

# Indexes & B+trees

## Indici

L'indicizzazione è una tecnica di ottimizzazione utilizzata per velocizzare le interrogazioni, gli indici sono una struttura dati che contiene informazioni complementari che supportano un accesso efficiente ai dati. La chiave di ricerca è definita utilizzando alcuni attributi; le chiavi di ricerca non sono chiavi primarie, infatti la stessa chiave può contenere più valori.

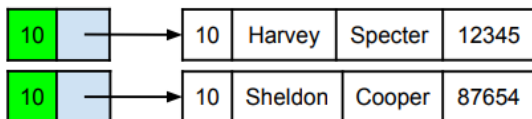
Un indice è una coppia  $\langle \text{key}, \text{label} \rangle$  e supporta il recupero di tutte le etichette con un dato valore K in modo efficiente.

Le *etichette* (labels) possono essere:

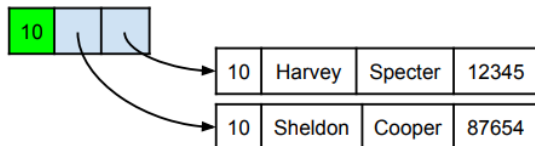
1. il dato stesso;

10	Harvey	Specter	12345
10	Sheldon	Cooper	87654

2. l'identificatore del dato (RID) con il valore K della chiave;



3. una lista di identificatori dello stesso valore con chiave K;



La rappresentazione delle etichette è indipendente dal metodo di ricerca.

Osservazioni:

- in una base di dati, è possibile avere al massimo un solo indice sui dati utilizzando la prima rappresentazione;
- utilizzando la prima rappresentazione, la dimensione dell'indice è la stessa dei dati;
- la stessa chiave di ricerca può contenere più valori;
- la terza rappresentazione è la soluzione più compatta, ma le etichette hanno dimensioni variabili.

In SQL:

```
//per creare indici
CREATE [UNIQUE] INDEX IndexName ON Table(AttributeList)

//per eliminare un indice
DROP INDEX IndexName
```

## Classificazione

- **indice primario:**

- è un indice basato su un insieme di attributi che include la chiave primaria;
- i dati sono ordinati in base a questi attributi;
- se l'indice non è basato sulla chiave primaria, allora è un **indice secondario**.

- **indice denso:**

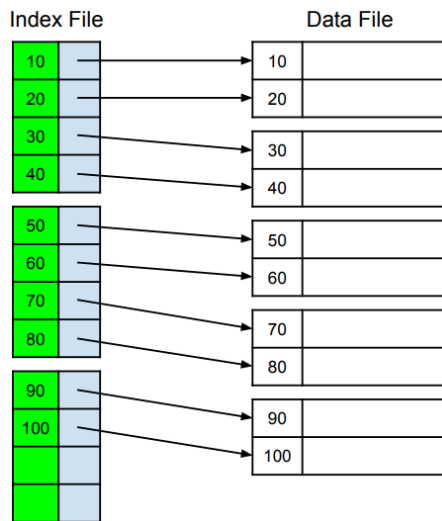
- ogni valore chiave di ricerca nel file dei dati ha almeno una voce corrispondente nell'indice;
- se non tutti i valori chiave di ricerca sono rappresentati, allora l'indice è **sparso**.

- **indice clusterizzato:**

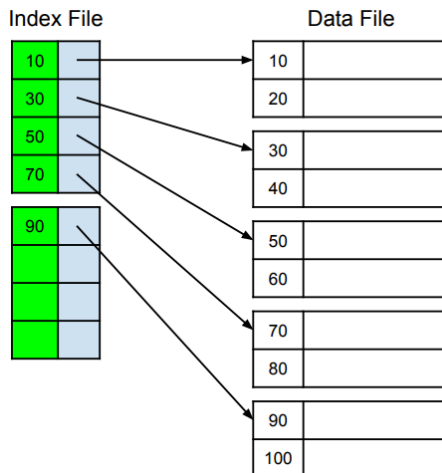
- l'ordine dei record nel file dati corrisponde (o è simile) all'ordine delle etichette (chiavi) nell'indice;
- se l'ordine dei record non corrisponde, allora l'indice è **non clusterizzato**.

Esempi:

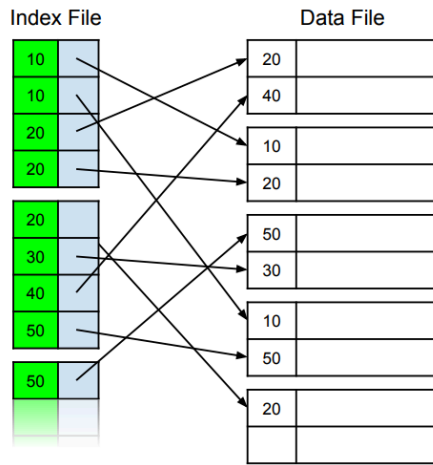
- **indice clusterizzato e denso:**



- **indice sparso e clusterizzato:**



- indice secondario, denso e non clusterizzato



gli indici non clusterizzati danno meno efficienza nell'accesso ai dati (ad esempio tre record con lo stesso valore sono memorizzati in tre blocchi diversi).

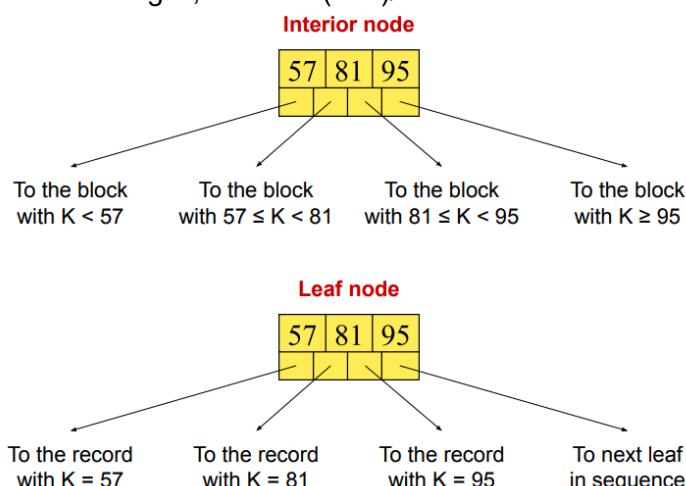
## B+trees

Le strutture ad albero dinamiche di tipo B+trees (un tipo speciale di B-alberi), sono le più frequentemente usate nei DBMS relazionali per la realizzazione degli indici.

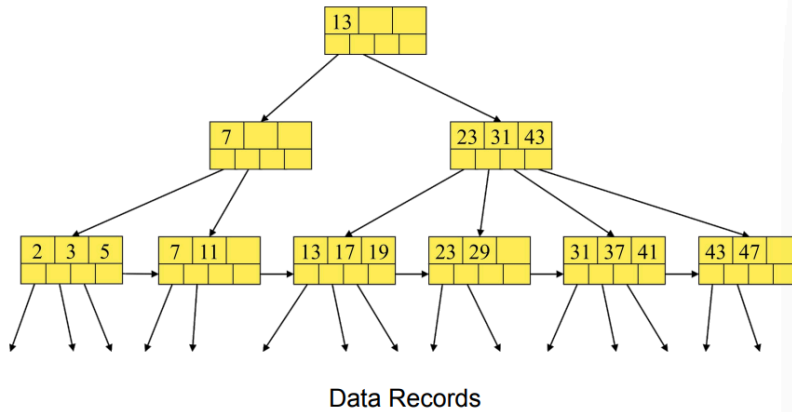
- Ogni albero è caratterizzato da un nodo radice, vari nodi intermedi e vari nodi foglia;
- ogni nodo ha un numero di discendenti che dipende dall'ampiezza della pagina;
- gli alberi sono **bilanciati**, ovvero la lunghezza di un cammino che collega il nodo radice a un qualunque nodo foglia è costante; in questo modo il tempo di accesso alle informazioni contenute nell'albero è lo stesso per tutte le foglie ed è pari alla profondità dell'albero.

## Regole:

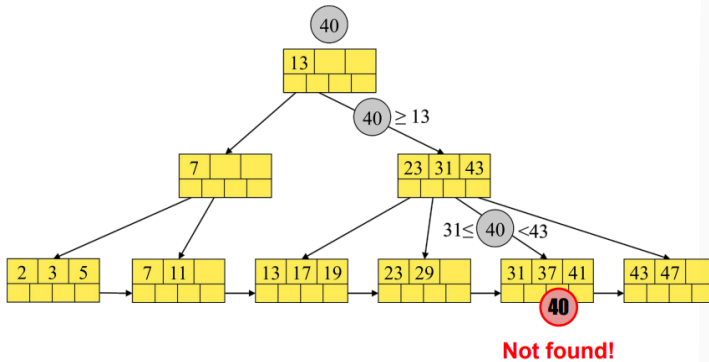
- le chiavi nei nodi foglia sono copie delle chiavi del data file. Queste chiavi sono distribuite tra le foglie in modo ordinato, da sinistra a destra.
- alla radice, ci sono almeno due puntatori utilizzati (con almeno due record di dati nel file). Tutti i puntatori puntano ai blocchi del livello sottostante;
- in presenza di  $n$  chiavi, bisogna avere  $n+1$  puntatori;
- In un nodo interno, tutti i puntatori utilizzati puntano a blocchi al livello immediatamente inferiore e almeno  $\lfloor (n+1)/2 \rfloor$  devono essere utilizzati;
- in una foglia, l'ultimo puntatore punta al blocco foglia successivo a destra. Tra gli altri puntatori in un blocco foglia, almeno  $\lfloor (n+1)/2 \rfloor$  di essi sono utilizzati e puntano a un record di dati.



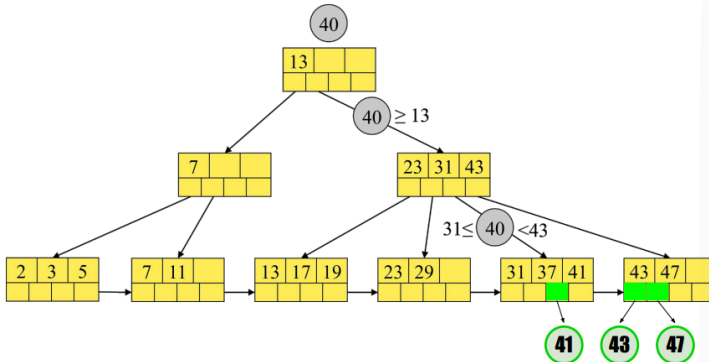
Esempio di B+tree:



Esempio di **equality search**:



Esempio di **range search**:



## Inserimento:

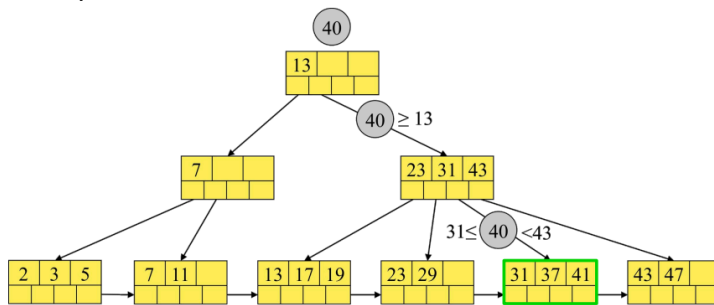
L'inserimento è, in linea di principio, ricorsivo:

1. trovare il nodo corretto:
  - per inserire un valore, si parte dalla radice e si scende lungo l'albero seguendo le chiavi fino a trovare il nodo foglia che dovrebbe contenerlo.
2. inserire il valore nel nodo foglia:
  - se il nodo foglia ha spazio disponibile (cioè non ha già n-1 chiavi), semplicemente aggiungi la nuova chiave in ordine crescente.
3. gestione dell'overflow se il nodo è pieno:
  - se il nodo foglia è pieno, viene diviso in due nodi (split):
    - le chiavi vengono divise in due gruppi di dimensioni approssimativamente uguali;
    - la chiave centrale viene promossa al nodo genitore (il nodo superiore)
  - se il genitore non ha spazio per la chiave promossa, si ripete il processo di divisione verso l'alto (propagazione dello split).

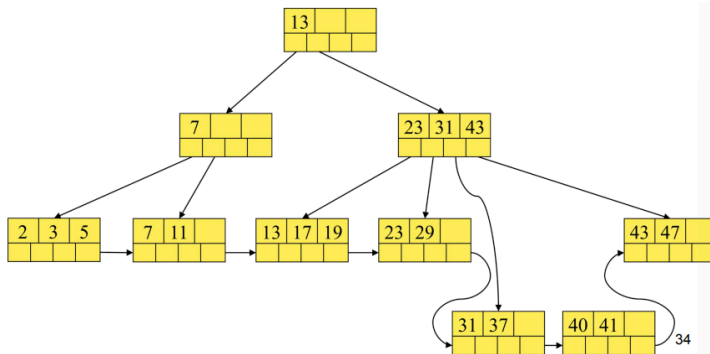
4. aggiornare la struttura dell'albero:

- se anche la radice deve essere divisa (overflow nella radice), si crea una nuova radice con due figli. L'altezza dell'albero aumenta di 1.

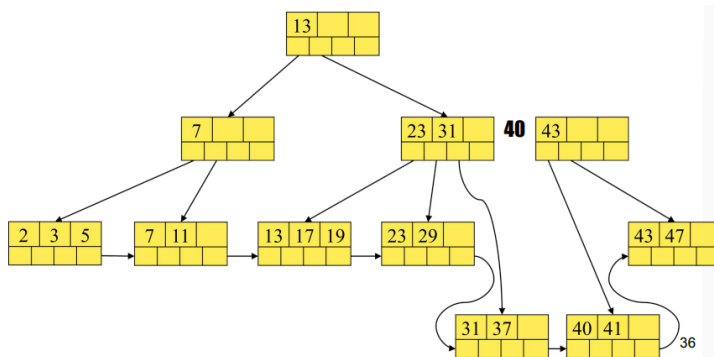
Esempio, inseriamo 40:

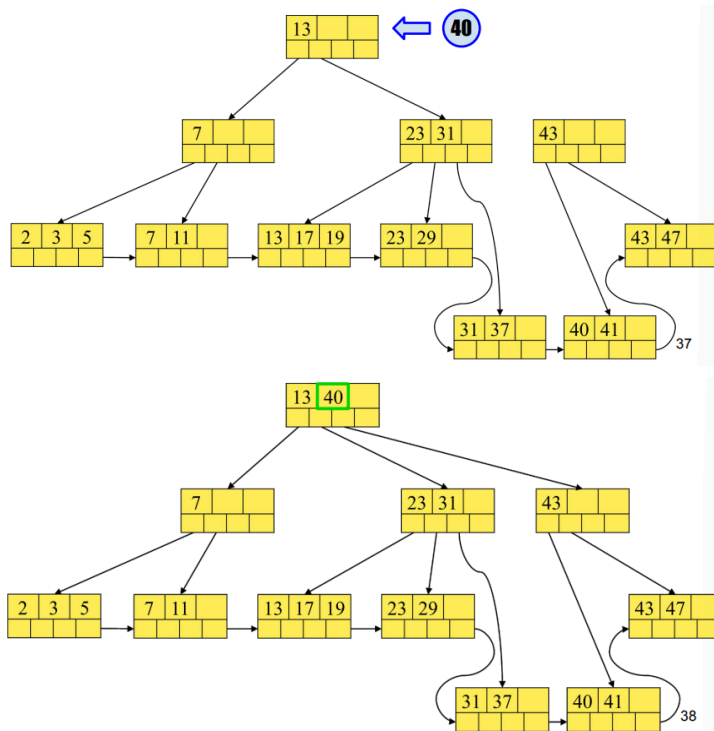


non c'è spazio nella foglia quindi, dobbiamo splittare:



lo split di una foglia al livello inferiore, corrisponde all'inserimento di una nuova coppia chiave-puntatore al livello superiore:





## Eliminazione:

Dobbiamo garantire che l'albero rimanga bilanciato e rispetti tutte le sue proprietà.

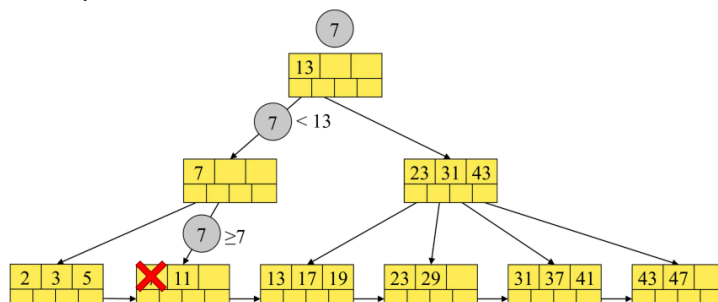
1. trovare il nodo contenente la chiave da eliminare:
  - partendo dalla radice, scendere lungo l'albero seguendo le chiavi per trovare il nodo foglia, non è necessario toccare le chiavi nei nodi intermedi immediatamente, in quanto i nodi intermedi contengono solo "puntatori" (chiavi-guida).
2. eliminare la chiave:
  - se la foglia, dopo l'eliminazione della chiave, contiene ancora il numero minimo di chiavi, non dobbiamo fare più nulla;
3. problema di sotto-riempimento:

ci sono due modi per risolvere questo problema:

  1. *prestito da un fratello*:
    - se un nodo adiacente (fratello) ha più del numero minimo di chiavi, gli "prendiamo in prestito" una chiave.
    - aggiorniamo anche la chiave-guida nel nodo genitore per riflettere i cambiamenti.
  2. *fusione con un fratello*:
    - se nessun fratello ha chiavi in eccesso, fondiamo il nodo sotto-riempito con un fratello adiacente.
    - la chiave-guida nel genitore viene abbassata e inclusa nel nodo fuso.
    - se il genitore rimane sotto-riempito, si applica ricorsivamente lo stesso processo al genitore.

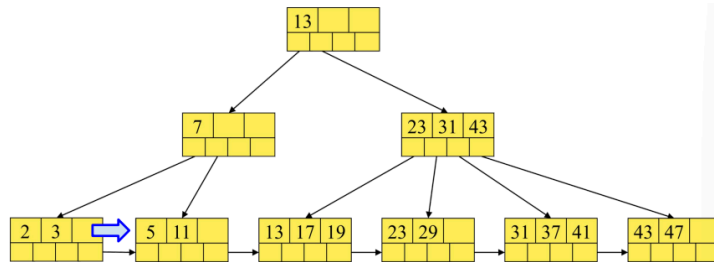
Esempio: cerchiamo la chiave con valore 7 e la eliminiamo:

Esempio: cerchiamo la chiave con valore 7 e la eliminiamo:

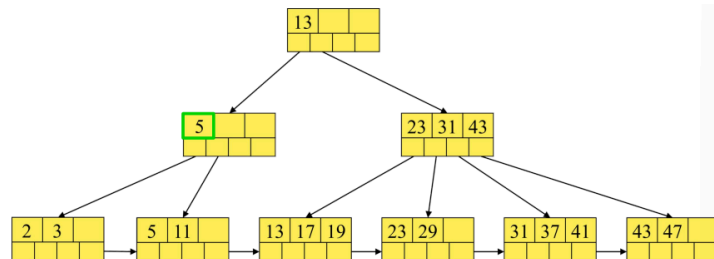


in questo modo la seconda foglia ha solo una chiave, mentre abbiamo bisogno di almeno due

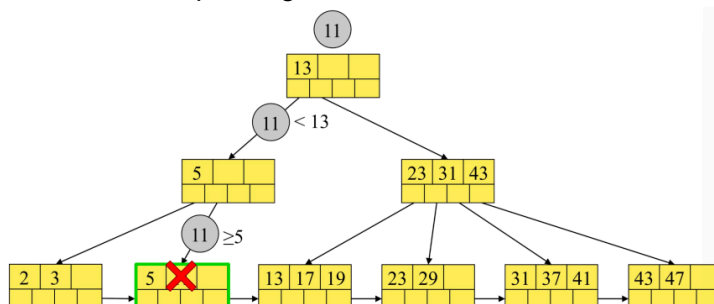
chiavi in ogni foglia; allora il nodo di sinistra "presta" una chiave al nodo di destra:



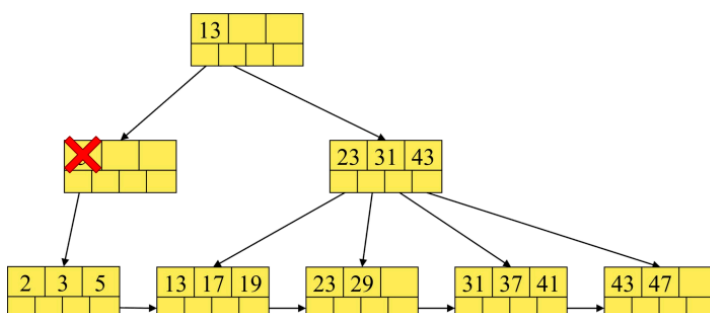
e aggiorniamo la "chiave-guida":



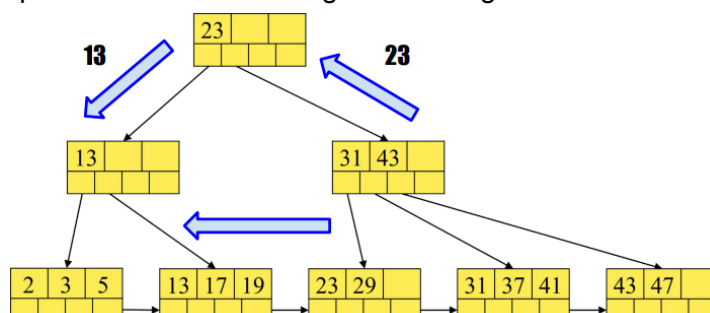
Secondo esempio: vogliamo eliminare la chiave con valore 11



Non possiamo prendere in prestito dalla prima foglia e non c'è nessun fratello a destra da cui prendere in prestito, perciò abbiamo bisogno di fondere la seconda foglia con un fratello (la prima):



I puntatori e le chiavi nel genitore vengono sistemate in modo da riflettere la situazione nei figli:




**Search complexity:**

$$\log_{(n/2)} N \leq L \leq \log_{(n/2)} N + 1$$

Let's assume that:

- Block size: 4096 B.
- Key size: 4 B.
- Pointer size: 8 B.
- There is no header information kept on the blocks.

The value of n is:

-  A. 340
- B. 4096
- C. 48
- D. 8

Hint: we want to find the largest integer value of n such that

$$4n + 8(n + 1) < 4096$$