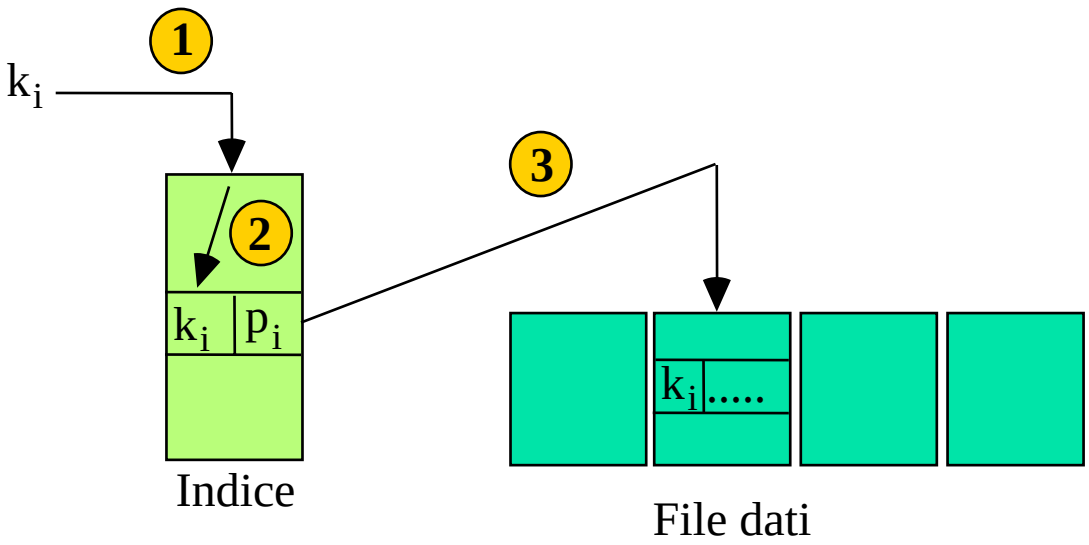




Accesso con indice: schema generale

- Si consideri un indice su chiave primaria usato per ricercare il record con chiave k_i. I passi da compiere sono:

1. Accedere all'indice
2. Ricercare la coppia (k_i, p_i)
3. Accedere alla pagina dati relativa



Nonostante occupi un minor spazio rispetto al file dati, un indice può raggiungere notevoli dimensioni che determinano problemi di gestione simili a quelli del file dati

Esempio:
un indice per un file di 500K record, in cui i valori di chiave sono stringhe di 20 byte e i puntatori sono di 4 byte, richiede circa 12 MB

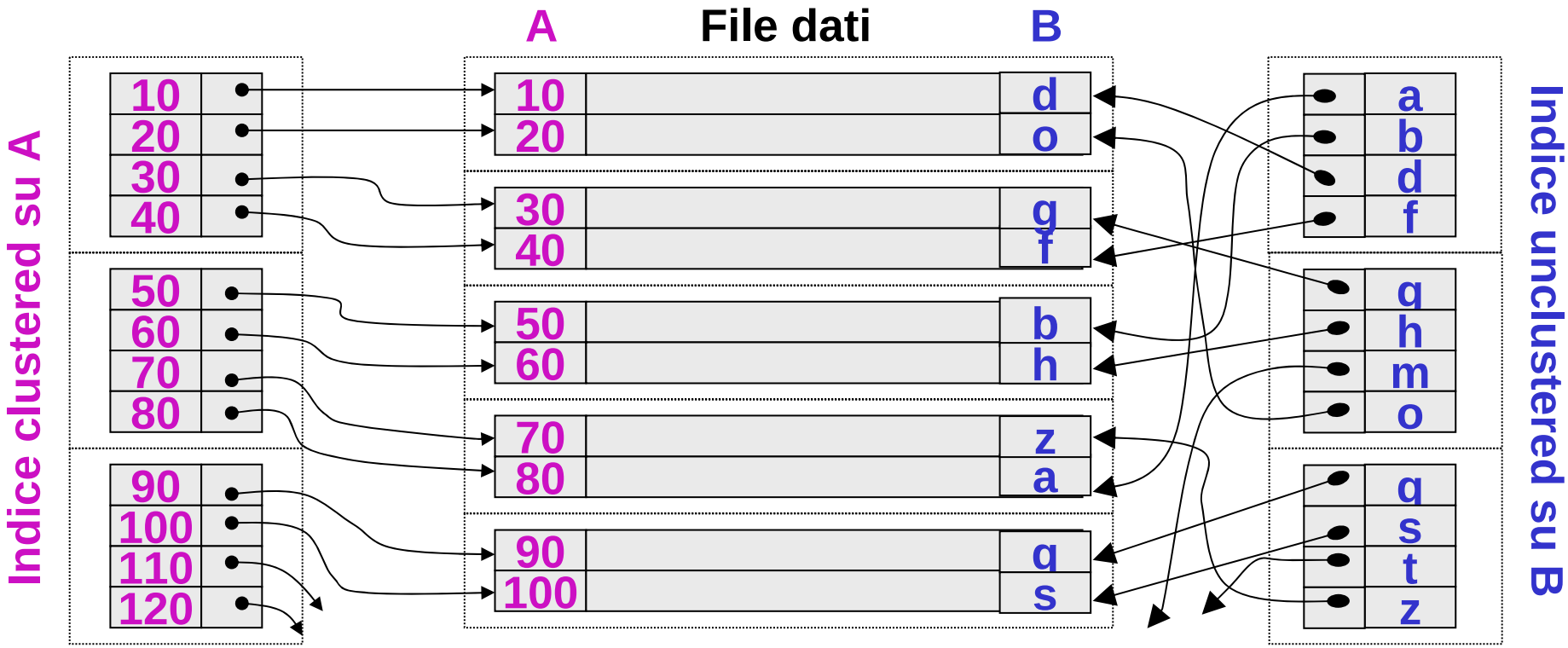
Tipi di indici

- Esistono diverse tipologie di indici; la prima distinzione è tra
  - Indici ordinati: i valori di chiave k_i vengono mantenuti ordinati, in modo da poter essere reperiti più efficientemente
  - Indici hash: si usa una funzione hash per determinare la posizione dei valori di chiave k_i; questi indici tuttavia non forniscono prestazioni soddisfacenti per ricerche di intervallo
- Nel seguito vediamo solo gli indici ordinati, che a loro volta si dividono in:
  - Clustered o unclustered
  - Primary o secondary
  - Single-level o multi-level
  - Dense o sparse

- La terminologia non è standard, ad esempio qualcuno chiama primary gli indici che noi chiamiamo clustered e secondary gli unclustered

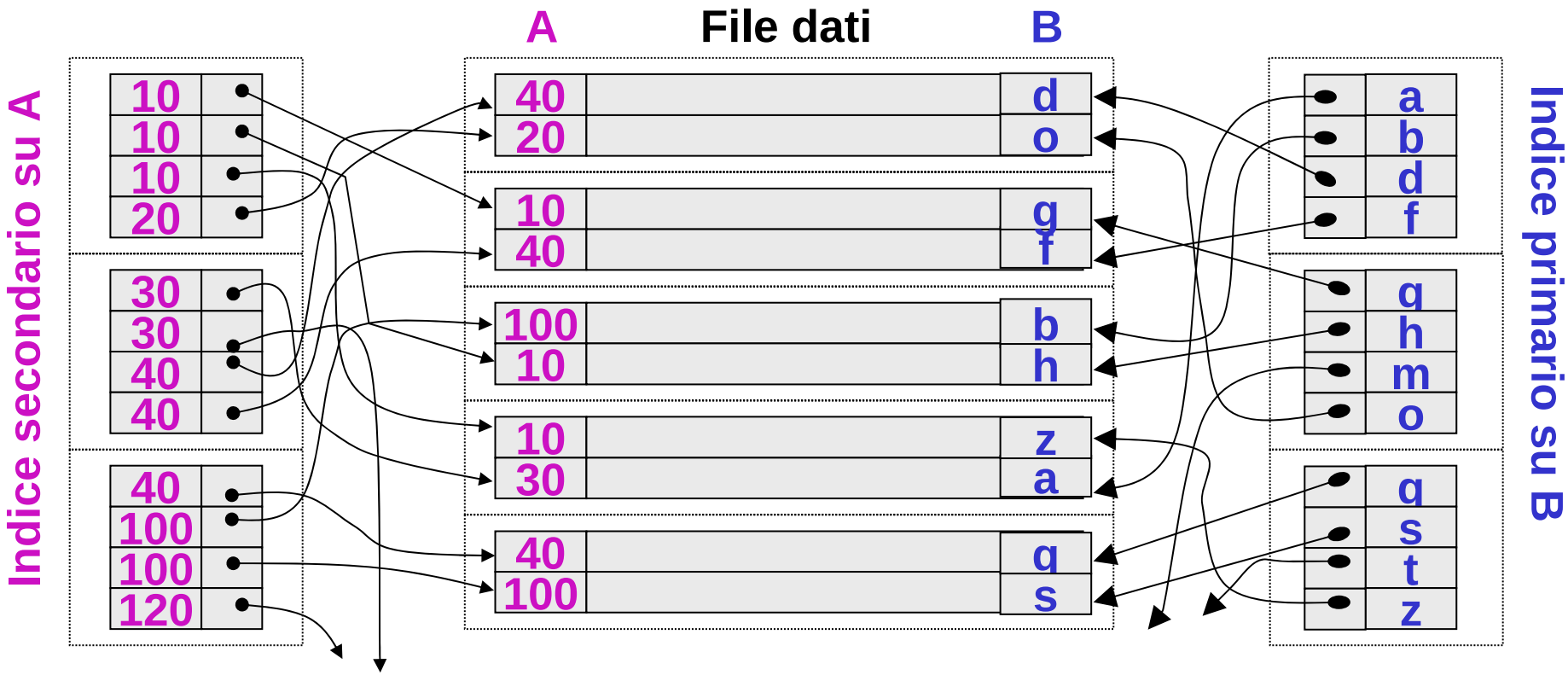
Indici clustered e unclustered

- Un indice è detto clustered se è costruito sul campo su cui i record nel file dati sono mantenuti ordinati, altrimenti è detto unclustered
- Ovviamente si può costruire al massimo un indice clustered per relazione, mentre si può costruire un arbitrario numero di indici unclustered



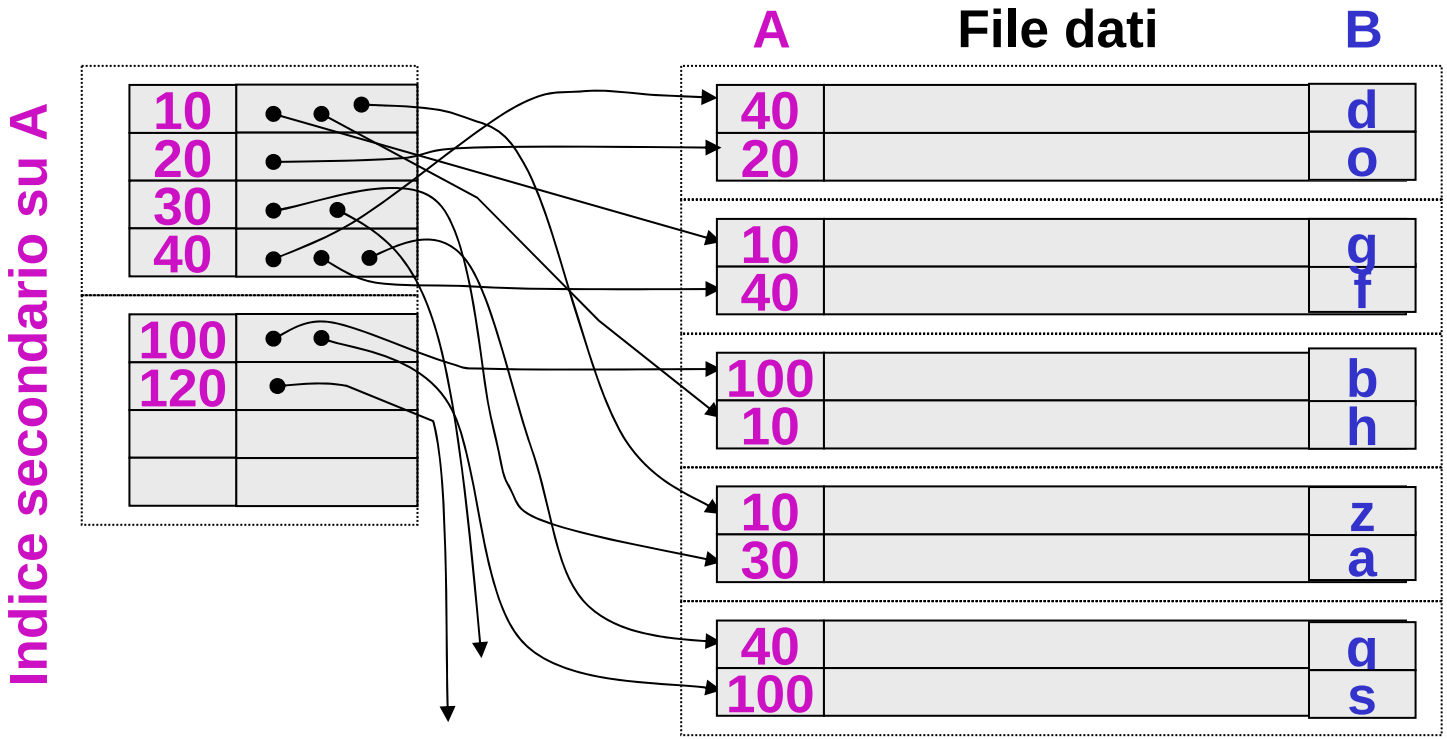
Indici primary e secondary

- Un indice è detto primary (primario) se è costruito su un campo a valori non ripetuti (chiave relazionale), altrimenti è detto secondary (secondario)



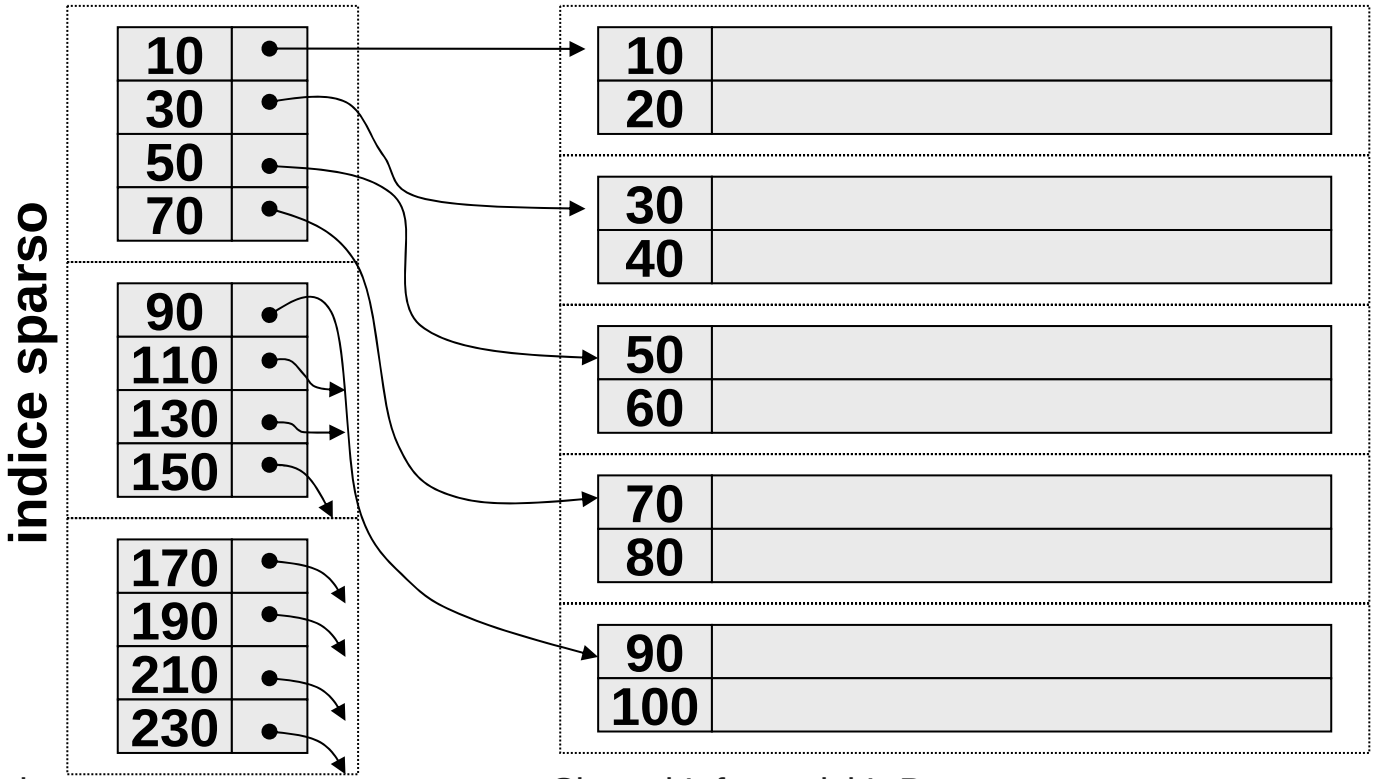
Indici secondari a liste di puntatori

- Per evitare di replicare inutilmente i valori di chiave, la soluzione più comunemente adottata per gli indici secondari consiste nel raggruppare tutte le coppie con lo stesso valore di chiave in una lista di puntatori



Indici dense e sparse

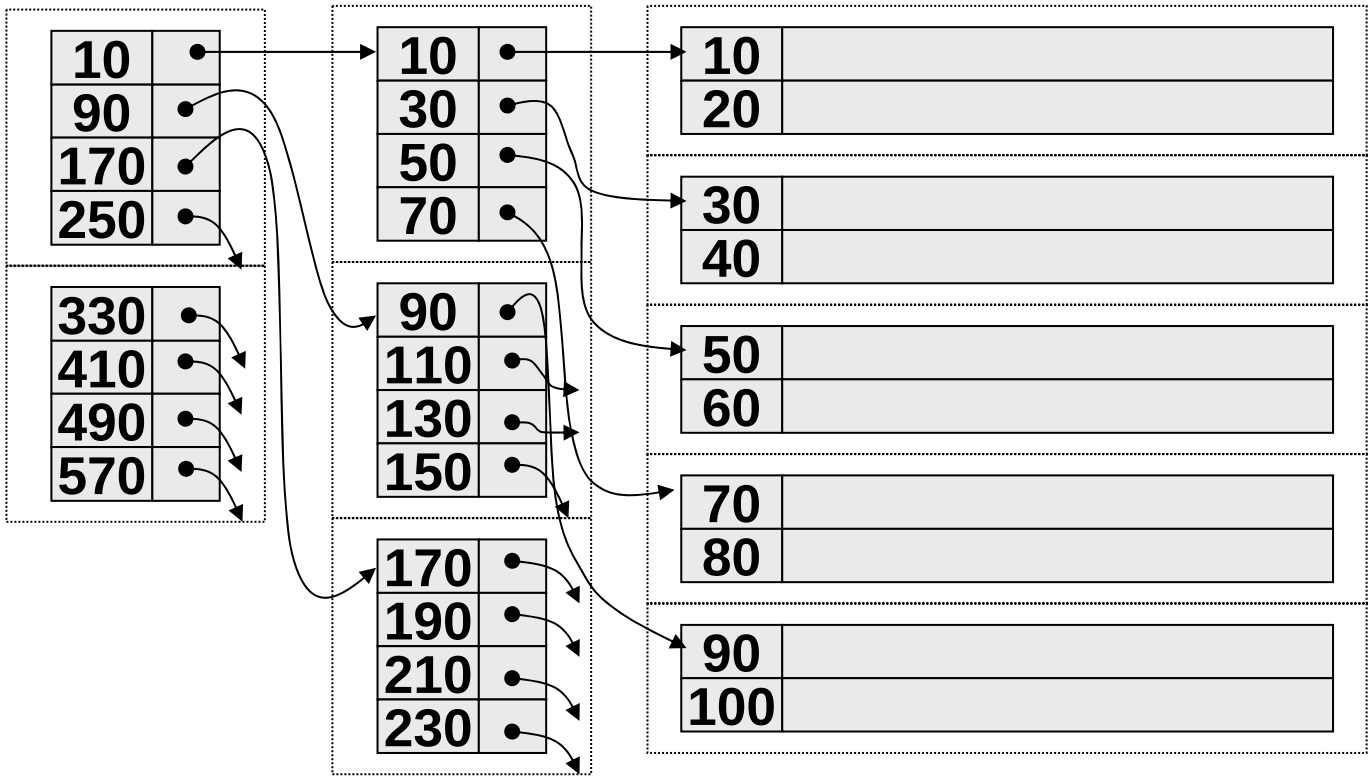
- Negli indici dense (densi) il numero di puntatori è pari al numero di record del file dati, negli indici sparse (sparsi) è minore (tipicamente uno per pagina dati)
  - È una soluzione applicabile con indici clustered



Indici single-level e multi-level

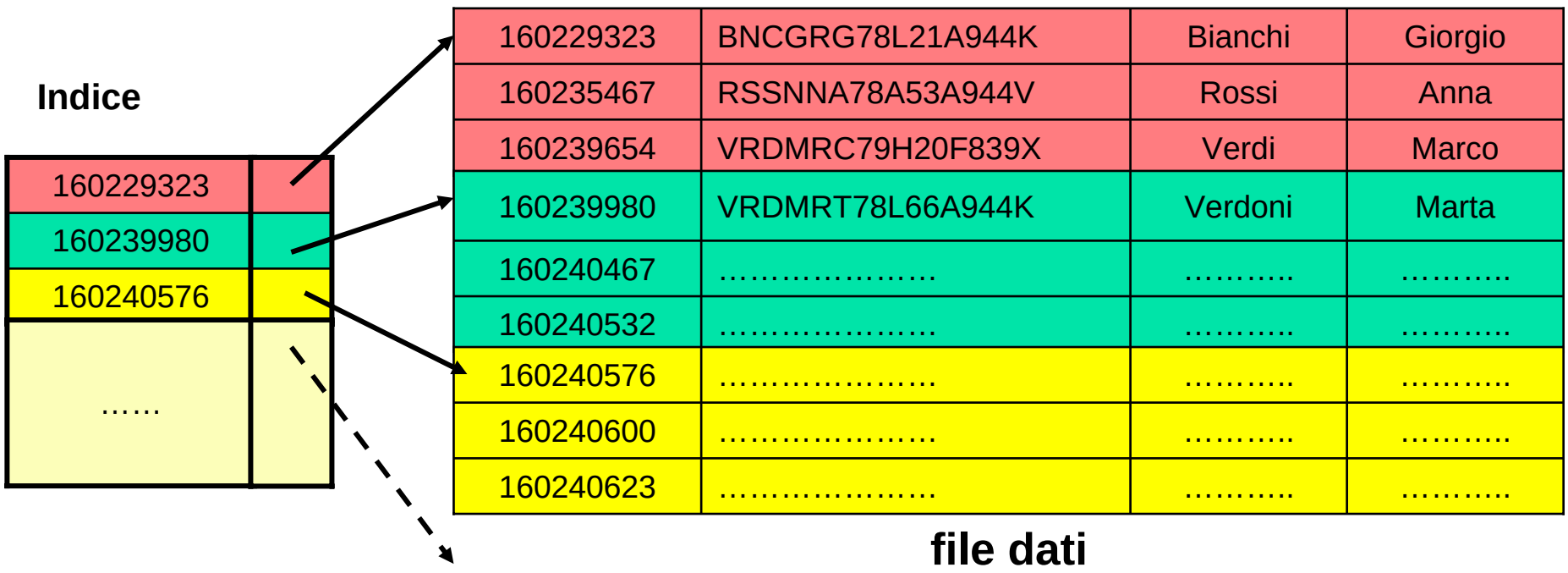
- Gli esempi visti sono tutti relativi a indici single-level (o “flat”)
- È tuttavia possibile “indicizzare l’indice” usando un indice (sparso!), e così via ricorsivamente, in modo da creare una struttura multi-livello (albero)

Indice a 2 livelli



Esempio (1)

primary clustered sparse single-level index



Esempio (2)

secondary unclustered dense single-level index

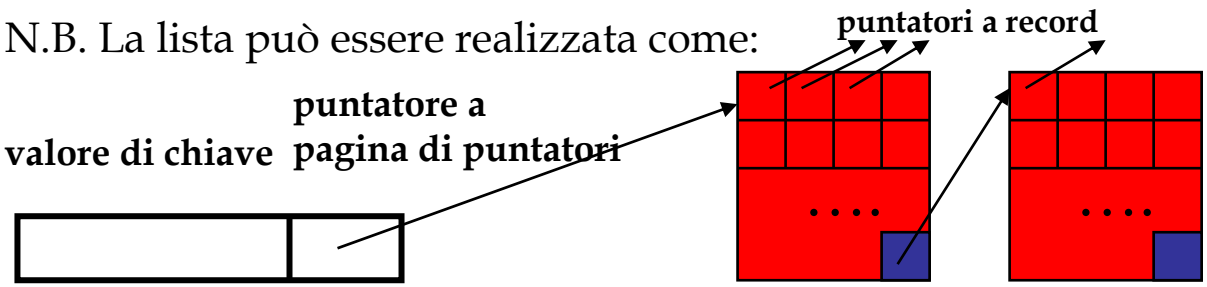
pagine dati

Lista di RID

Alessandri	
Bianchi	
Bianchini	
… … …	
Rossi	
……	
Verdi	
Verdoni	
…… …	

160229323	BNCGRG78L21A944K	Bianchi	Giorgio
160235467	RSSNNA78A53A944V	Rossi	Anna
160239654	VRDMRC79H20F839X	Verdi	Marco
160239980	VRDMRT78L66A944K	Verdoni	Marta
160240467	……	Alessandri	Maria
160240532	……	Bianchini	Carlo
……	……	……	……
160240576	……	Rossi	Demetrio
160240600	……	Bianchi	Arrigo
160240623	……	Verdi	Remo

N.B. La lista può essere realizzata come:



Indici in SQL

- In SQL la definizione di indici avviene mediante lo statement CREATE INDEX (ma non è standardizzato!). In DB2:

```
CREATE INDEX VotoIDX              -- secondary unclustered index
ON Esami(Voto DESC)               -- ASC è il default

CREATE UNIQUE INDEX MatrIDX       -- primary unclustered index
ON Studenti(Matricola)

CREATE INDEX VotoIDX              -- clustered index
ON Esami(Voto DESC) CLUSTER

CREATE INDEX Anagrafica
ON Persone(Cognome,Nome)          -- indice multi-attributo
```

Indici multi-attributo

- Un indice multi-attributo costruito su A1,A2,…An è efficace se si specificano valori per tutti gli attributi o per i primi i < n, ovvero

  l'ordine di definizione è rilevante!

- Ad esempio, l'indice EsamiIDX:

```
CREATE INDEX EsamiIDX
ON Esami(CodCorso,Data,Voto)
```

  è utile solo per query che specificano valori per
  - tutti e tre gli attributi
  - per CodCorso e Data
  - solo per CodCorso

- Pertanto se la clausola WHERE contenesse solo i predicati

```
Data = '28-06-2002' AND Voto = 30
```

  l'indice non verrebbe utilizzato dal DBMS!

Le statistiche sugli indici

- Per poter capire quando l'uso di un indice è conveniente (non lo è sempre!), il DBMS mantiene un catalogo con statistiche sugli indici, in particolare:

SQL Catalog	SQL attribute	Descrizione	Simbolo
SYSSTAT.INDEXES	NLEAF	Numero di pagine foglia (ossia del livello più basso) dell'indice	NL o NL(index)
SYSSTAT.INDEXES	NLEVELS	Numero di livelli dell'indice	h o h(index)
SYSSTAT.INDEXES	FULLKEYCARD	Numero di valori distinti di chiave nell'indice	NK o NK(index)

Esercizio riassuntivo (1)

- Consideriamo ancora la relazione Studenti, che consiste di NT = 200000 record di 160 byte l'uno, organizzati in un file sequenziale ordinato su matricola. Le pagine hanno dimensione P = 4 KB e il disco ha le seguenti caratteristiche:
  - Rotational latency: Tr = 8.5 ms
  - Seek time: Ts = 12 ms
  - Page transfer time: Tt = 1 ms
- Sono presenti anche due indici:

```
CREATE UNIQUE INDEX MatrIDX
ON Studenti(matricola) CLUSTER

CREATE INDEX DataNascIDX
ON Studenti(datanascita)
```

Esercizio riassuntivo (2)

B) Dimensione del file dati e degli indici:

per il file sequenziale e per gli indici ogni pagina viene riempita solo all'85%, il page header e la directory lasciano a disposizione uno spazio pari a 4000 byte in ogni pagina

Gli indici supponiamo siano a un livello e i puntatori siano RID di 4 byte

L'indice MatrIDX è primario (e denso), quindi NK = 200000 con ogni coppia che occupa (10 + 4) byte

L'indice DataNascIDX è secondario; supponendo NK = 4000, ad ogni valore di chiave corrispondono in media 50 record; quindi valore di chiave e puntatori occupano (10 + 50*4) = 210 byte

Struttura	Numero pagine dati (NP) o foglia (NL)	Dimensione file
File dati	$\lceil 200000/\lfloor 0.85*4000 / 160 \rfloor \rceil = 9524$	37.20 MB
MatrIDX	$\lceil 200000/\lfloor 0.85*4000 / 14 \rfloor \rceil = 827$	3.23 MB
DataNascIDX	$\lceil 4000/\lfloor 0.85*4000 / 210 \rfloor \rceil = 250$	0.98 MB

Esercizio riassuntivo (3)

C) Ricerca per matricola:

Usando MatrIDX le letture sono random in quanto si fa ricerca binaria

Si lasciano per raffronto i tempi relativi alle organizzazioni heap e hash

Cammino d'accesso	Costo di ricerca
Heap	Ts + Tr + NP/2 * Tt = 4020.5 ms
Hash	Ts + Tr + Tt = 21.5 ms
Sequenziale	$\lceil \log_2(NP) \rceil$ * (Ts + Tr + Tt) = **301 ms**
MatrIDX	$\lceil \log_2(NL) \rceil$ * (Ts + Tr + Tt) + (Ts + Tr + Tt) = **215 ms**

Il vantaggio che si ottiene usando un indice clustered è significativo, ma ancora lontano dalle prestazioni del file hash, in quanto si leggono comunque in modo random molte pagine dell'indice

Esercizio riassuntivo (4)

D) Ricerca per intervallo di data di nascita

Si cercano tutti gli studenti nati in un certo periodo in cui vi sono 80 date di nascita (e quindi 4000 studenti)

Ogni foglia dell'indice contiene 16 valori di chiave, quindi oltre alla foglia individuata dalla ricerca binaria bisogna leggerne altre 4 (in sequenza)

Cammino d'accesso	Costo di ricerca per chiave
Sequenziale	Ts + Tr + NP * Tt = **9544.5 ms**
DataNascitaIDX	$(\lceil \log_2(NL) \rceil * (Ts + Tr + Tt) + 4 * Tt$ + 4000 * (Ts + Tr + Tt) = **86176 ms**

L'uso di un indice unclustered può risultare svantaggioso
se il numero dei record da reperire è elevato,
in quanto ogni accesso al file dati comporta una lettura random

Indici: altri usi importanti

- Oltre che per risolvere predicati di ricerca, un indice può essere usato per accedere ai record secondo un certo ordine, e quindi può anche essere utile se nella query ci sono clausole ORDER BY e GROUP BY

```
SELECT   *
FROM     Studenti
ORDER BY datanascita
```

- Per alcune query la presenza di un indice può addirittura evitare di dover accedere al file dati!

```
SELECT   COUNT(DISTINCT datanascita)
FROM     Studenti
```

Se esiste un indice su (DeptNo, Salary), anche la seguente query non ha bisogno di accedere al file dati:

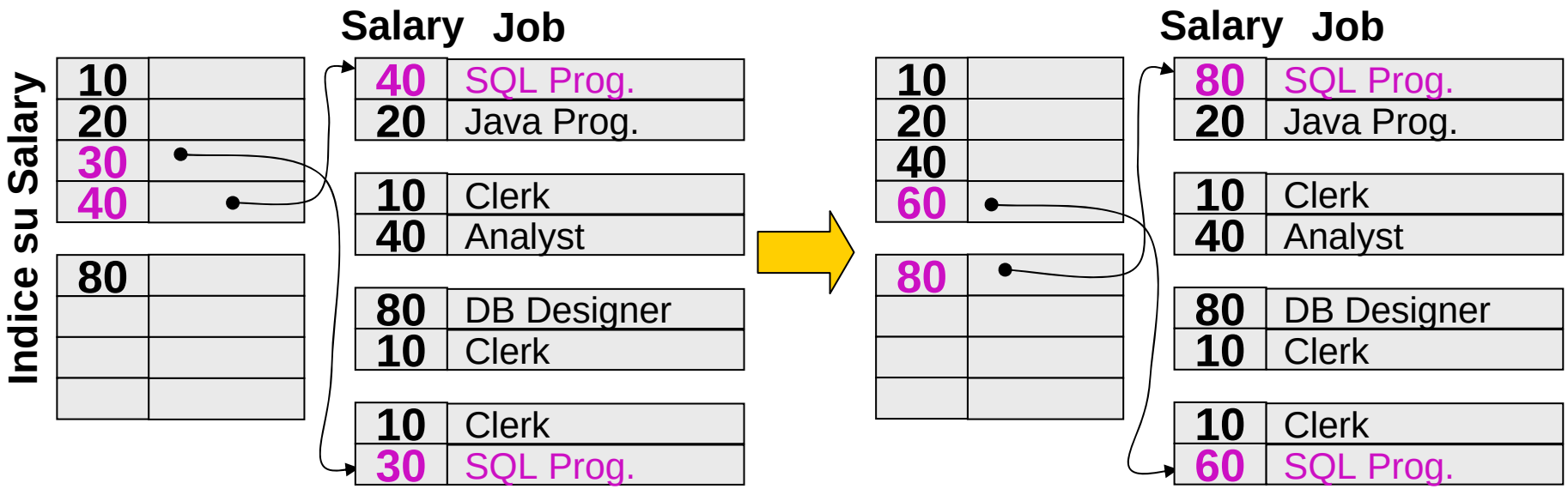
```
SELECT   DeptNo, MAX(Salary)
FROM     Employee
GROUP BY DeptNo
```

Indici e aggiornamenti

- Poiché ogni indice deve riflettere la situazione corrente nel file dati, è evidente che ogni operazione di modifica dei dati si deve propagare anche agli indici interessati. Ad esempio:

```
UPDATE   Employee
SET      Salary = 2 * Salary
WHERE    Job = 'SQL Programmer'
```

richiede che si modifichino anche tutti gli indici costruiti su Salary

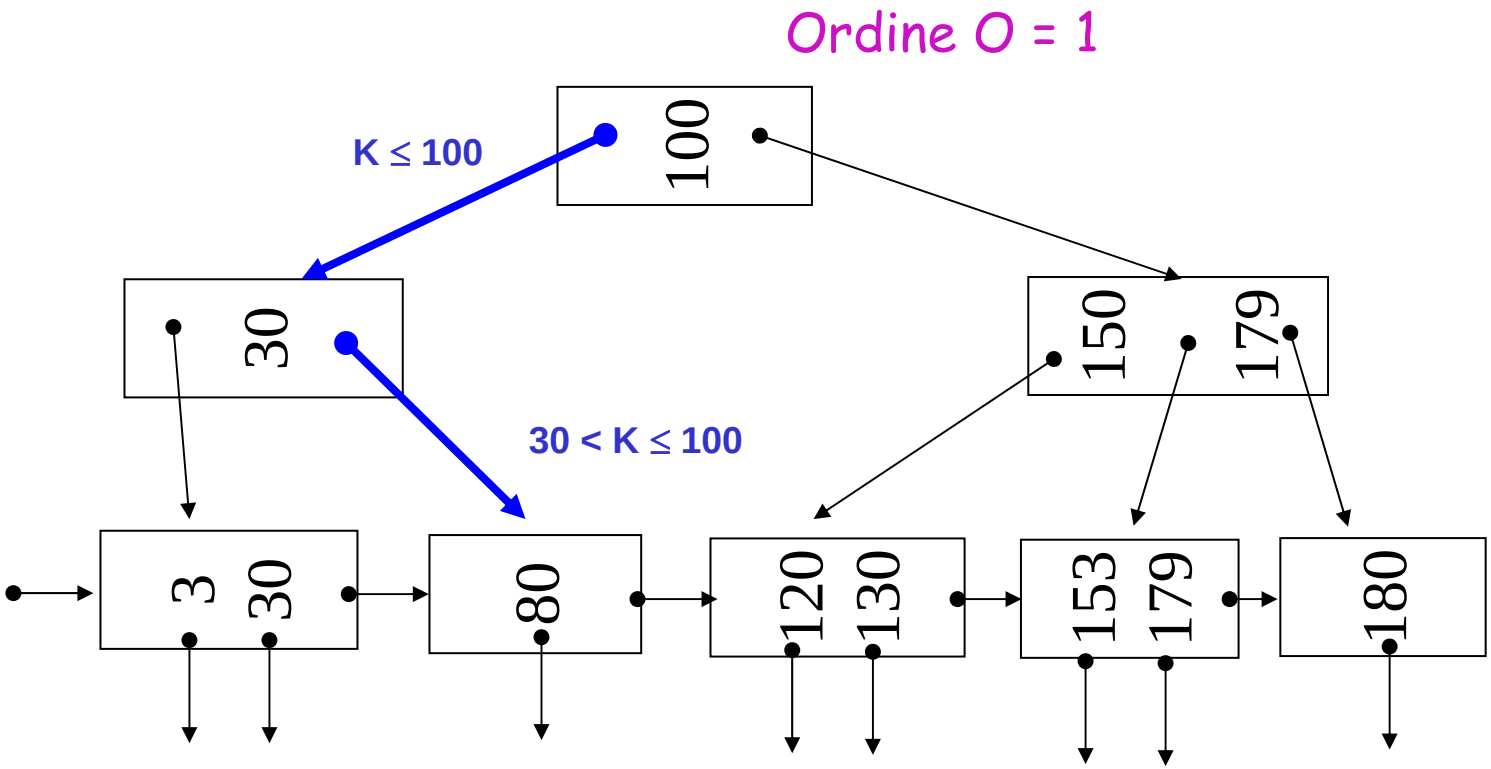


B^+-tree

- Il B^+-tree è la struttura comunemente usata nei DBMS per realizzare indici multi-livello dinamici, ossia che non richiedono riorganizzazioni periodiche
  - Il B^+-tree è una variante del B-tree (Bayer McCreight, 72)
- Le principali caratteristiche di un (primary) B^+-tree sono:
  - Le coppie (k_i,p_i) sono tutte contenute in pagine (o "nodi") foglia e le foglie sono collegate a lista mediante puntatori (PID) per favorire la risoluzione di query di intervallo
  - Ogni foglia contiene un numero di coppie che varia tra O e 2 * O, dove **O** è detto "**ordine**" del B^+-tree
  - Ogni percorso dal nodo radice a una foglia ha lunghezza **h** (altezza del B^+-tree), quindi l'albero è perfettamente bilanciato
  - Ogni nodo intermedio (né radice né foglia) ha almeno O + 1 puntatori ad altrettanti nodi figli; la radice, se non è una foglia, ha almeno 2 figli
  - Un nodo con F + 1 figli contiene F valori di chiave, detti anche "**separatori**"
  - Ogni nodo ha al più 2 * O + 1 nodi figli

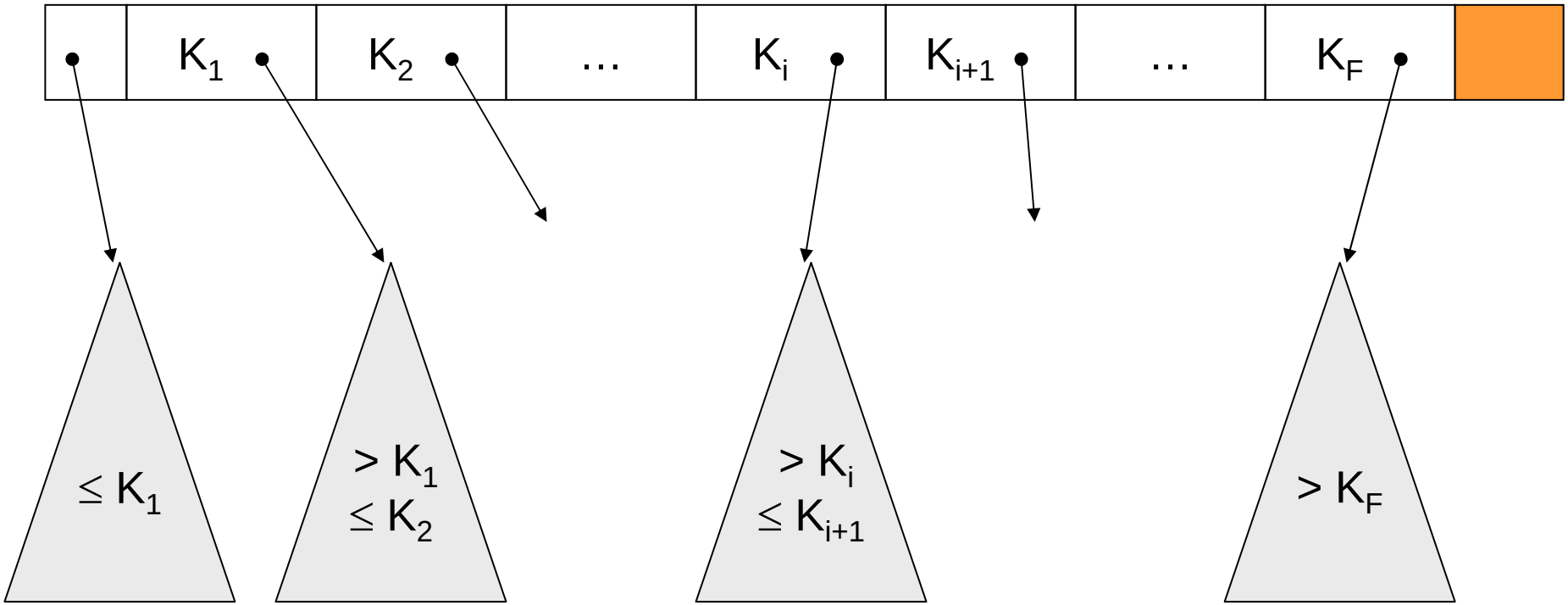
Esempio di B$^+$-tree

- La figura mostra un esempio di B$^+$-tree di altezza h = 3



Formato dei nodi interni

- I nodi interni hanno il formato mostrato in figura, in cui è $K_1 < K_2 < \ldots < K_F$



Ordine e altezza

- L'altezza h è una funzione logaritmica di NK (numero di valori di chiave indicizzati); in particolare si può dimostrare che vale la relazione

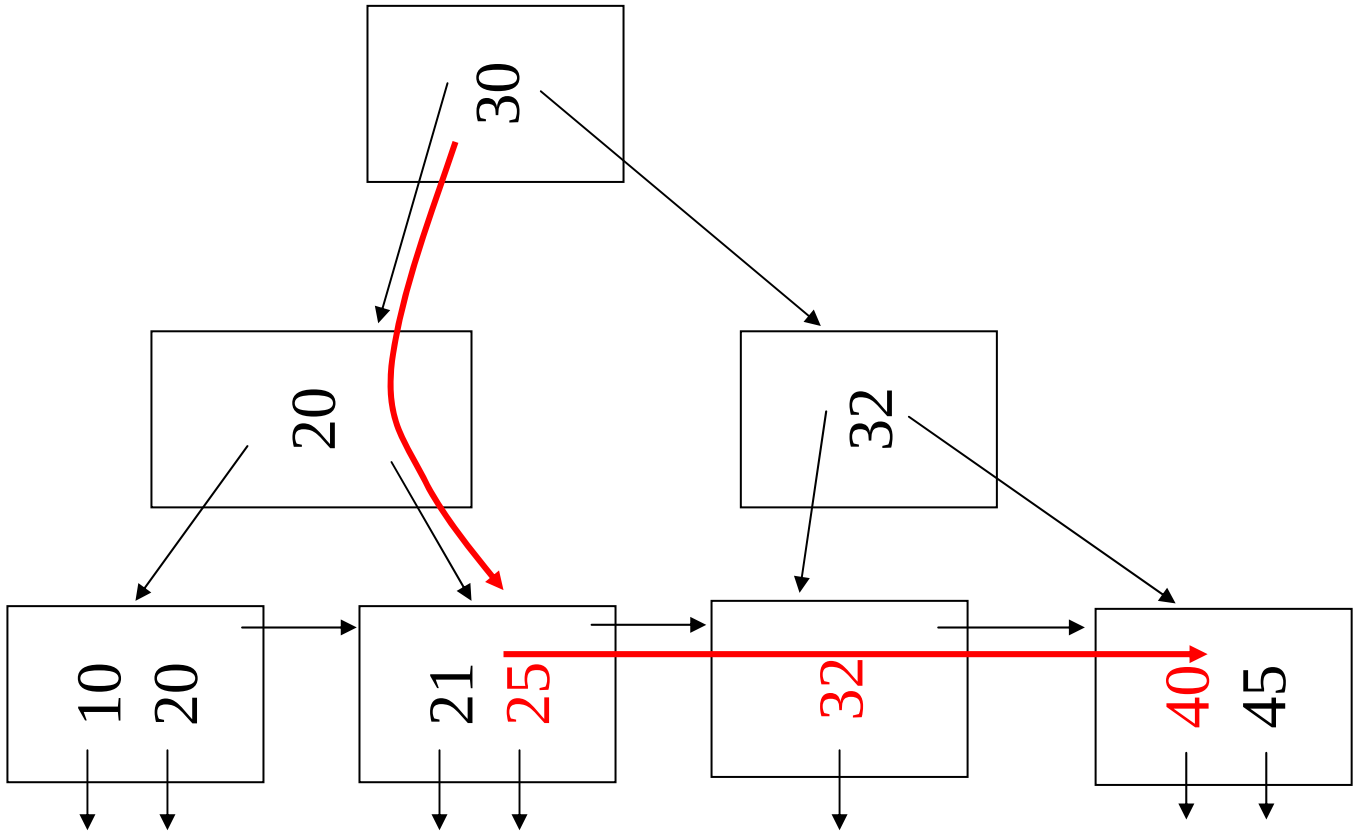
$$1+\left\lceil \log_{2*O+1} \frac{NK}{2*O} \right\rceil \leq h \leq 2+\left\lfloor \log_{O+1} \frac{NK}{2*O} \right\rfloor$$

- Per avere un'idea del valore di O, con valori di chiave di 8 byte, RID di 4 byte e pagine con disponibili 4000 byte si ottiene O = 166
- Pertanto se NK = 10^9, la ricerca di un valore di chiave (che accede a un nodo per ogni livello) richiede al massimo $2 + \lfloor \log_{167} (10^9 / 332) \rfloor = 4$ operazioni di I/O!
  - Per contro, una ricerca binaria richiederebbe 22 accessi a disco, supponendo di avere le pagine indice piene
- La variabilità di h è, fissati NK e O, molto limitata (differenza di 1 tra minima e massima)
- Con questi valori di O, con h = 3 si gestiscono fino a circa 3 milioni e mezzo di chiavi

Ricerche per intervallo

- Per cercare tutti i valori di chiave nell'intervallo [L,H] si cerca innanzitutto il valore L, quindi si prosegue in sequenza sulle foglie

  Esempio: si cercano le chiavi nell'intervallo [23,42]



Inserimento

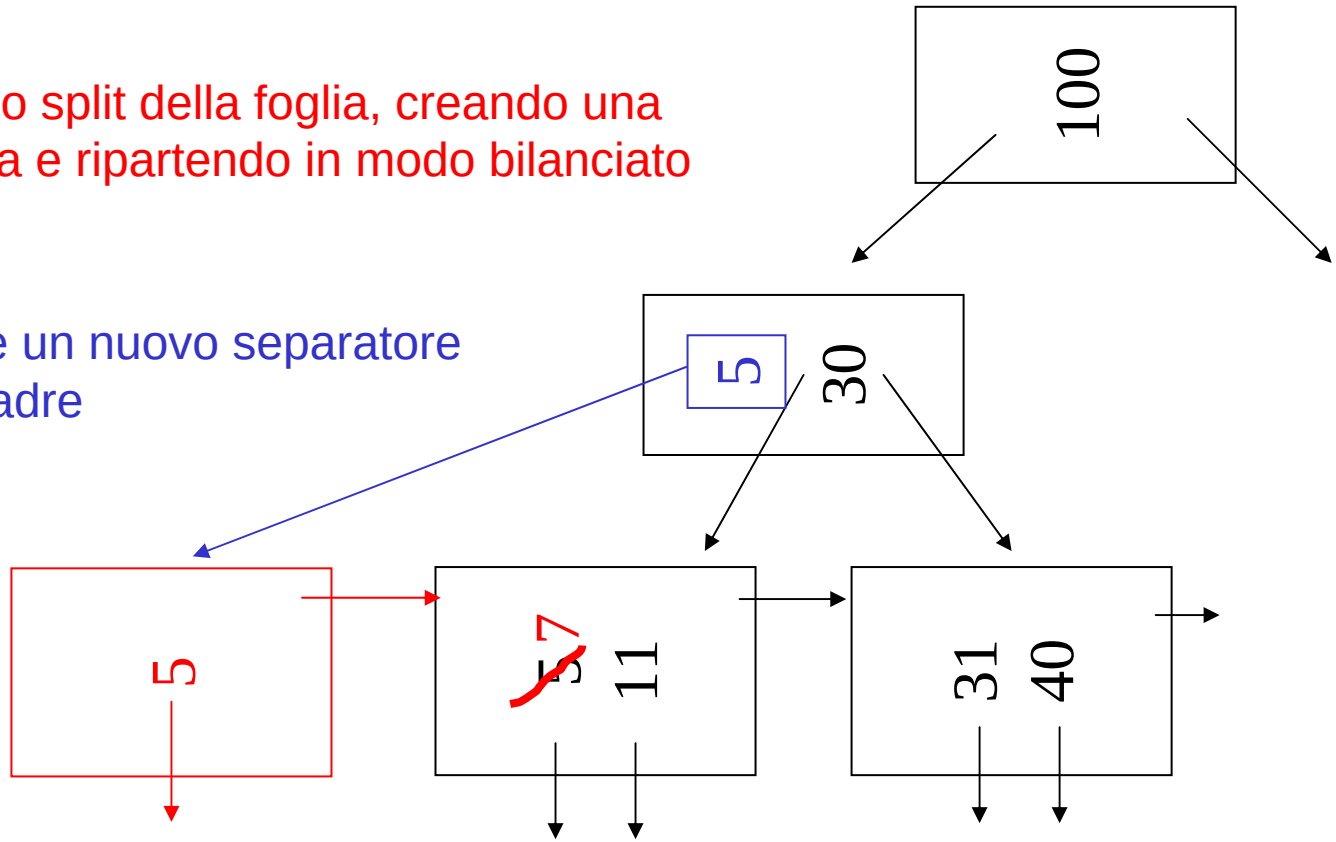
- La procedura di inserimento procede innanzitutto cercando la foglia in cui inserire il nuovo valore di chiave
- Se c'è posto la nuova coppia (k_i,p_i) viene inserita nella foglia e la procedura termina
- Se non c'è più posto, si attiva una procedura di "split" ricorsiva che, al limite, si propaga fino alla radice
- La procedura di cancellazione segue una logica simile (qui non descritta)

Inserimento: esempio (1)

- Si inserisce la chiave 7

Si esegue lo split della foglia, creando una nuova foglia e ripartendo in modo bilanciato le chiavi

Si inserisce un nuovo separatore nel nodo padre



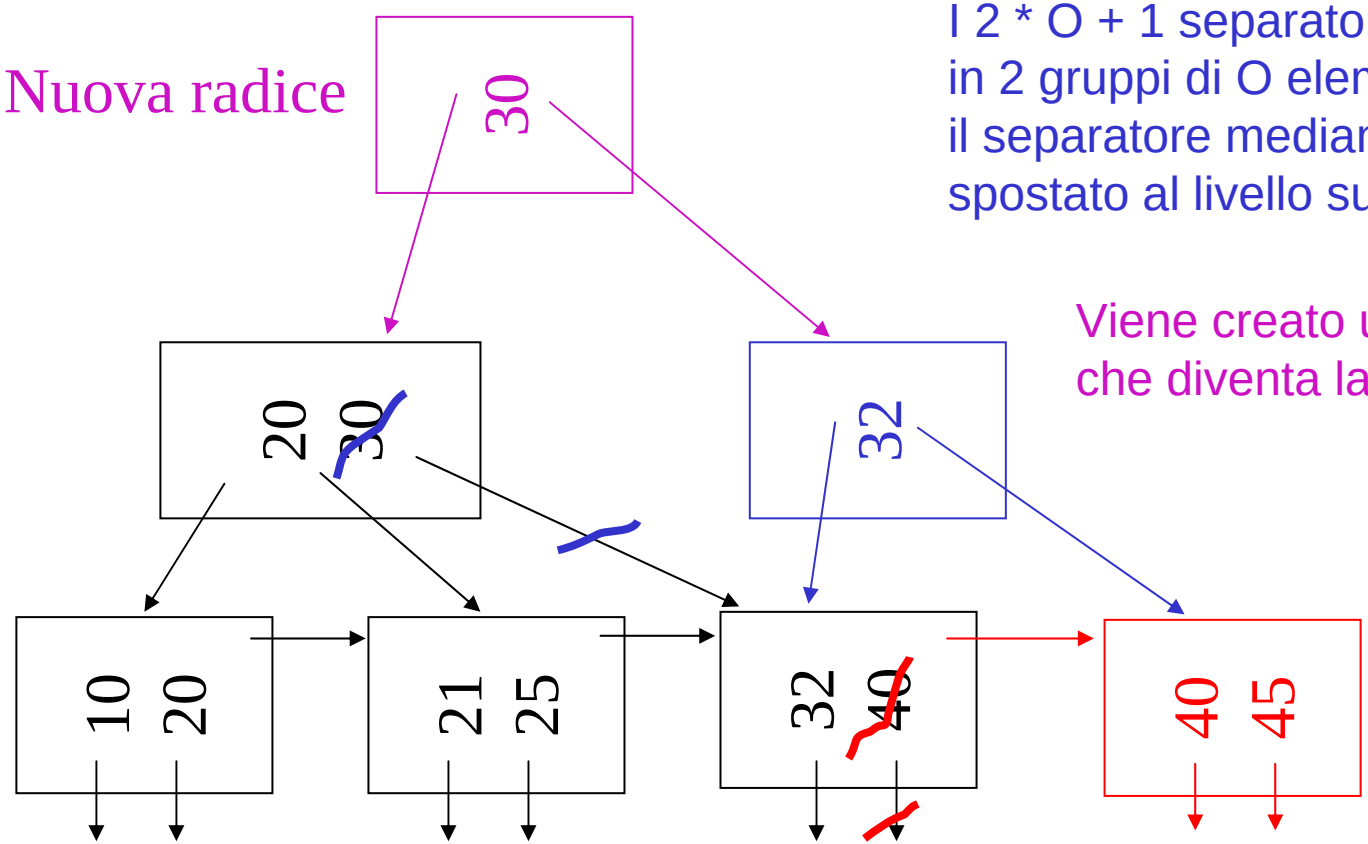
Inserimento: esempio (2)

- Si inserisce la chiave 45

Si esegue lo split della foglia

Il nuovo separatore (32)
attiva lo split del nodo padre:
I 2 * O + 1 separatori vengono divisi
in 2 gruppi di O elementi ciascuno,
il separatore mediano (30) viene
spostato al livello superiore

Viene creato un nuovo nodo,
che diventa la radice dell'albero



Confronto con le altre tecniche (1)

- Consideriamo ancora la relazione Studenti, che consiste di NT = 200000 record di 160 byte l'uno, organizzati in un file sequenziale ordinato su matricola. Le pagine hanno dimensione P = 4 KB e il disco ha le seguenti caratteristiche:
  - Rotational latency: Tr = 8.5 ms
  - Seek time: Ts = 12 ms
  - Page transfer time: Tt = 1 ms
- Sono presenti anche due indici:

```
CREATE UNIQUE INDEX MatrIDX
ON Studenti(matricola) CLUSTER

CREATE INDEX DataNascIDX
ON Studenti(datanascita)
```

Confronto con le altre tecniche (2)

- Confrontiamo le prestazioni ottenibili usando B+-tree per le query sulla relazione Studenti viste in precedenza:

A) Dimensione dei B^+-tree:

Consideriamo il caso peggiore, in cui ogni nodo diverso dalla radice è riempito al 50%

Per l'indice MatrIDX, si ha O = 142, e si deriva che è h = 3

Per l'indice DataNascIDX, supponendo sempre NK = 4000 e usando liste di puntatori nelle foglie (50 RID per valore di chiave), si ha O = 9 con h = 4

Struttura	Numero pagine dati (NP) o foglia (NL)	Dimensione file
File dati	$\lceil 200000/\lfloor 0.85*4000 / 160 \rfloor \rceil = 9524$	37.20 MB
MatrIDX B^+-tree	$\lceil 200000/ 142 \rceil = 1409$	5.54 MB
DataNascIDX B^+-tree	$\lceil 4000/ 9 \rceil = 445$	1.93 MB

Confronto con le altre tecniche (3)

C) Ricerca per matricola:

Usando MatrIDX si leggono h = 3 pagine indice e 1 pagina dati

Si lasciano per raffronto i tempi relativi alle organizzazioni heap e hash

Cammino d'accesso	Costo di ricerca
Heap	Ts + Tr + NP/2 * Tt = 4020.5 ms
Hash	Ts + Tr + Tt = 21.5 ms
Sequenziale	$\lceil \log_2(NP) \rceil$ * (Ts + Tr + Tt) = **301 ms**
MatrIDX B^+-tree	(h + 1) * (Ts + Tr + Tt) = **86 ms**

Confronto con le altre tecniche (4)

D) Ricerca per intervallo di data di nascita

Si cercano tutti gli studenti nati in un certo periodo in cui vi sono 80 date di nascita (e quindi 4000 studenti)

Ogni foglia contiene 9 valori di chiave, quindi oltre alla foglia individuata bisogna leggerne altre 8 (le letture sono random perché non c'è nessuna garanzia che i nodi foglia siano allocati in modo contiguo sul disco)

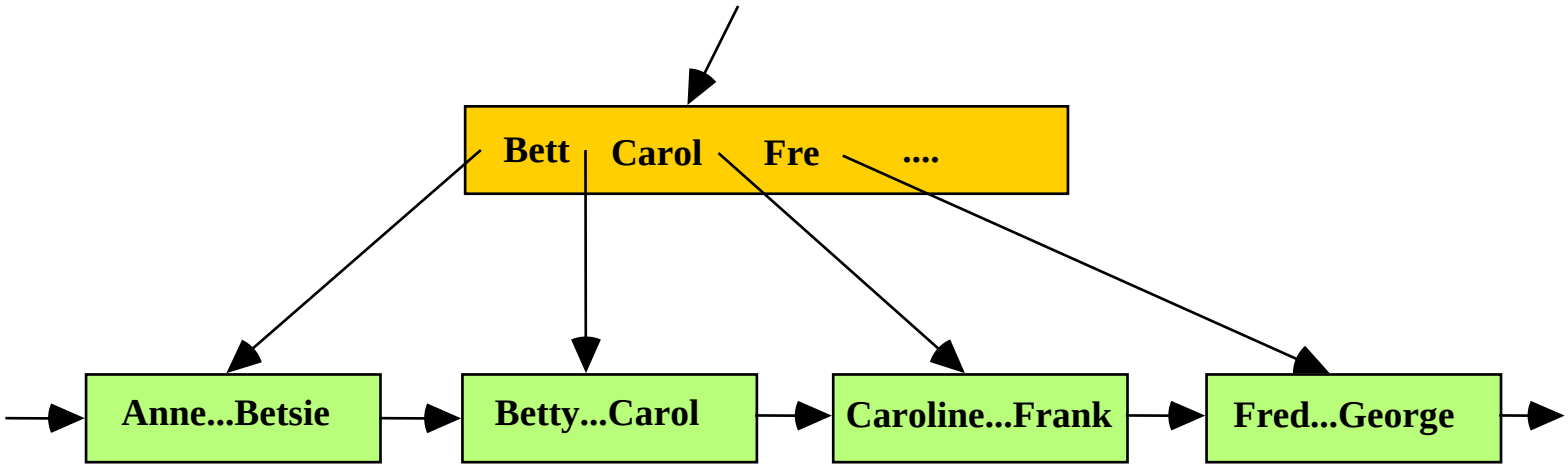
Cammino d'accesso	Costo di ricerca per chiave
Sequenziale	Ts + Tr + NP * Tt = **9544.5 ms**
DataNascitaIDX B^+-tree	(h + 8) * (Ts + Tr + Tt) + 4000 * (Ts + Tr + Tt) = **86258 ms**

Anche in questo caso l'uso dell'indice non è conveniente, in quanto è prevalente il costo di accesso al file dati

B+-tree per chiavi a lunghezza variabile

- Nel caso di chiavi a lunghezza variabile l'ordine non può essere più definito in modo esatto, ma mantiene un significato solamente in senso medio
- D'altronde con chiavi a lunghezza varibile è possibile comprimere i separatori presenti nei nodi interni (non foglia)

  Esempio: Per separare "Frank" da "Fred" è sufficiente la stringa "Fre"



Riassumiamo:

- L'utilizzo degli indici permette di accedere ai record di un file secondo modalità alternative (cammini di accesso)
- Organizzazioni degli indici di tipo single-level presentano sia problemi prestazionali che di gestione, per quanto riguarda la dinamicità
- Il B^+-tree è la soluzione comunemente adottata nei DBMS commerciali per realizzare indici paginati, dinamici e multi-livello