Aspetti di progettazione fisica

Progettazione fisica

- Il risultato della progettazione logica è la definizione di uno schema relazionale normalizzato per il dominio applicativo
- Il passo successivo è quello di scegliere gli indici, effettuare clustering delle relazioni, e di raffinare lo schema concettuale e logico al fine di migliorare le performance del sistema
- Questa attività viene chiamata *progettazione fisica*
- Il risultato della progettazione fisica è lo schema fisico della base di dati
- È possibile ottenere diversi schemi fisici a partire dallo stesso schema logico (a seconda delle operazioni che si vogliono rendere più efficienti)

Workload

- Il workload è il carico di lavoro che deve essere effettuato dal sistema realizzato. Per determinare il workload occorre stabilire:
  - Le più importanti interrogazioni che verranno effettuate sulla base di dati e quanto spesso tali interrogazioni possono essere invocate
  - Le più importanti operazioni di modifica e quanto spesso tali operazioni possono essere richieste
  - La performance che il sistema deve garantire per tali interrogazioni e operazioni di modifica

Workload

- Inoltre, per ogni interrogazione occorre specificare:
  - Quali relazioni accede
  - Quali attributi restituisce
  - Quali attributi sono utilizzati per porre condizioni o effettuare operazioni di join. Quanto queste condizioni si presume sia no selettive
- Inoltre, per ogni operazione di modifica occorre specificare:
  - Quali attributi sono interessati da operazioni di join o utilizzati per porre condizioni. Quanto queste condizioni si presume sia no selettive
  - Il tipo di modifica (INSERT/DELETE /UPDATE) e gli attributi che ne sono influenzati
- Queste informazioni servono per prendere le successive decisioni

Decisioni da prendere

- Quali indici si dovrebbero creare?
  - Quali relazioni dovrebbero avere degli indici? Quale/i campo/i dovrebbero servire da chiavi di ricerca?
- Per ogni indice, che tipo di indice utilizzare?
  - Clusterizzato? Hash/tree? Dinamico/statico? Denso/sparso?
- È necessario fare delle modifiche allo schema concettuale?
  - Occorre considerare degli schemi normalizzati alternativi? (si ricordi che ci sono molti modi per decomporre uno schema in BCNF)
  - Occorre effettuare una operazione di “undo” dell’operazione di scomposizione dello schema e accontentarsi di una forma normale più debole? (denormalizzazione)
  - Partizionamento orizzontale, replicazione, viste…

Componenti di un DBMS

- Un DBMS contiene inoltre alcune strutture dati che includono:
  - i file con i dati (cioè i file per memorizzare il DB stesso)
  - i file dei dati di sistema (che includono il dizionario dei dati e le autorizzazioni)
  - indici (esempio Btree o tabelle hash)
  - dati statistici (esempio il numero di tuple in una relazione) che sono usati per determinare la strategia ottima di esecuzione

Prestazioni di un DBMS

- Un fattore importante nell'accettazione da parte dell'utente e' dato dalle prestazioni
- Le prestazioni del DBMS dipendono dall'efficienza delle strutture dati e dall'efficienza del sistema nell'operare su tali strutture dati

Prestazioni di un DBMS

- Esistono varie strutture alternative per implementare un modello dei dati
- La scelta delle strutture più efficienti dipende dal tipo di accessi che si eseguono sui dati
- Normalmente un DBMS ha le proprie strategie di implementazione di un modello dei dati
- tuttavia l'utente (esperto) può influenzare le scelte fatte dal sistema (**livello fisico**)

Clustering

- Si consideri la seguente interrogazione:

```
SELECT Imp#, Nome, Sede
FROM Impiegati, Dipartimenti
WHERE Impiegati.Dip# = Dipartimenti.Dip#
```

- una strategia di memorizzazione efficiente è basata sul raggruppamento (clustering) delle tuple che hanno lo stesso valore dell'attributo di join
- il clustering può rendere inefficiente l'esecuzione di altre interrogazioni
  - es. SELECT * FROM Dipartimenti

Clustering

10	Edilizia Civile	1100	D1	7977		
7782	Neri	ingegnere	01-Giu-81	2,450.00	200.00	10
7839	Dare	ingegnere	17-Nov-81	2,600.00	300.00	10
7934	Milli	ingegnere	23-Gen-82	1,300.00	150.00	10
7977	Verdi	dirigente	10-Dic-80	3,000.00	?	10
20	Ricerche	2200	D1	7566		
7369	Rossi	ingegnere	17-Dic-80	1,600.00	500.00	20
7566	Rosi	dirigente	02-Apr-81	2,975.00	?	20
7788	Scotti	segretaria	09-Nov-81	800.00	?	20
7876	Adami	ingegnere	23-Set-81	1,100.00	500.00	20
7902	Fordi	segretaria	03-Dic-81	1,000.00	?	20
30	Edilizia Stradale	5100	D2	7698		
7499	Andrei	tecnico	20-Feb-81	800.00	?	30
7521	Bianchi	tecnico	20-Feb-81	800.00	100.00	30
7654	Martini	segretaria	28-Set-81	800.00	?	30
7698	Blacchi	dirigente	01-Mag-81	2,850.00	?	30
7844	Turni	tecnico	08-Set-81	1,500.00	?	30
7900	Gianni	ingegnere	03-Dic-81	1,950.00	?	30

Efficienza- Strutture ausiliarie di accesso

- Spesso le interrogazioni accedono solo un piccolo sottoinsieme dei dati
- Per risolvere efficientemente le interrogazioni può essere utile allocare delle strutture ausiliarie che permettano di determinare direttamente i record che verificano una data query (senza scandire tutti i dati)
- I meccanismi più comunemente usati dai DBMS sono: indici, funzioni hash

Strutture ausiliarie di accesso

- Ogni tecnica deve essere valutata in base a:
  - tempo di accesso
  - tempo di inserzione
  - tempo di cancellazione
  - occupazione di spazio
- Molto spesso è preferibile aumentare l'occupazione di spazio se questo contribuisce a migliorare le prestazioni
- Si usa il termine chiave di ricerca per indicare un attributo o insiemi di attributi usati per la ricerca (diverso dalla chiave primaria)

Strutture ausiliarie di accesso

- Una ricerca può essere effettuata per:
  - chiave primaria: il valore della chiave identifica un unico record
    - Es. il contribuente con codice fiscale GRRGNN69R48
  - chiave secondaria: il valore della chiave puo' identificare piu' record
    - Es. i contribuenti di Genova
  - intervallo di valori (sia per chiave primaria che per secondaria)
    - Es. i contribuenti con reddito compreso tra 60 e 90 milioni
  - combinazioni delle precedenti
    - Es. i contribuenti di Genova e La Spezia con reddito compreso tra 60 e 90 milioni

Strutture ausiliarie di accesso

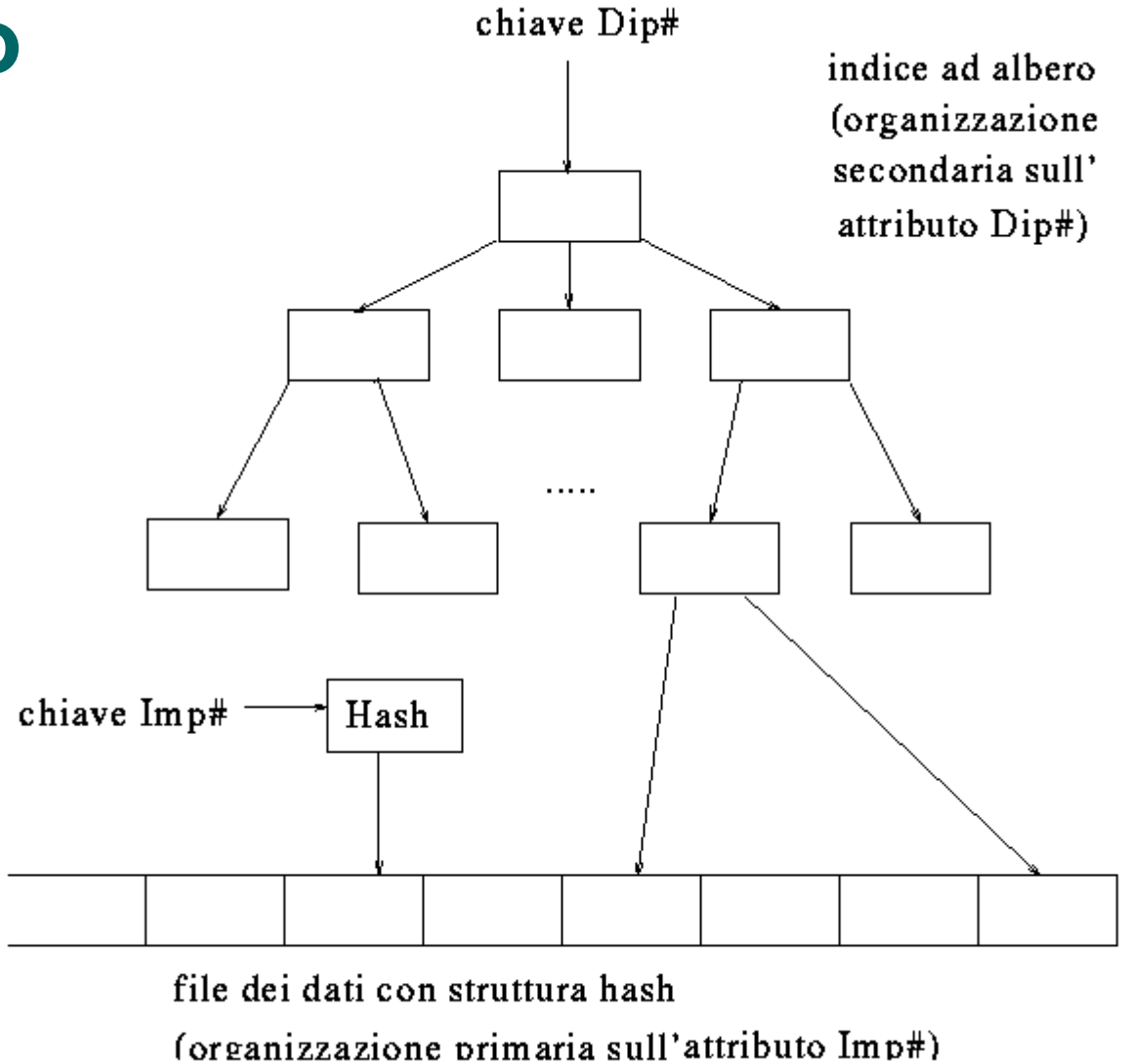
- Per effettuare la ricerca in modo più efficiente si può pensare di mantenere il file ordinato secondo il valore di una chiave di ricerca

- in questo caso però la ricerca su altri campi è inefficiente

Strutture ausiliarie di accesso

- **Organizzazioni primarie**:
  - impongono un criterio di allocazione dei dati (organizzazioni ad albero o hash)
- **organizzazioni secondarie**:
  - uso di **indici** (separati dal file dei dati) normalmente organizzati ad albero
- in generale si hanno a disposizione più modalità (cammini) di accesso ai dati

Strutture ausiliarie di accesso



Indici

- Idea base: associare al file dei dati una “tabella" nella quale l'entrata iesima memorizza una coppia (k_i ,r_i) dove:
  - k_i è un valore di chiave del campo su cui l'indice è costruito
  - r_i è un riferimento al record (eventualmente il solo) con valore di chiave k_i
    - il riferimento può essere un indirizzo (logico o fisico) di record o di blocco

Efficienza – Indici: esempio

- File dei dati:

c5	c2	c11	c7	c4
0	8	16	32	48

- indice:

chiave	indirizzo
c2	8
c4	48
c5	0
c7	32
c11	16

Indici

- Le diverse tecniche differiscono nel modo in cui organizzano l'insieme di coppie (k_i ,r_i)
- vantaggio nell'uso di un indice:
  - la chiave è solo parte dell'informazione contenuta in un record, quindi l'indice occupa meno spazio del file dei dati
- un indice può comunque raggiungere notevoli dimensioni che determinano problemi di gestione simili a quelli del file dei dati

indici

- Esempio: Indice per un file di 50k record, in cui i valori di chiave sono stringhe di 20 byte e i puntatori sono di 4 byte, richiede circa 1,2Mb
  - ogni entrata nell'indice: 20 + 4 byte
  - numero max entrate: 50 k
  - totale: 24 * 50 K = 1,2 Mb
- Possibile soluzione:
  - Creare un indice per un indice
  - Indici multilivello
  - Non li vediamo

Efficienza- Tipi di indice

- Gli indici possono essere classificati rispetto a diverse dimensioni:
  - unicità valori chiave di ricerca
  - numero di entrate nell'indice
  - ordinamento dei record nel file di dati

Unicità dei valori di chiave

- **Indice su chiave primaria**:
  - indice su un attributo che è chiave primaria per la relazione (a ogni entrata dell'indice corrisponde un solo record)

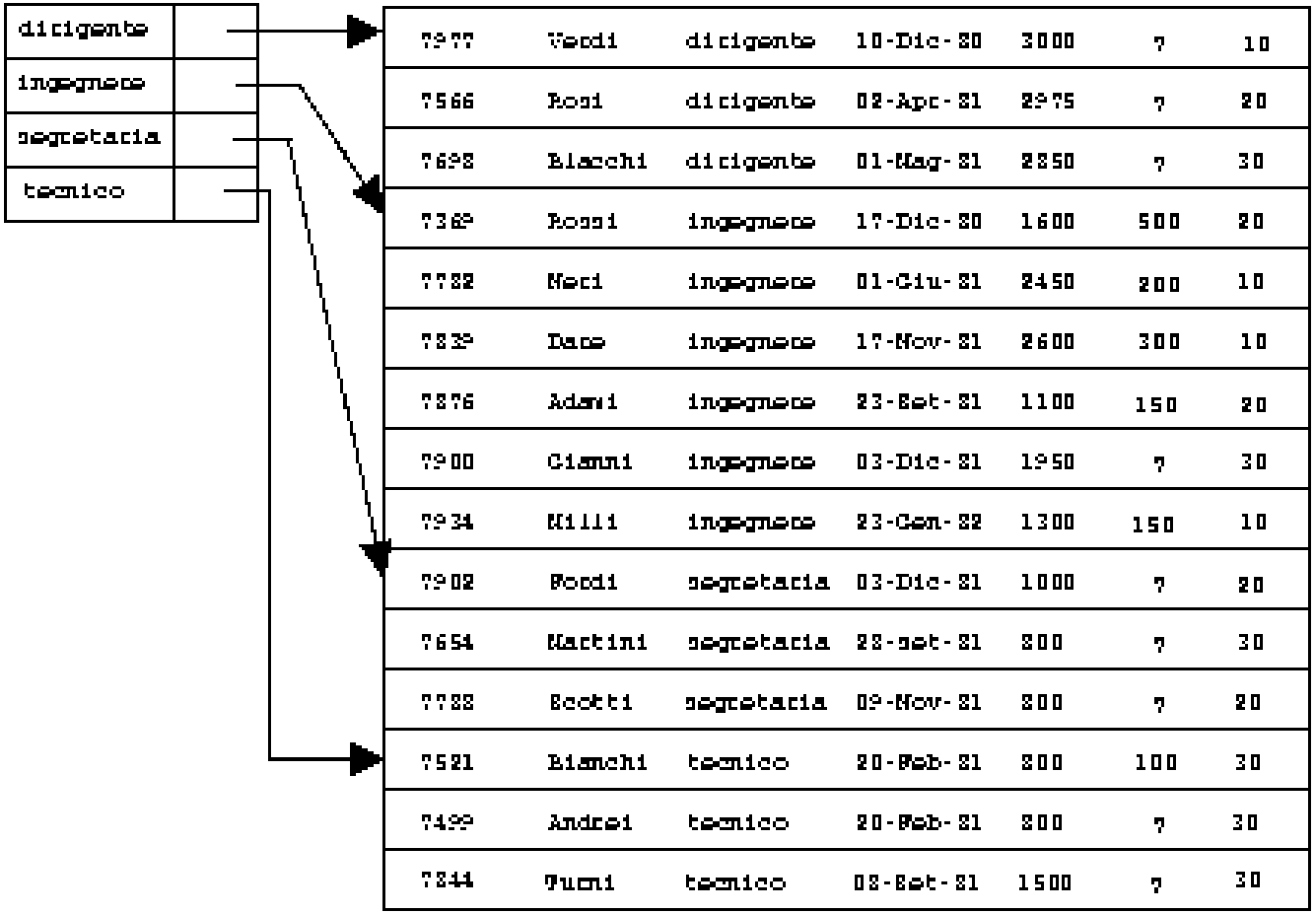
- **Indice su chiave secondaria**:
  - indice su un attributo che non è chiave primaria per la relazione (ad ogni entrata dell'indice possono corrispondere più record)

Numero di entrate dell'indice

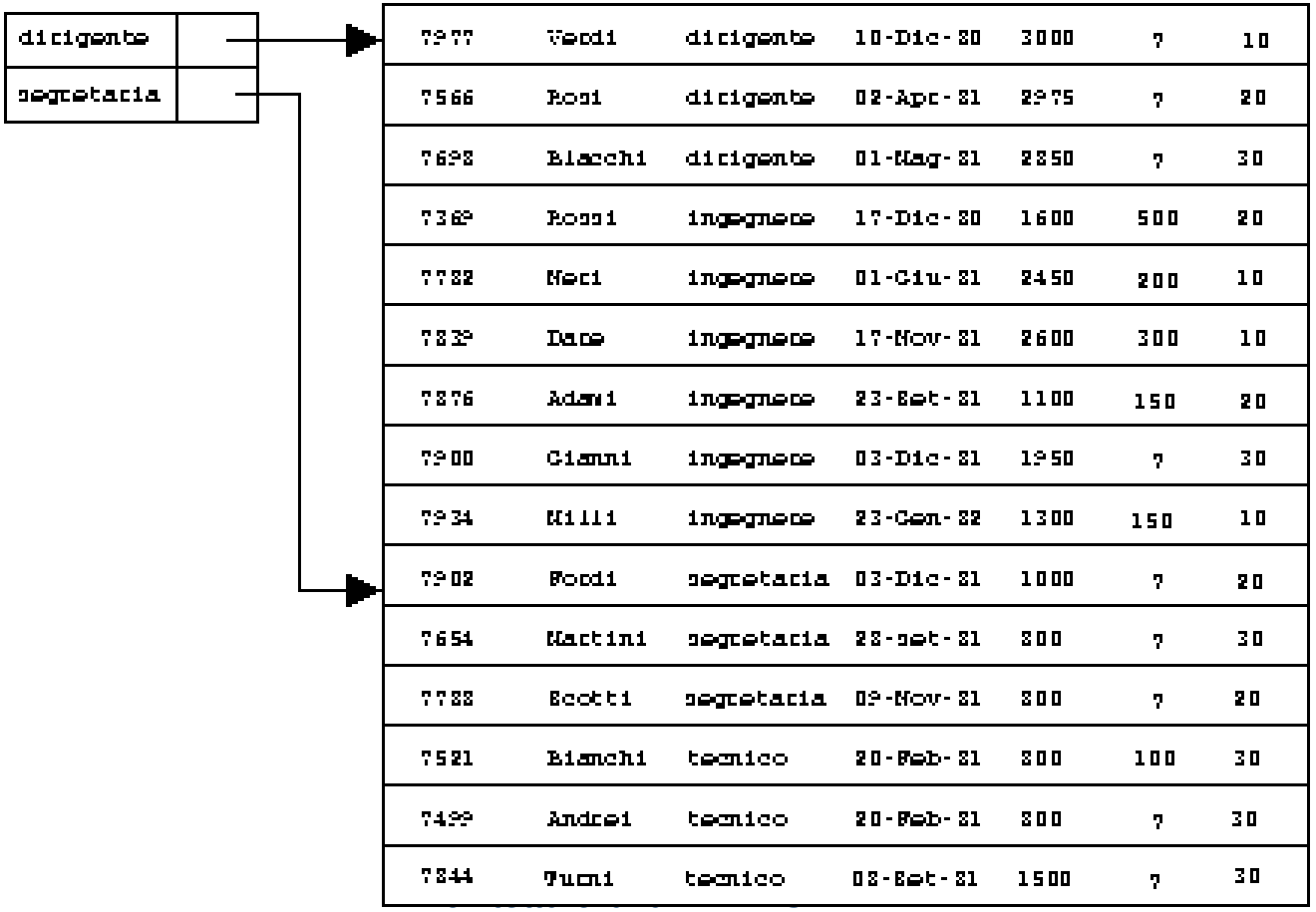
- **Indice denso**: l'indice contiene un'entrata per ogni valore della chiave di ricerca nel file
  - Ricerca:scansione per trovare il record con valore chiave uguale al valore cercato
  - recupero dati fino a che il valore non cambia

- **Indice sparso**: le entrate dell'indice sono create solo per alcuni valori della chiave.
  - Ricerca: scansione fino a trovare il record con il più alto valore della chiave che sia minore o uguale al valore cercato
  - scansione sequenziale del file dei dati fino a trovare il record cercato

Efficienza – Indice denso



Efficienza – indice sparso



Numero di entrate dell'indice

- Un indice denso consente una ricerca più veloce, ma impone maggiori costi di aggiornamento
- Un indice sparso è meno efficiente ma impone minori costi di aggiornamento
- Poiché molto spesso la strategia è di minimizzare il numero di blocchi trasferiti, un compromesso spesso adottato consiste nell'avere una entrata nell'indice per ogni blocco

Ordinamento record

- **Indice clusterizzato** (o indice primario):
  - indice sull'attributo secondo i cui valori il file dei dati e' mantenuto ordinato

- **indice non clusterizzato** (o indice secondario):
  - indice su un attributo secondo i cui valori il file dei dati non è mantenuto ordinato

Efficienza - Ordinamento record

- L'uso di più indici secondari rende l'esecuzione delle interrogazioni più efficiente, ma rende più costosi gli aggiornamenti
- quando si esegue l'inserzione o la cancellazione di un record è necessario modificare tutti gli indici allocati sul file
- gli indici secondari sono in genere di solito indici densi, mentre gli indici primari sono indici sparsi (notare che i record nel file dei dati sono ordinati in base al valore delle chiavi di ricerca dell'indice primario)

Tecniche

- Le strutture per MS differiscono da quelle per MM perché si cerca di minimizzare è il numero di blocchi acceduti (che determina il costo della ricerca)
  - basate su alberi
  - basate su tabelle hash

Alberi

- Btree e varianti
- organizzazioni ad albero binario di ricerca bilanciato in cui ogni nodo corrisponde a un blocco (quindi memorizza molti valori di chiave e tipicamente ha centinaia di figli)
- il costo delle operazioni è lineare nell'altezza dell'albero (logaritmico negli elementi memorizzati), raramente si hanno alberi di altezza superiore a tre

Hashing

- La funzione di hashing, data una chiave, restituisce l'indirizzo (blocco o bucket di blocchi) da cui partire per cercare i record con quel valore di chiave (in relazione al tipo di indice)
- il costo delle operazioni è costante (se la struttura è ben progettata)
  - questo vale se ogni indirizzo corrisponde ad un singolo blocco

Efficienza – Confronto tecniche

- L'uso di una tecnica piuttosto che di un'altra dipende spesso dal tipo di query
- Esempio: Se la maggior parte delle interrogazioni ha la forma:
  - select A1,A2,.....An from R where Ai=C

  la tecnica hash e' preferibile
  - la scansione di un indice ha un costo proporzionale al logaritmo del numero di valori in R per Ai
  - in una struttura hash il tempo di ricerca e' indipendente dalla dimensione del DB

Efficienza – Confronto tecniche

- Strutture ad albero sono preferibili se le interrogazioni usano condizioni di range
  - select A1,A2,.....An from R where C1 <Ai <C2
- infatti e' difficile determinare funzioni hash che mantengono l'ordine
- Quasi tutti i sistemi usano strutture di indici ad albero perché è difficile prevedere a priori il tipo di interrogazioni

Definizione di cluster e indici in SQL

- La maggior parte dei DBMS relazionali fornisce varie primitive che permettono al progettista della base di dati di definire la configurazione fisica dei dati
- Queste primitive sono rese disponibili all'utente come comandi del linguaggio SQL
- I comandi più importanti sono il comando per la creazione di indici (CREATE INDEX), su una o più colonne di una relazione, e il comando per la creazione di cluster (CREATE CLUSTER)

Linee guida nella scelta del tipo di indice da utilizzare

- Presentiamo una serie di passi che dovrebbero essere seguiti per stabilire quando è utile utilizzare un indice su una relazione e quale indice utilizzare rispetto a quelli che abbiamo presentato
- Ovviamente solo le relazioni che sono accedute attraverso una query possono richiedere l'uso di un indice
- È importante considerare le query più importanti, cioè quelle query che influenzano pesantemente il workload del sistema

Un approccio per scegliere gli indici

- si considerano le interrogazioni che sono più importanti. Si considera il piano di esecuzione migliore utilizzando gli indici correntemente definiti
- Si prova a vedere se si riesce a ottenere un piano di esecuzione migliore aggiungendo un indice
- Se si, lo si crea
- Prima di creare un indice, occorre sempre considerare l'impatto di tale indice sulle operazioni di modifica previsti nel workload!
  - Trade-off: gli indici possono rendere particolarmente veloci le interrogazioni, ma la loro modifica è costosa e richiedono spazio sul disco per essere memorizzati

Linea Guida 1: Quando inserire un indice

- Non costruire un indice inutile per una interrogazione – comprese le query di selezione delle operazioni di update
- Scegliere sempre indici che migliorano le prestazioni di più di una interrogazione

- Consigli ovvi…ma importanti!

Linea guida 2: Scelta della chiave di ricerca

- Gli attributi che appaiono nella clausola WHERE sono candidati a ricevere un indice:
  - indici hash vengono usati in caso di condizioni di uguaglianza (es. nome =“Rossi”)
  - Indici organizzati ad albero sono utili quando si pongono condizioni su range di valori (es: year BETWEEN 2000 AND 2002)
  - Il clustering è specialmente utilizzato per interrogazioni su range di valori, anche se può essere utile in condizioni di uguaglianza (ad es. nel caso in cui ci sono duplicati)

Linea guida 3: chiavi di ricerca su attributi multipli

- Indici su chiavi di ricerca definiti su attributi multipli si considerano nei seguenti casi:
  - La clausola WHERE contiene condizioni su più di un attributo di una relazione
  - Permettono di accedere esclusivamente all'indice (*index only evaluation strategies*). Cioè la relazione non viene acceduta. (si noti che questo può portare a chiavi di ricerca con più attributi di quelli specificati nella clausola WHERE)
- Se sono presenti condizioni su range di valori, l'ordine degli attributi deve essere scelto con grande attenzione in modo da "matchare" l'ordine del range di valori

Linea guida 4: quando clusterizzare?

- Al più un indice può essere clusterizzato per una relazione. Dal momento che il clustering ha un profondo effetto nelle prestazioni del sistema occorre scegliere in modo accurato l'attributo su cui effettuare il clustering
  - Intuitivamente, in presenza di interrogazioni su range di valori utilizzare il clustering. Se più interrogazioni sono specificate su range di valori per una determinata relazione, su attributi diversi, nella scelta dell'attributo da clusterizzare si consideri la selettività delle varie clausole WHERE e la relativa frequenza rispetto al carico di lavoro.
  - Nel caso in cui si utilizzano strategie di interrogazioni che accedono solo l'indice per interrogazioni particolarmente importanti la presenza di un indice clusterizzato non è rilevante

Linea guida 5: Hash Vs tree index

- Abbiamo già visto prima…ad ogni modo… in interrogazioni con condizioni di uguaglianza è meglio utilizzare hash index,
- Mentre, in interrogazioni con condizioni su range di valori è meglio usare le strutture ad albero

Linea guida 6: Valutare il costo del mantenimento dell'indice

- Una volta stabilita la lista degli indici che occorrerebbe creare, si consideri l'impatto che ogni indice sulle operazioni di modifica previste nel workload
  - Se il mantenimento di un indice riduce le prestazioni di operazioni di modifica molto frequenti, si valuti la possibilità di eliminare l'indice
  - Si ricordi, comunque, che la presenza di un indice può migliorare anche le prestazioni delle operazioni dimodifica

    (ad esempio un indice sull'ID di un impiegato può rendere più efficiente l'operazione di aumentare il salario di un dato impiegato– specificato attraverso l'ID)

```
SELECT E.ename, D.mgr
FROM Emp E, Dept D
WHERE D.dname= “Toy”
        AND E.dno=D.dno
```

Esempio

- Un indice Hash su D.dname permette la selezione di “Toy”
  - Dato questo indice un indice su D.dno non serve. Si suppone che il numero di tuple che soddisfa la condizione su dname sia basso
- Un indice hash su E.dno permette di selezionare le tuple di Emp

```
SELECT E.ename, D.mgr
FROM Emp E, Dept D
WHERE D.dname= "Toy"
        AND E.dno=D.dno
```

Esempio

- Cosa succede se la clausola WHERE include una condizione: "… AND E.age=25"?
  - Si potrebbe recuperare le tuple usando un indice su E.age, quindi effettuare in join con le tuple di Dept che soddisfano la selezione su dname. Questa soluzione è confrontabile con quella che utilizza l'indice su E.dno.
  - Quindi, se l'indice su E.age è già stato creato, questa interrogazione non richiede di introdurre un indice su E.dno