

INDICE HASH:

Alternativa al B+Tree (Albero)

Questa è una Struttura ad **Accesso Calcolato**

Trasformarli in Numeri

Garantiscono Accesso Associativo (SIMILE AD ARRAY)

Locazione fisica delle TUPLE dipende dal Valore Assunto del Campo CHIAVE

Se ha uno Spazio delle Chiavi nell' Ordine del Milione non è EFFICIENTE Come Cosa.

Allora definisco $h: K \rightarrow B$

dominio CHIAVI codominio BLOCCHI

Soffre però del **Problema delle Collisioni** \Rightarrow da Chiavi diverse OTTENGO Stesso Valore ... un Valore da Minimizzare

Vanno a Sfruttare le Caratteristiche dell' Organizzazione a Blocchi (della mem. SECONDARIA)

$T \rightarrow$ # Tuple Prese

$F \rightarrow$ fattore di Blocco \Rightarrow # Tuple in \leq Blocco al max.

$f \rightarrow$ fattore di Riempimento

Posso Sapere Quanti Blocchi Mi Servono:

$$B = \# \text{Blocchi} = \left\lceil \frac{T}{F \cdot f} \right\rceil$$

Allocherò Quindi B Bucket di Puntatori

funzione di **FOLDING**: $f: K \rightarrow \mathbb{Z}^+$ (da chiavi ad Interi Positivi)

funzione di **Hash**: $h: \mathbb{Z}^+ \rightarrow B$ (da Interi + a Blocco)

Sto però facendo un IPOTESI di DISTRIBUZIONE UNIFORME

Considero che Scegliere una Buona Funzione di HASH è ESSENZIALE, altrimenti devo RICOSTRUIRE da Zero l'INDICE... NON posso in NESSUN modo Aggiornare le fz. di HASH

DATI \Rightarrow file. Seq. Ordinato

$$T = 7$$



$$F = 3$$

$$f = 0.4$$

$B = 6$ Buckets di Puntatori

con al max 3 Puntatori

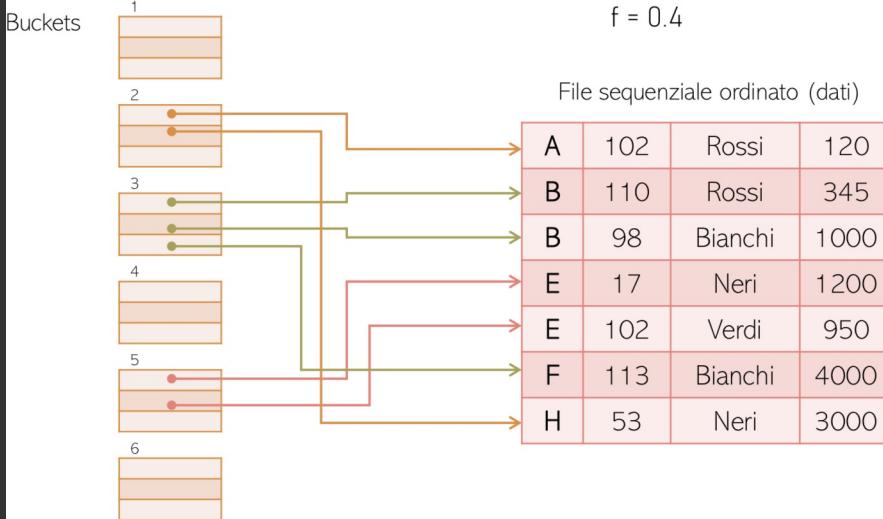
A	102	Simo	120
B	110	da	345
B	98	Omar	1010
E		Badia	3142
E		Rossi	10
F			5
H			0

$$T = 7$$

$$F = 3$$

$$f = 0.4$$

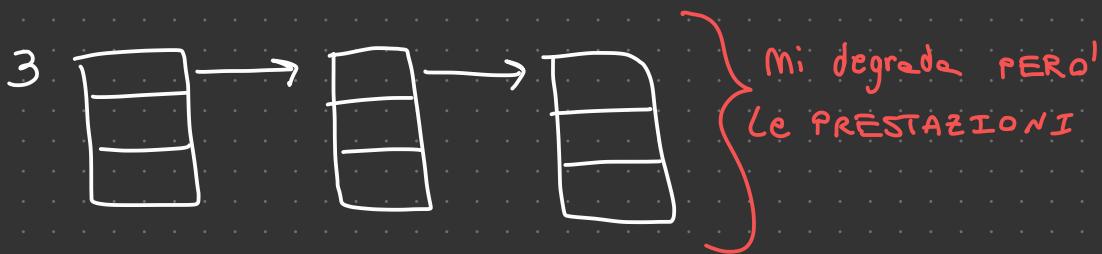
$$\rightarrow B = 6$$



Cosa Succede se devo INSERIRE un Nuovo Puntatore sul BUCKET 3?

Devo usare i BUCKET DI OVERFLOW (Quando ho la Saturaz. di un Blocco)

Per Ogni BUCKET Bisogna Crearne uno parallelo per Ospitare Nuove TUPLE



Ha Quindi un COSTO COSTANTE per la Ricerca visto che devo:

- ① Calcolare $h(K) = B = \text{Blocco}$ [0 Accessi mem Second.]
- ② Accedere al BUCKET B [1 se è BUCKET OVERFLOW]

③ Accedo Alle TUPLE Tramite i PUNTATORI Nel Blocco B

Se ci Sono dei BUCKET DI OVERFLOW Cresce in Modo da Rendere le Prestazioni Non Più Performanti

Le PRESTAZIONI Sono Mentretenute COSTANTI Quando i BUCKET hanno un Basso Coefficiente di Riempimento

Devo GARANTIRE di Ridurre al Minimo le Collisioni, che si Verificano con la Seguente Probabilità:

$$P(t) = \binom{n}{t} \left(\frac{1}{B}\right)^t \left(1 - \frac{1}{B}\right)^{(n-t)}$$

BUCKET T
riceva t
chiavi Su n
Inserimenti

Combinazioni di
n Elementi Presi t
a t

Dipende da B (#Bucket), che Aumentandoli diminuisce la PROBABILITÀ di Collisioni.

È il DBMS che Calcola la fz. di HASH

Dobbiamo Capire Quando è Meglio Indice B+TREE o HASH

In RICERCA: A = const allora - HASH = Costante
- B+TREE = Logaritmico su #Key

 Su INTERVALI
(range Query)

A < Val1 AND A > Val2

- HASH = dovrei confrontare A con TUTTI i Valori dell' Intervallo
- B+TREE = Logaritmica per Primo Val dell' Intervallo e Scorro le foglie fino al Secondo Valore

Scelta dipende dal TIPO di Operazione Che faccio Sui miei Dati.

TEMPO DI RICERCA verrà IMPATIATO

Indice Sulla PRIMARY KEY Viene Creato di DEFAULT, visto che è il Più Comune.

GESTORE DELLA CONCORRENZA:

Riceve Richieste di Accesso ai dati decide se AUTORIZZARLE o differirle (Per Questo Chiamato SCHEDULER)

Input fornito al DBMS è una Transazione Ti che dopo il GESTORE OTTIMIZZAZIONI/INTERROGAZIONI ci produce il PIANO D'ESECUZIONE che fa parte delle RICHIESTE al GESTORE DEI METODI D'ACCESSO

Sia BUFFER che METODI D'ACCESSO si interfaccia con il GESTORE dell'Affidabilità (RIPRESA A CALDO & A FREDDO)

CARICO APPLICATIVO di un DBMS \Rightarrow #Transazioni al Secondo
TPS
Transaction \downarrow per Second

DBMS Garantisce un'ESECUZIONE CONCURRENTE delle TRANSAZIONI

PARALLELO \Rightarrow 2 Operazioni in Esecuzione Contemporaneamente
CONCURRENTE \Rightarrow Ho l'impressione che sia in Parallello ma in Realtà viene Eseguita un po' una e un po' l'altra

Può Generare Anomalie se non c'è CONTROLLO (problemi di Correttezza)

ANOMALIE TIPICHE:

- Perdita di Aggiornamento
- Lettura Inconsistente
- Lettura Sporca
- Aggiornamento fantasma
- Invecchiamento fantasma



Almeno 2 ti con Almeno 2 operazioni

ti Transazione

$r_i(x)$ lettura di t_i su RISORSA x

$w_i(x)$ scrittura di t_i su RISORSA x

PERDITA DI AGGIORNAMENTO:

Gli Effetti di una Transac. Concorrente Sono Andati Persi

$t_1 : r_1(x), x = x + 1, w_1(x), \text{Commit};$

$t_2 : r_2(x), x = x + 1, w_2(x), \text{Commit};$

Andiamo a SIMULARE:

OPERAZIONI t_1	MEMORIA CENTRALE t_1	MEMORIA SECONDARIA	MEMORIA CENTRALE t_2	OPERAZIONI t_2
		$x = 2$		
BOT				
$r_1(x)$	2	$\leftarrow [2]$		BOT
$x = x + 1$	3	$[2]$		

OPERAZIONI t_1	MEMORIA CENTRALE t_1	MEMORIA SECONDARIA	MEMORIA CENTRALE t_2	OPERAZIONI t_2
		[2]	2	$r_2(x)$
			3	$x = x + 1$
		[3]	3	$w_2(x)$
		[3]	3	COMMIT
		[3]	3	EOT
$w_1(x)$	3	$\rightarrow [3]$		

Velore finale $x=3$ Quando qualsiasi Esecuzione Seq.
Avrebbe Concluso con Velore 4.

Risultato \neq Exec. Seq. (pero \neq Aggiornamento)

Risorse è State già Acquisite da un Altra Transazione

LETTURA INCONSISTENTE:

Accessi successivi ad uno stesso dato/risorsa all'interno di una Transazione ritornano valori diversi

t_1 : $r_1(x)$, $r'_1(x)$, commit

t_2 : $r_2(x)$, $x = x + 1$, $w_2(x)$, commit

OPERAZIONI t_1	MEMORIA CENTRALE t_1	MEMORIA SECONDARIA	MEMORIA CENTRALE t_2	OPERAZIONI t_2
		[2]		
BOT				
$r_1(x)$	2	$\leftarrow [2]$		
		[2]		BOT
			$\rightarrow 2$	$r_2(x)$
		[2]	3	$x = x + 1$
		$\leftarrow [3]$	3	$w_2(x)$
				Commit
$r'_1(x)$	3	$\leftarrow [3]$		

RISULTATO Non Atteso, visto che t è INDIVISIBILE e non ho modificato x

LETTURA SPORCA:

Viene letto un dato che rappresenta uno Stato Intermedio
Nell'Evoluzione di una T

