Indici e progettazione fisica

corso di basi di dati e laboratorio

Prof. Alfio Ferrara

Anno Accademico 2020/2021

Indice

1 Progettazione fisica

1.1 Introduzione

Il concetto di progettazione fisica

- Le basi di dati sono memorizzate fisicamente su file di record.
- I DBMS consentono diverse opzioni di tuning e regolazione dei criteri di memorizzazione fisica.

- Al livello più semplice, la memorizzazione fisica può essere controllata defininendo specifici indici sulla base del carico di operazioni previste sui dati.

Collocazione su disco dei record di un file

- I dati sono memorizzati sotto forma di **record**. Ogni record è una collezioni di valori di dati collegati in cui ogni valore è un insieme di uno o più byte e prende il nome di **campo** del record.
- I dati di tipo `BLOB` e `CLOB` sono invece memorizzati a parte e il record contiene solo un puntatore al dato.

File di record

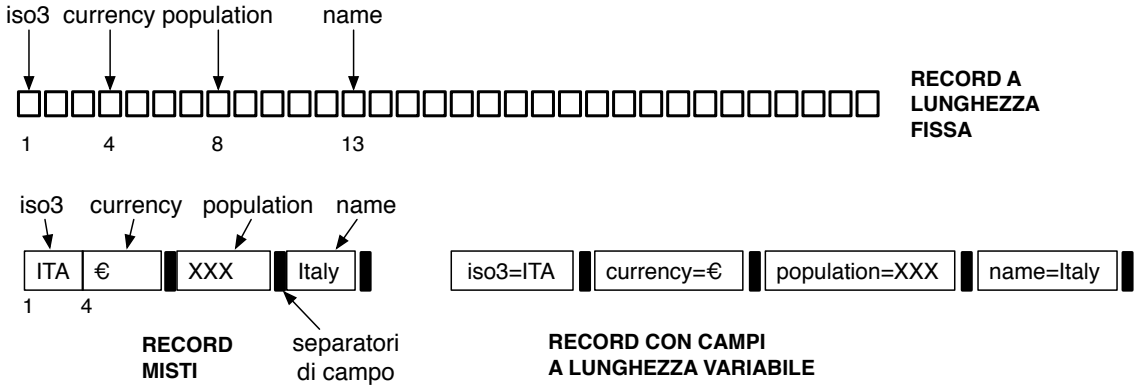
- Un file è semplicemente una sequenza di record. Se i record hanno tutti la stessa dimensione in byte, allora di parla di **record a lunghezza fissa**, altrimenti di **record a lunghezza variabile**.

Esempio. Si consideri la seguente tabella:

```
CREATE TABLE country (
  iso3 CHAR(3),
  currency CHAR(4),
  population INTEGER(9),
  name VARCHAR(20),
)
```

Esempio

Possibili strategie di memorizzazione:



1.2 Gestione dei blocchi

Ripartizione in blocchi

- I record vanno ripartiti in blocchi. Se un blocco è di dimensione B byte, un record composto da $R \leq B$ byte, si possono inserire

$$bfr = \left\lfloor \frac{B}{R} \right\rfloor$$

  dove bfr è detto **blocking factor** (fattore di blocco).

- Ciò comporta l'esistenza di $B - (bfr * R)$ byte.

- I questi casi si può usare un'organizzazione **spanned** in cui si inserisce una parte del record nello spazio inutilizzato con un puntatore alla fine verso la parte restante del record in un altro blocco.

- In tal caso il bfr è il numero medio di record per blocco. Perciò il numero di blocchi necessari per un file è $\lceil (R/bfr) \rceil$.

File di record non ordinati

- In questa organizzazione detta **file heap** i record sono collocati nei file nell'ordine in cui sono inseriti.

- L'inserimento di un record è pertanto molto efficiente.

- La ricerca richiede però una **ricerca lineare** nel file.

- La cancellazione comporta il ritrovamento del record e successivamente copiare il blocco corrispondente in un buffer, cancellare il record e poi riscrivere il blocco (sprecando spazio sul file).

- Un altro metodo consiste nell'associare a ogni record un bit detto **indicatore di cancellazione**. In questo caso i record cancellati sono semplicemente marcati come cancellati e vengono fisicamente eliminati solo in fase di riorganizzazione dei file.

File di record ordinati

- Un metodo alternativo (**file sorted**) di organizzazione dei file consiste nell'ordinare i record basandosi su uno dei loro campi, detto **campo di ordinamento**.

- In questo modo, record consecutivi nell'ordinamento sono frequentemente all'interno dello stesso blocco.
- La ricerca per campo di ordinamento è estremamente efficiente.

Ricerca binaria

In un file ordinato di b blocchi in cui si cerchi il valore K di una chiave di ordinamento e che contenga nell'header del file l'indirizzo su disco dei blocchi del file, si può utilizzare una ricerca binaria:

sia $l = 1; u = b$
finché $(u \geq l)$ esegui:
$i = (l + u) div 2$
trasferisci il blocco i del file nel buffer
se $K <$ valore del campo chiave del primo record di i
allora $u = i - 1$
se invece $K >$ valore del campo chiave dell'ultimo record di i
allora $l = i + 1$
se invece K è nel blocco i allora **termina**

Inserimento e cancellazione

- In un file ordinato, inserimento e cancellazione sono operazioni dispendiose.
- Un'opzione consiste nel tenere in ogni blocco un certo spazio per l'inserimento di nuovi record.
- Un altro metodo è mantenere un file di appoggio non ordinato (detto **file di overflow**). Gli inserimenti avvengono solo nel file di overflow e periodicamente si fondono il file di overflow e quello ordinato.

File hash

L'idea dei file hash è di applicare a un campo detto **campo hash** dei record una funzione di hash h che, applicata al valore del campo hash fornisca l'indirizzo del blocco in cui è memorizzato il record corrispondente.

L'hash è implementato in una tabella di hash realizzata per mezzo di vettori di indice variabile fra 0 e $N-1$. A questo punto useremo una funzione di hash che trasforma il valore del campo hash in un intero compreso fra 0 e $N-1$, ad esempio $h(K) = K \ mod \ N$.

Lo scopo principale di una funzione di hash è distribuire i record nel modo più uniforme possibile, minimizzando le collisioni fra hash. Le collisioni si possono gestire tramite: i) indirizzamento aperto, scorrendo il vettore a partire dalla posizione già occupata fino a trovarne una libera, ii) concatenamento con posizioni di overflow (estendendo il vettore); iii) hash multipli utilizzati a cascata nel caso una posizione sia già occupata.

2 Indici

2.1 Caratteristiche generali

Indice
Nei DBMS un indice è una **struttura di accesso secondaria** utilizzata per rendere più veloce l'accesso ai dati memorizzati in una delle strutture fisiche descritte in precedenza.

Gli indici possono essere costruiti su qualsiasi combinazione di campi dei record e funzionano con lo stesso principio di un indice analitico in un libro: mettono a disposizione una struttura ordinata a accesso veloce per la chiave di indicizzazione e poi utilizzano un puntatore al blocco del record corrispondente per reperire i dati.

Un indice velocizza l'accesso, ma ha un costo di mantenimento in inserimento, cancellazione e modifica. Occorre perciò bilanciare e scegliere con attenzione su quali campi costruire un indice.

Creazione in SQL
La sintassi di creazione di un indice è:

```
CREATE INDEX name ON table [ USING method ] ( column )
```

Tipologie di indici

- **Indice primario**: specificato sul campo chiave di ordinamento di un **file ordinato** di record. Il campo di ordinamento è tale che tutti i record hanno **valori univoci** per quel campo.

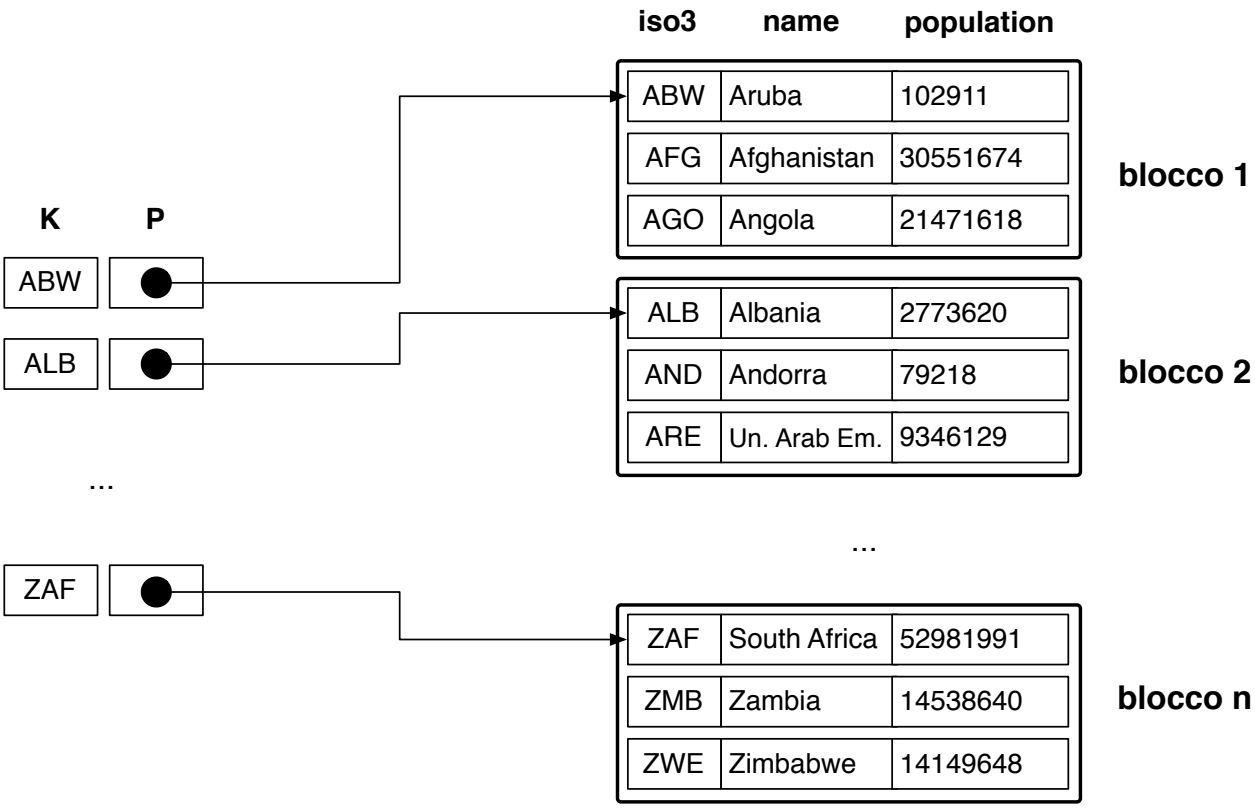
- **Indice di cluster**: se si vuole indicizzare un campo con valori non univoci (i.e., non chiave) si usa un indice di cluster su file chiamati **file clustered**.
- Si noti che un file può avere al massimo un campo di ordinamento fisico, perciò può avere al massimo un indice primario o un indice di cluster, ma non entrambi. Su qualsiasi campo non di ordinamento di un file si può invece definire un **indice secondario**.

2.2 File ordinati

Indici primari

- Un **indice primario** è un file ordinato i cui record hanno lunghezza fissa e solo due campi: i) un campo chiave, detto anche chiave primaria dell'indice, e ii) un puntatore a un blocco del disco.
- Una voce i dell'indice è perciò della forma $\langle K(i), P(i) \rangle$.

Esempio



Struttura dell'indice primario

- Il numero delle voci dell'indice primario è uguale al numero di blocchi memorizzati nel file. Il primo record di ciascun blocco è detto record àncora.
- In genere, gli indici possono essere **densi**, ovvero contenere una voce per ogni valore della chiave di ricerca, oppure **sparsi**, ovvero contenere voci solo per alcuni dei valori della chiave di ricerca.
- Pertanto, un indice primario è sempre un indice sparso, poiché un blocco contiene più record.
- E' facile vedere che il file di indice contiene molti meno blocchi del file dei record perché c'è una voce per ogni blocco del file dei record e i record dell'indice hanno solo due campi.

Esempio con ricerca binaria sull'indice (I)

Supponiamo di avere un file ordinato:

- Record a dimensione fissa $R = 100$ byte e indivisibili.
- File con $r = 30.000$ record e blocchi $B = 1024$ byte.
- Fattore di blocco: $bfr = \lfloor B/R \rfloor = \lfloor 1024/100 \rfloor = 10$ record per blocco.
- Il numero di blocchi necessari per il file è $b = \lceil r/bfr \rceil = \lceil 30.000/10 \rceil = 3000$ blocchi.
- Una ricerca binaria sul file di dati richiederebbe approssimativamente $\lceil \log_2 b \rceil = \lceil \log_2 3000 \rceil = 12$ accessi al file.

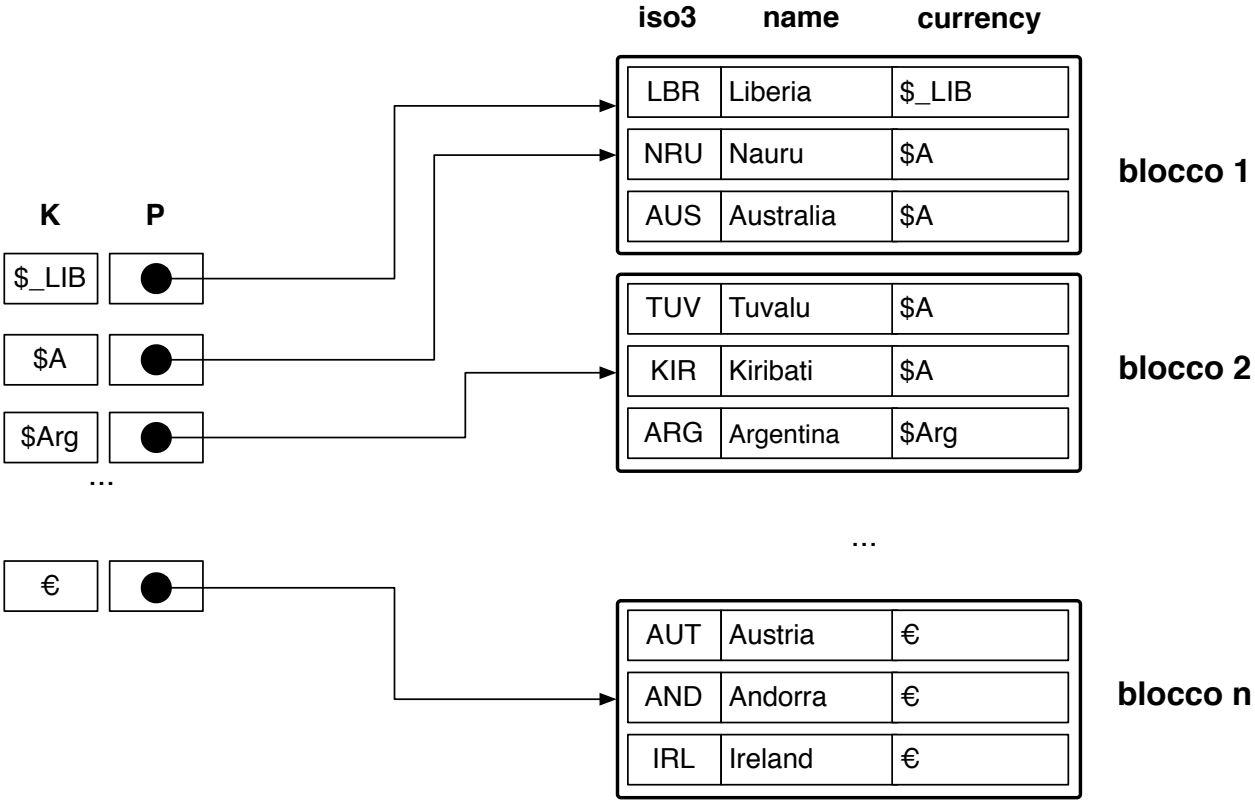
Esempio con ricerca binaria sull'indice (II)

- Ipotizziamo di avere 3 byte per il campo chiave di ordinamento e un puntatore a blocco di 6 byte, per un totale di 9 byte per record dell'indice.
- Il fattore di blocco per l'indice è: $bfr_i = \lfloor B/R_i \rfloor = \lfloor 1024/9 \rfloor = 113$ voci dell'indice per blocco.
- L'indice ha tante voci quanti sono i blocchi del file ordinato, ovvero 3000.
- Perciò, il file di indice ha $\lceil r_i/bfr_i \rceil = \lceil 3000/113 \rceil = 27$ blocchi.
- La ricerca binaria richiede quindi $\lceil \log_2 b_i \rceil = \lceil \log_2 27 \rceil = 5$ accessi ai blocchi mediamente.

Indice di cluster

- Un indice di cluster è usato per file ordinati secondo un campo di ordinamento non chiave (ovvero che può contenere valori duplicati).
- La chiave dell'indice contiene una voce per ogni valore distinto del campo di ordinamento. Il puntatore si riferisce al primo blocco che contiene almeno un record corrispondente alla chiave dell'indice.
- Pertanto, anche l'indice di cluster è un indice sparso.

Esempio



2.3 Indici secondari

Indice secondario

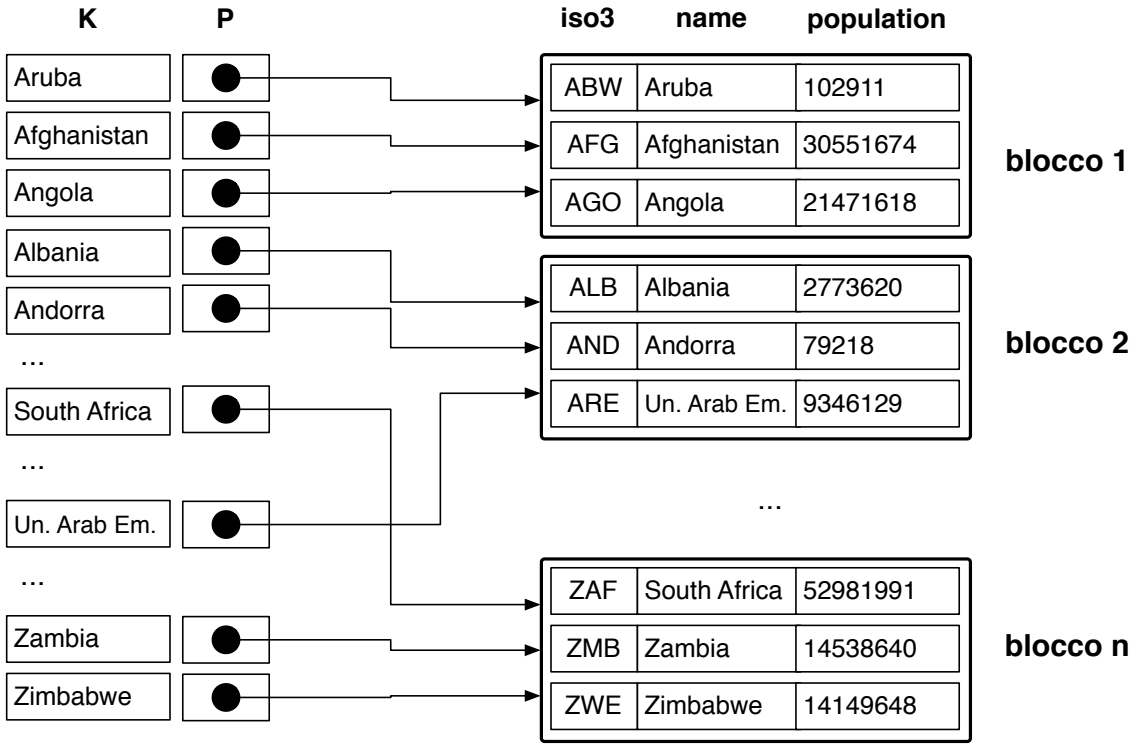
- Un indice secondario fornisce un'ulteriore struttura di accesso a un file per il quale ci sia già un indice primario.

- Il file dei record può essere ordinato (non rispetto al campo di indicizzazione secondaria), non ordinato o hash.
- Il campo indicizzato può essere una chiave (ovvero avere valori univoci) o no.
- In termini relazionali, la differenza è data dal fatto che il campo indicizzato presenti o meno un vincolo `UNIQUE`.

Indici secondari su campi chiave

- Se il campo di indicizzazione è chiave, l'indice contiene tutti i valori del campo come chiave dell'indice e puntatori al blocco in cui è memorizzato il record o al record stesso. In ogni caso, l'indice risultante è denso.
- L'indice contiene molte voci (una per ogni valore del campo di indicizzazione) ma il risparmio in termini di tempo di ricerca è comunque notevole perché il campo di indicizzazione non è campo di ordinamento e perciò una ricerca senza indice non consentirebbe la ricerca binaria.
- La chiave dell'indice è invece mantenuta ordinata e perciò permette una ricerca binaria.

Esempio (I)



Esempio (II)

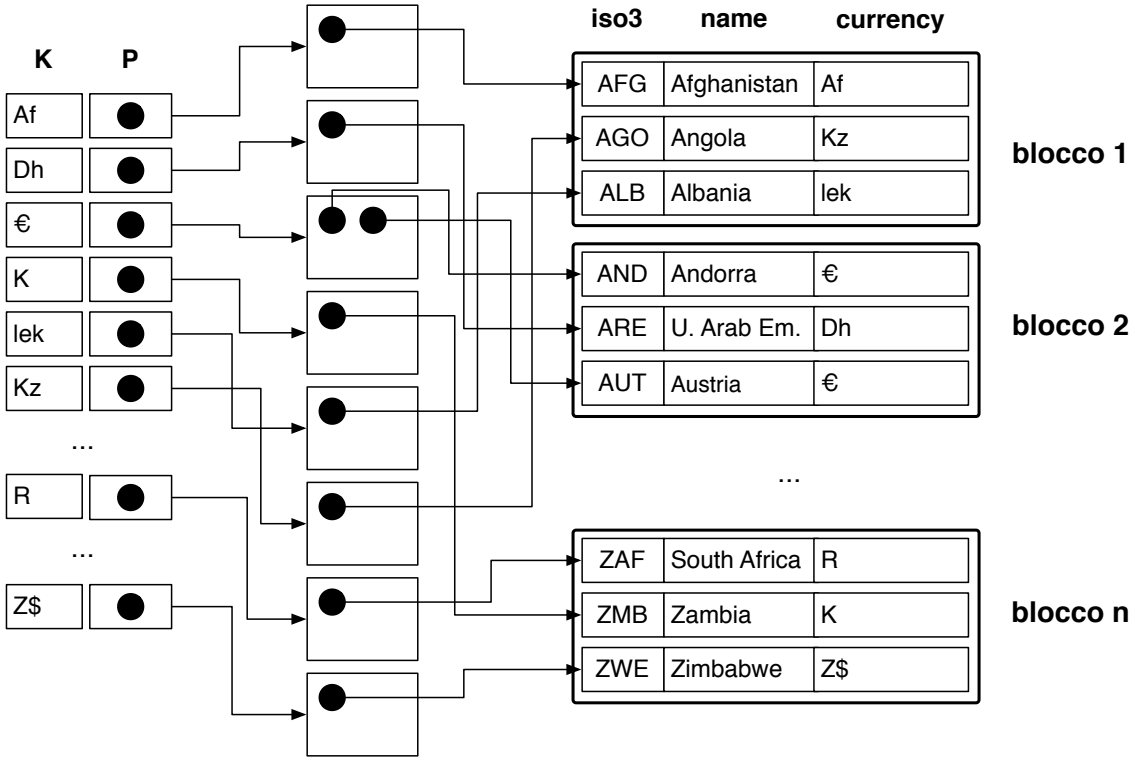
- Con un file che abbia $r = 30.000$ record di dimensione $R = 100$ byte e $B = 1024$ byte per blocco, avremmo $b = 3000$ blocchi.
- La ricerca lineare su un campo non ordinato richiederebbe mediamente $b/2 = 3000/2 = 1500$ accessi.
- Usando invece un indice secondario il cui record abbia per esempio dimensione $R_i = 15$ byte avremmo 30.000 voci nell'indice (indice denso) con un'occupazione di 442 blocchi.
- La ricerca binaria utilizzando l'indice perciò consentirebbe una media di $\lceil \log_2 442 \rceil = 9$ accessi, più un'ulteriore accesso al blocco, per un totale di 10 accessi contro i 1500 della ricerca lineare.

Indici secondari su campi non chiave

Se il campo di indicizzazione può avere valori duplicati, vi sono tre opzioni di memorizzazione dell'indice:

- Inserire più voci dell'indice con medesimo valore di K.
- Usare un record a lunghezza variabile per l'indice in modo da inserire più puntatori per ogni voce dell'indice (che sarà in questo caso sparso).
- Mantenere voci a lunghezza fissa, ma inserendo un ulteriore livello per i puntatori.

Esempio



Classificazione per campo di indicizzazione

	Campo indice ordinato	**Campo indice non ordinato**
Chiave	primario	secondario (chiave)
Non chiave	cluster	secondario (non chiave)

Classificazione per proprietà

Indice	**Num. Voci**	**Denso**	**Oggetto del puntatore**
primario	numero blocchi	No	blocco
cluster	numero valori distinti	No	blocco
secondario chiave	num. record	Sì	record / blocco
secondario non chiave	num. record	Sì/No	record

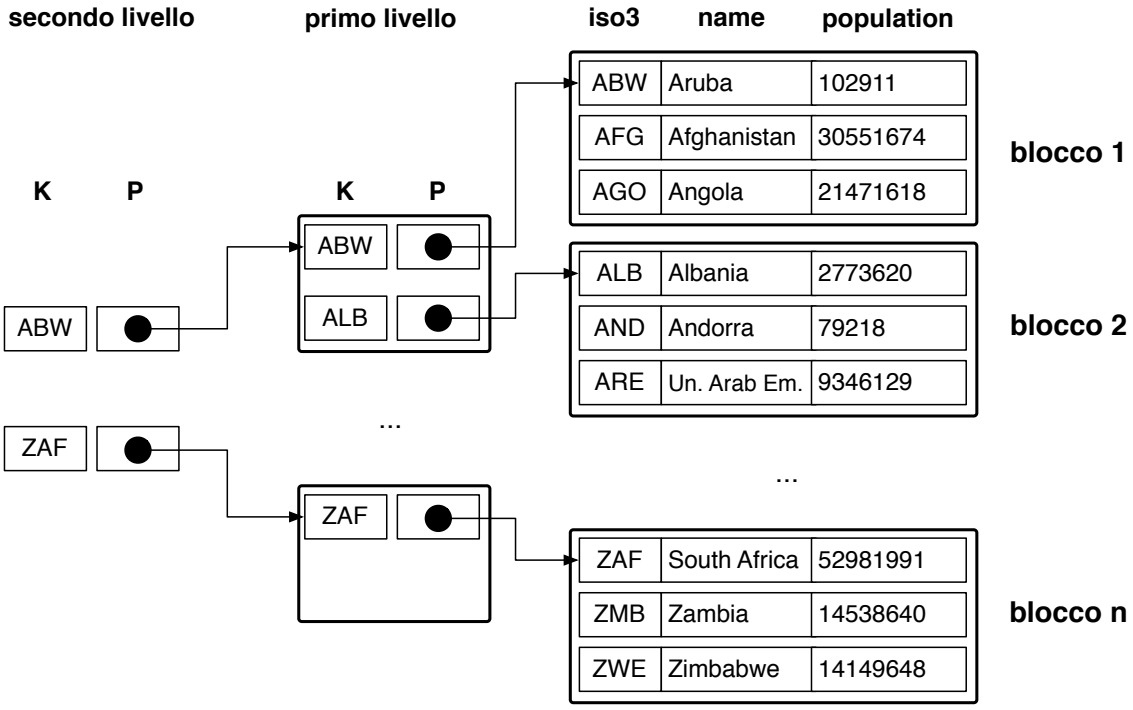
3 Indici multilivello

3.1 Introduzione

Indice multilivello

- L'idea alla base della ricerca binaria è di dividere per 2 ad ogni passo lo spazio di ricerca del valore desiderato. Da ciò deriva la stima di costo $\log_2 b_i$.
- Con un indice multilivello, creiamo un indice primario del file di indice. In tal modo, definendo come **fan-out (fo)** dell'indice di primo livello il suo blocking factor bfr_i, otteniamo un indice di secondo livello che ha un costo di accesso $\log_{fo} b_i$.
- Il motivo è che a ogni passo della ricerca, usando l'indice di secondo livello, siamo in grado di dividere lo spazio di ricerca di un fattore pari al numero di voci del blocco di file di indice.
- E' così possibile usare più livelli di indicizzazione a partire da indici di qualsiasi tipo purché il campo chiave K dell'indice di primo livello contenga valori univoci.

Esempio (I)

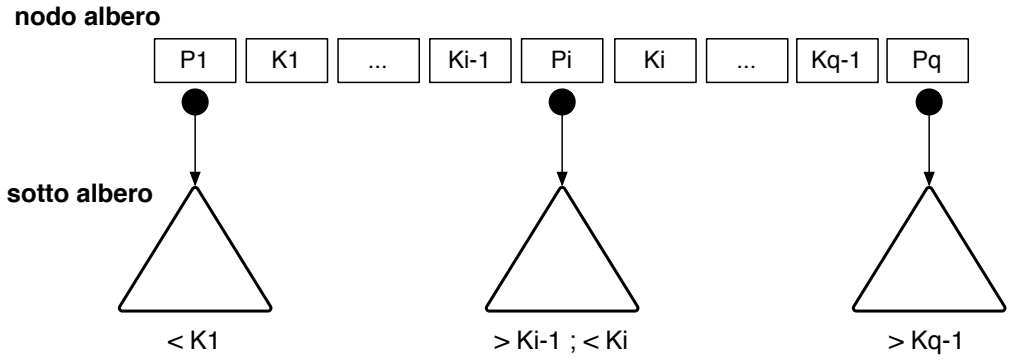


Esempio (II)

- Riprendiamo l'esempio dell'indice secondario con $bfr\_i = 68$ e 442 blocchi.
- Il numero di blocchi al secondo livello sarà $b_2 = \lceil b_1/bfr_i \rceil = \lceil 442/68 \rceil = 7$ e (come è facile calcolare) il numero di blocchi al terzo livello sarà $t_3 = 1$.
- Ora, per accedere a un record sarà necessario accedere a un blocco per ogni livello più il blocco del file di dati, per un totale di $3 + 1 = 4$ accessi.
- Lo schema di indicizzazione usando un file ordinato e un indice primario multilivello è la più usata e prende spesso il nome di **file sequenziale indicizzato**.

Alberi di ricerca

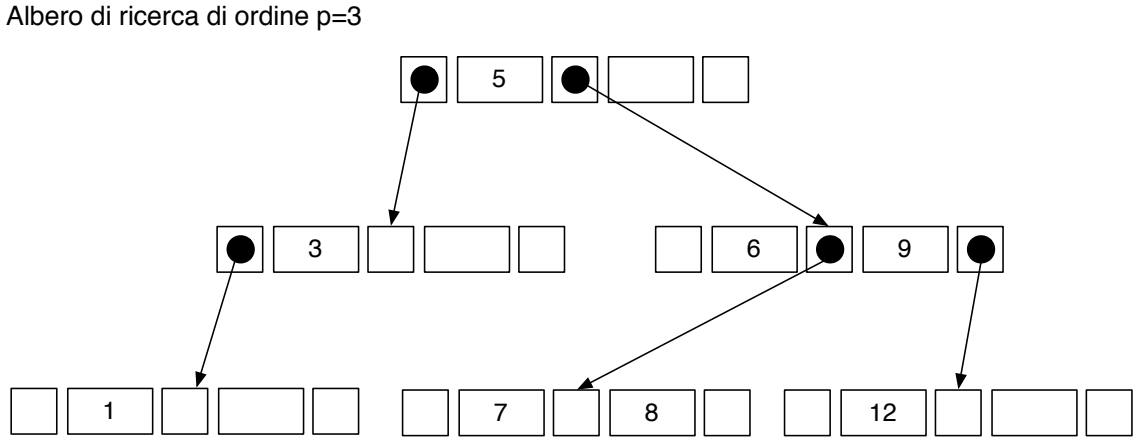
Un **albero di ricerca** di ordine p è un albero tale per cui ogni nodo contiene **al massimo** $p - 1$ valori di ricerca e i p puntatori sono nell'ordine: $\langle p_1, K_1, p_2, K_2, \ldots, P_{q-1}, K_{q-1}, P_p \rangle$.



I vincoli che devono essere rispettati sono:

- in ciascun nodo: $K_1 < K_2 < \cdots < K_{q-1}$
- per tutti i valori X del sottoalbero a cui si riferisce P_i si ha $K_{i-1} < X < K_i$

Esempio



Bilanciamento degli alberi di ricerca

La caratteristica più importante nella gestione di un albero di ricerca è mantenerne il bilanciamento in modo che:

- i nodi siano distribuiti uniformemente e la profondità dell'albero sia minimizzata;
- rendere uniforme la velocità di ricerca in modo che il tempo medio per trovare una qualsiasi chiave sia lo stesso.

3.2 Alberi B-tree

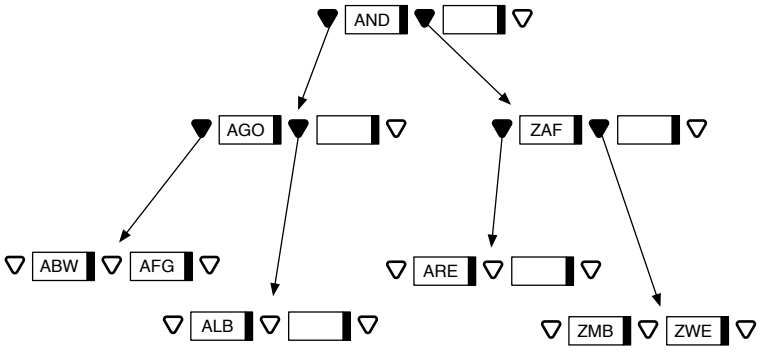
B-tree

Un albero B (B-tree, balanced tree) è un albero di ricerca con ulteriori vincoli che ne garantiscono il bilanciamento. In particolare, un B-tree di ordine p usato come struttura di accesso su un campo chiave di un file di dati ha struttura:

- ogni nodo interno ha forma: $\langle P_1, \langle K_1, Pr_1\rangle, P_2, \langle K_2, Pr_2\rangle, \ldots, \langle K_{q-1}, Pr_{q-1}\rangle, P_q\rangle$, in cui Pr_i è il puntatore al record o blocco corrispondente a K_i, mentre p_i è un puntatore al sotto-albero.
- all'interno di ogni nodo $K_1 < K_2 < \cdots < K_{q-1}$
- per tutti i valori X del campo chiave nel sottoalbero puntato da P_i si ha: $K_{i-1} < X < K_i$ per $1 < i < q$; $X < K_i$ per $i = 1$; K_{i-1} per $i = q$
- ogni nodo contiene al massimo p puntatori dell'albero
- ogni nodo tranne radice e foglie ha almeno $\lceil p/2 \rceil$ puntatori dell'albero; il nodo radice ha almeno due puntatori all'albero salvo che sia l'unico nodo dell'albero
- un nodo con q puntatori a albero (con $q \leq p$) contiene $q - 1$ valori del campo chiave e, quindi, $q - 1$ puntatori ai dati

- tutti i nodi foglia sono allo stesso livello e hanno la medesima struttura dei nodi intermedi a parte il fatto che tutti i loro puntatori a albero sono nulli

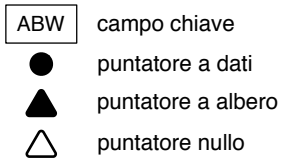
Esempio



FILE DATI

ABW	Aruba	102911
AFG	Afghanistan	30551674
AGO	Angola	21471618
ALB	Albania	2773620
AND	Andorra	79218
ARE	Un. Arab Em.	9346129
ZAF	South Africa	52981991
ZMB	Zambia	14538640
ZWE	Zimbabwe	14149648

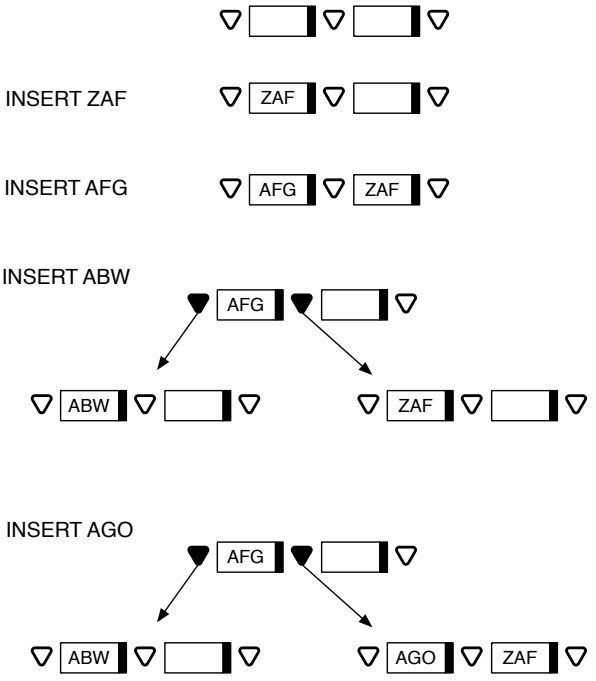
LEGENDA



Inserimento in B-tree

- Trovare il nodo foglia che dovrebbe contenere il nuovo valore di inserimento
- Se il nodo contiene meno di $p - 1$ chiavi, inserire il nuovo elemento e ordinare le chiavi e i rispettivi puntatori
- Se il nodo non ha più spazio, dividerlo equamente in due nodi come segue: trovare il valore intermedio nella lista formata dalle chiavi del nodo più il nuovo valore
- Inserire i valori strettamente inferiori al valore intermedio in un nodo di sinistra e i valore strettamente superiori al valore intermedio in un nodo di destra
- Inserire il valore intermedio nel nodo di ordine superiore, causando se necessario l'aggiornamento ricorsivo dei nodi superiori per mezzo della stessa operazione di divisione. Se il nodo di ordine superiore è la radice dell'albero e non può contenere nuovi valori, allora procedere alla creazione di una nuova radice

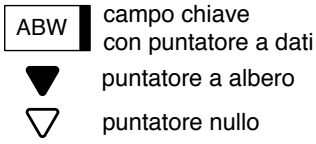
Esempio 1



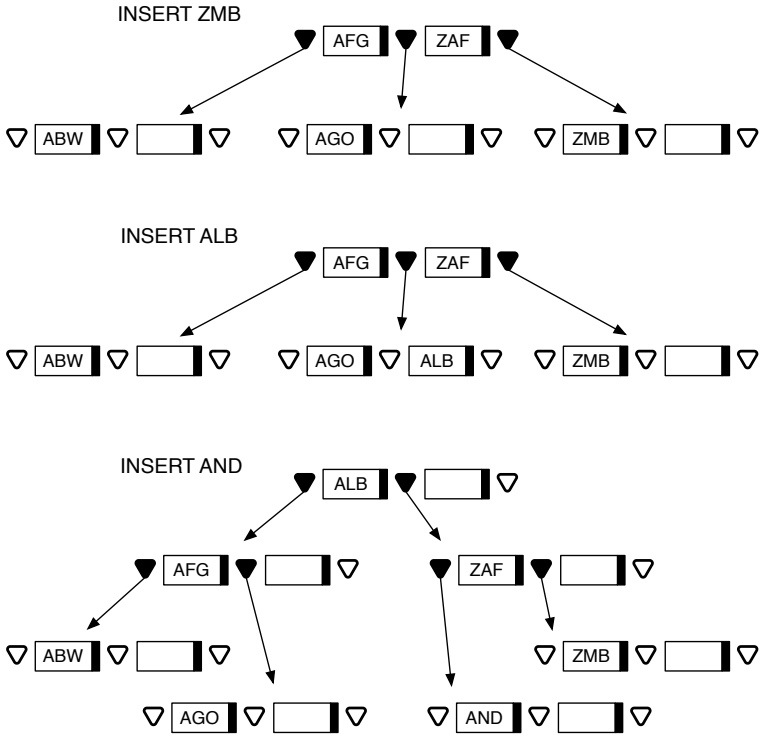
FILE DATI

ABW	Aruba	102911
AFG	Afghanistan	30551674
AGO	Angola	21471618
ALB	Albania	2773620
AND	Andorra	79218
ARE	Un. Arab Em.	9346129
ZAF	South Africa	52981991
ZMB	Zambia	14538640
ZWE	Zimbabwe	14149648

LEGENDA



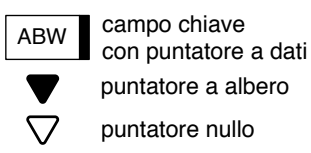
Esempio 2



FILE DATI

ABW	Aruba	102911
AFG	Afghanistan	30551674
AGO	Angola	21471618
ALB	Albania	2773620
AND	Andorra	79218
ARE	Un. Arab Em.	9346129
ZAF	South Africa	52981991
ZMB	Zambia	14538640
ZWE	Zimbabwe	14149648

LEGENDA



Ricerca
La ricerca su B-tree è simile alla ricerca su albero binario: si cerca il valore desiderato nel nodo radice dell'albero. Se il valore non è stato trovato, si ripete ricorsivamente l'operazione sull'albero a cui di riferisce il puntatore a sinistra del primo valore maggiore alla chiave. Se non ci sono valori maggiori della chiave di ricerca si procede con il puntatore a destra dell'ultimo valore del nodo.

3.3 Alberi B^+

Alberi B^+

- Un albero B^+ i puntatori ai dati sono memorizzati solo nei nodi foglia. Perciò i nodi foglia contengono tutti i valori, mentre i nodi intermedi contengono solo alcuni valori K che sono usati per guidare la ricerca come in un B-tree.
- Inoltre i nodi foglia contengono degli ulteriori puntatori che permettono di accedere da una foglia direttamente alla successiva.

Inserimento
L'inserimento avviene come dei B-tree, con una sola differenza: se il nodo richiede di essere diviso, se è un nodo intermedio si procede come nei B-tree, altrimenti se è una foglia occorre dividere il nodo come segue:

- Creare due nodi foglia
- Inserire i primi $\lceil (n-1)/2 \rceil$ valori nel primo nodo
- Inserire i rimanenti nel secondo
- Inserire nel nodo padre un nuovo puntatore per il secondo nodo foglia e sistemare i valori chiave del nodo padre
- Se il nodo padre è pieno, propagare l'operazione di divisione verso l'alto

Ricerca
La ricerca per una chiave K si effettua come segue:

- Cercare nel nodo radice il più piccolo valore di chiave maggiore di K
- Se il valore esiste, seguire il puntatore subito precedente, altrimenti seguire l'ultimo puntatore del nodo
- Se raggiungiamo un nodo foglia, cercare K nel nodo, altrimenti ripetere il passo successivo.

Vantaggi

- Apparentemente, il fatto di non avere puntatori diretti ai dati nei nodi intermedi può apparire svantaggioso rispetto all'uso dei B-tree.
- Poiché nei nodi intermedi occorre memorizzare solo puntatori ai sottoalberi e chiavi e non anche i puntatori ai dati, il numero di puntatori a sottoalberi memorizzabili in un nodo intermedio di un albero B^+ è superiore rispetto a un B-tree; ciò comporta un vantaggio in termini di accessi al disco necessari per attraversare l'albero.
- Il fatto che i nodi foglia siano linkati in modo da formare di fatto un indice ordinato permette di combinare i vantaggi di un indice multilivello con i vantaggi del metodo di accesso sequenziale indicizzato (ISAM).