Indici per il Data Warehouse

DATA WAREHOUSE

- Riferimento bibliografico:

  "DATA WAREHOUSE", *Golfarelli e Rizzi*, Mc Graw-Hill, 2005 II edizione

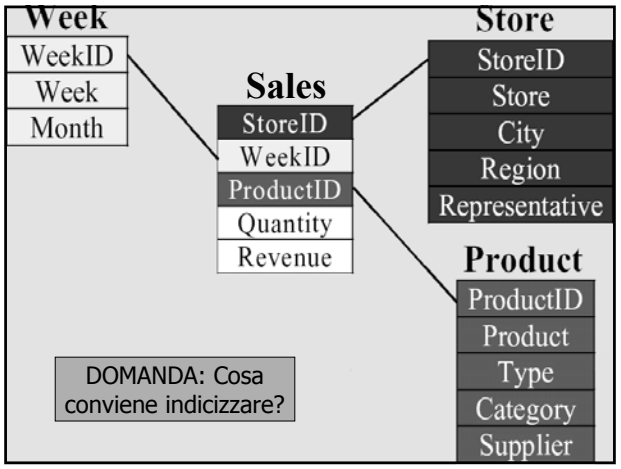
Indici per il Data Warehouse

- Le tecniche di indicizzazione ci sono già note dai DBMS.
- L'uso degli indici in un DW è particolarmente importante ed insieme alla "materializzazione delle viste" costituisce la tecnica più efficace per aumentare le prestazioni del sistema

Indici per il Data Warehouse

- Nei DBMS commerciali la più nota tecnica di indicizzazione è rappresentata dai B^+-Tree.
- In un DW si possono usare:
  - Tecniche specifiche per il DW
  - Tecniche già in uso nei DBMS
- In DW in cui abbiamo situazioni di sola lettura (scritture soltanto durante il processo di alimentazione) è possibile usare tecniche poco usate nei DBMS a causa del loro elevato costo

➔ Possiamo sfruttare la riorganizzazione periodica degli indici (refresh del DW)



Indici per il Data Warehouse

Alcune esigenze particolari che possono guidarci:

- costruire indici su attributi non chiave delle Dimension Table per accelerare le operazioni di selezione
- costruire indici sulle chiavi esterne della Fact Table per accelerare l'esecuzione delle join
- costruire indici sulle misure della Fact Table

B+-TREE

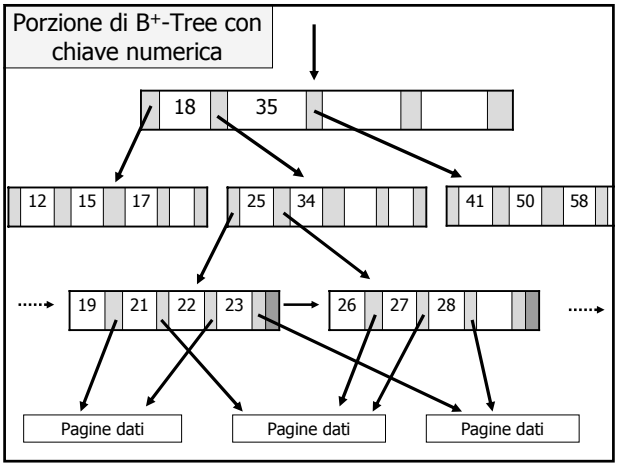
B+-Tree

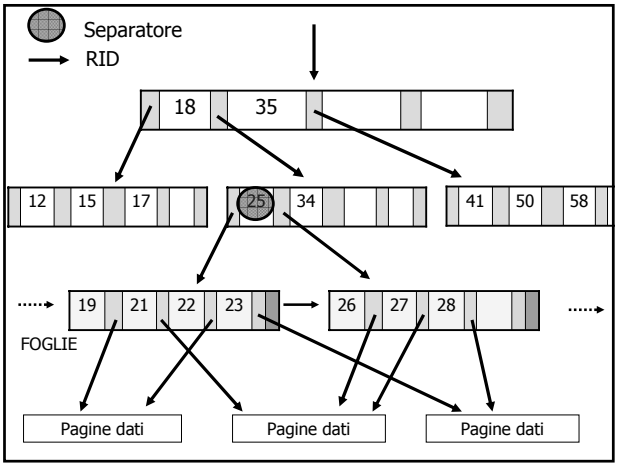
- Un indice B+-Tree per un attributo c di una relazione R è un albero bilanciato che consente accessi alle tuple di R tramite i valori di c.
- Le foglie dell'albero (concatenate tra loro) possiedono i puntatori alle pagine dati contenenti i record.
- I nodi intermedi, costituiti da una sequenza alternata di puntatori ai figli e di "separatori" consentono la rapida localizzazione delle foglie.

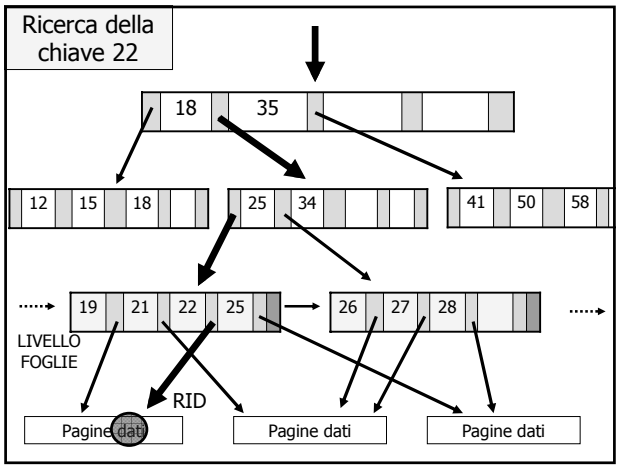
B+-Tree

- Un indice B+-Tree può essere costruito:
  - Su chiave primaria (un solo record per ogni valore di chiave)
  - Su chiave secondaria (più record per ogni valore di chiave): in tal caso, le foglie contengono la lista dei puntatori relativi alle tuple che presentano un certo valore di chiave ➔ RID (Row Identifier) : puntatore alla tupla.

Esempio di B+-Tree con chiave numerica

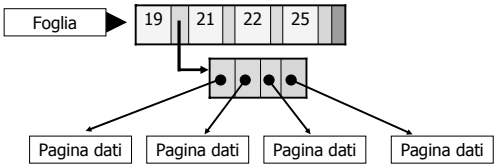






B+-Tree

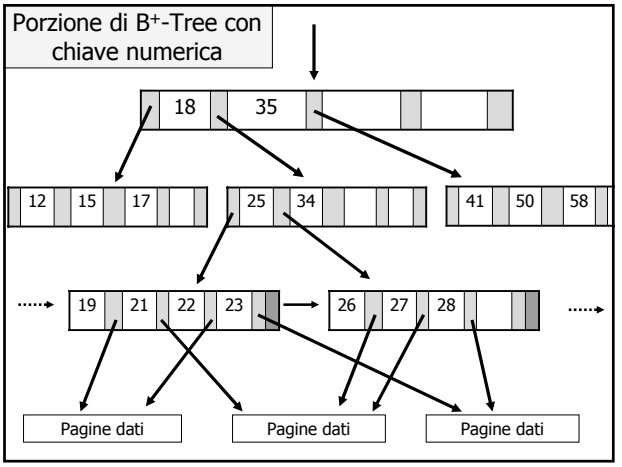
- Nel caso di più record con lo stesso valore di chiave (chiave secondaria), le foglie contengono la lista di RID relativi alle tuple che presentano quel particolare valore di chiave



B+-Tree

Domanda

- Come si effettua la ricerca su di un intervallo di valori [*a*, *b*] ?



B+-Tree

RISPOSTA

- Nel caso di ricerca su un intervallo di valori [*a*, *b*] occorre scendere nell'albero utilizzando *a* come valore di chiave e poi basta seguire la sequenza di nodi foglia fino ad incontrare un valore superiore di *b*.

B+-Tree

- VALUTAZIONE DEI B+-Tree
  - Si dimostrano particolarmente utili quando:
    - è alta la selettività di un attributo;
    - è semplice la query.
  - Lo spazio richiesto può essere molto elevato
    - in un DW, in cui sono richiesti indici su di un notevole numero di attributi, la dimensione degli indici può variare da 2 a 6 volte il volume dei dati grezzi (non indicizzati).

B+-Tree

- VALUTAZIONE DEI DW:
  - Le query sui DW spesso riguardano selezioni su attributi con basso grado di selettività.
  - Le query (complesse) sui DW richiedono molto spesso join ed aggregazioni.

B+-Tree

→ Necessità di adottare indici che siano più adatti dei B+ -Tree

- CONSIDERANDO CHE:
  - Gli accessi a sola lettura dei DW consentono di utilizzare tecniche di indicizzazione che possono anche avere costi alti per quanto riguarda le operazioni di scrittura (es. Update) e che solitamente non vengono usati in OLTP
- In alternativa:
  - Utilizzare alcuni indici che sono stati pensati e progettati appositamente per i DW.

Indici

- Bitmap Index
- Join Index
- Star Index
- Bitmapped Join Index

Bitmap Index

Bitmap Index

- Un indice costruito come una mappa di bit !?!
- Rappresentano uno dei migliori risultati nel settore della progettazione fisica di un DW anche se è utilizzabile solo in precise situazioni.

Bitmap Index

- Un Bitmap Index per un attributo c di una relazione R è composto da una matrice di bit con
  - tante *colonne* quanti sono i possibili valori dell'attributo (cardinalità del dominio dell'attributo)
  - tante *righe* quante sono le tuple della relazione R.

Bitmap Index

- Per ogni possibile valore dell'attributo, un bit (0/1) indica se il corrispondente record possiede quel valore

  ossia

  Il bit $B_{i,j}$ è posto uguale a 1 se l'attributo c della i-esima tupla presenta il valore c_j (ossia il j-esimo valore per c)

Bitmap Index

Esempio: Indice Bitmap sull'attributo **Posizione** della tabella **Impiegati**. i possibili valori sono:

'Ingegnere' - 'Consulente' - 'Manager' - 'Programmatore' - 'Segretario' - 'Ragioniere'

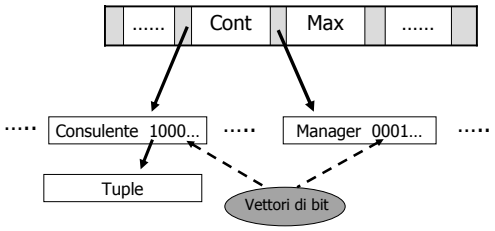
RID	Ing.	Cons.	Man.	Prog.	Segr.	Rag.
1	0	0	1	0	0	0
2	0	0	0	1	0	0
3	0	0	0	0	1	0
4	0	0	0	1	0	0
5	0	0	0	0	0	1

- La colonna RID (Row Identifier) permette di identificare a livello fisico la tupla della relazione. E' il PUNTATORE alla tupla.

Bitmap Index

- Organizzati con una struttura ad albero:
  - al fine di velocizzare l'individuazione delle colonne di interesse, gli indici Bitmap sono organizzati in una struttura ad albero, del tutto simile a quella dei B^+-tree in cui le foglie contengono vettori di bit invece che liste di RID.

Gli indici Bitmap sono organizzati in una struttura ad albero simile a quella dei B^+-Tree

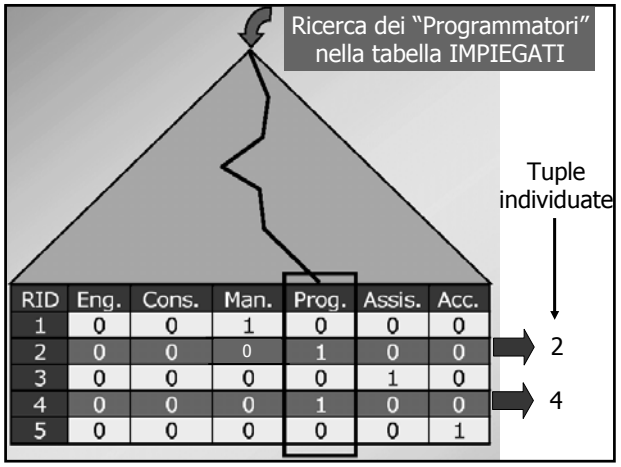


Rappresentazione di un indice bitmap associato all'attributo 'Posizione'.

Le foglie dell'albero contengono i diversi valori dell'attributo associati ai relativi vettori di bit.

Bitmap Index

- La ricerca delle tuple che presentano un particolare valore avviene nelle seguenti fasi:
  1. Discesa nell'albero
  2. Caricamento in memoria del corrispondente vettore di bit
  3. Individuazione dei RID con bit posti a 1
  4. Recupero delle tuple corrispondenti



Esempio

Cust	Regione	Tipo
C1	Asia	Retail
C2	Europa	Dealer
C3	Asia	Dealer
C4	America	Retail
C5	Europa	Dealer

Tabella Base

Indice sull'attributo REGIONE

RID	Asia	Europa	America
1	1	0	0
2	0	1	0
3	1	0	0
4	0	0	1
5	0	1	0

Indice sull'attributo TIPO

RID	Retail	Dealer
1	1	0
2	0	1
3	0	1
4	1	0
5	0	1

Bitmap Index

- Grazie alla rappresentazione binaria, gli indici Bitmap si prestano ad operare con operatori booleani (AND, OR, NOT) che sono molto utilizzati nelle interrogazioni OLAP per filtrare dati in presenza di più condizioni.

→Talvolta è possibile risolvere un'interrogazione senza nemmeno accedere ai dati.

- Esempio:

  Un DW costruito per una compagnia di assicurazioni contiene informazioni sugli abitanti di un certa regione (es. sesso, provincia di residenza e se sono o meno assicurati)

RID	Sesso	Assicurato	Provincia
1	M	N	MS
2	M	Y	FI
3	F	N	LU
4	M	Y	FI

- Interrogazione:

  Quanti sono gli uomini della provincia di Firenze che sono assicurati?

RID	Sesso	Assicurato	Provincia
1	M	N	MS
2	M	Y	FI
3	F	N	LU
4	M	Y	FI

Indice Bitmap sull'attributo Sesso

RID	M	F
1	1	0
2	1	0
3	0	1
4	1	0

Indice Bitmap sull'attributo Assicurato

RID	Y	N
1	0	1
2	1	0
3	0	1
4	1	0

Indice Bitmap sull'attributo Provincia

RID	FI	PI	LI	LU	MS	.....
1	0	0	0	0	1	.....
2	1	0	0	0	0	.....
3	0	0	0	1	0	.....
4	1	0	0	0	0	.....

RID	M	Y	FI
1	1	0	0
2	1	1	1
3	0	0	0
4	1	1	1

Risposta dell'interrogazione: 2

Abbiamo risposto senza accedere ai dati !!!

Confronto tra B$^+$-Tree e Bitmap

- Entrambi velocizzano le operazioni di selezione tipicamente presenti nelle elaborazioni OLAP
- Le loro caratteristiche strutturali ne fanno variare le prestazioni in base alle caratteristiche dell'attributo indicizzato

Confronto tra B$^+$-Tree e Bitmap

- La differenza principale sta nel modo in cui vengono memorizzati gli insiemi di RID
- **B$^+$-Tree**:
  - Memorizzano in modo esplicito i RID per ciascun valore della chiave
- **Bitmap**:
  - Ad ogni valore della chiave associano un vettore di bit di lunghezza uguale al numero di tuple della relazione (ossia al numero totale di RID)

  → la dimensione degli indici Bitmap dipende fortemente dal numero di valori distinti che l'attributo può assumere

Confronto tra il numero di foglie necessarie per memorizzare un Bitmap e un B+ Tree in funzione del numero di valori distinti della chiave.

La dimensione del Bitmap cresce rapidamente al crescere del numero di valori distinti di chiave, mentre la dimensione del B$^+$-Tree rimane costante è indipendente dal variare della chiave.

Bitmap Index

- Occupazione di un Bitmap Index
  - Dipende dal numero di valori distinti di chiave
  - Per ogni valore distinto di chiave
    - Si memorizza il vettore di bit: la dimensione del vettore è uguale al numero di tuple
  - Sia NR il numero di tuple
  - Sia NK il numero di valori distinti di chiave

INDICE BITMAP: NR * NK * 1 bit
RID List: NR * Len(Pointer)
*con Len(Pointer)= 4 byte =4 * 8bit = 32 bit*

Per cui: **INDICE BITMAP < RID List ↔ NK < 32**

Bitmap Index

- CONCLUSIONE:

Al crescere del numero di valori distinti della chiave si ha che:

1. La dimensione del B^+-Tree rimane pressoché costante
2. La dimensione del Bitmap Index cresce molto rapidamente e supera l'altra quando il numero di valori distinti di chiave diventa maggiore del numero di bit utilizzati per codificare un RID.

SPARSITA'

ESEMPIO
Indice Bitmap sull'attributo Provincia

RID	FI	PI	LI	LU	MS	…….
1	0	0	0	0	1	…….
2	1	0	0	0	0	…….
3	0	0	0	1	0	…….
4	1	0	0	0	0	……..

Quando il numero di valori possibili è molto alto, la matrice di bit che ne deriva è molto SPARSA

Bitmap Index

- Concludendo:
  - Indici Bitmap adatti ad indicizzare attributi con un dominio limitato; in tal caso:
    - Minor spazio richiesto
    - Minor costo di I/O per l'esecuzione delle query
  - Ottimo per query che non necessitano di accedere al file dati

→ Riguardo alla SPARSITA':
Sono state studiate varie tecniche per la compressione delle matrici di bit (Chan, 1998) che permettono di ridurre notevolmente lo spazio occupato su disco.

Join Index

Introduzione

- Nonostante che lo schema a stella tenda a ridurre il numero di join necessari a reperire i dati, il costo delle interrogazioni OLAP continua ad essere determinato da questo tipo di operazione.
- L'esecuzione di query su schemi a stella richiede join (anche se non multilivello) che risulta essere l'operazione più dispendiosa.

Join Index

→

concentriamo la nostra attenzione sull'operazione di join tra Fact Table e Dimension Table

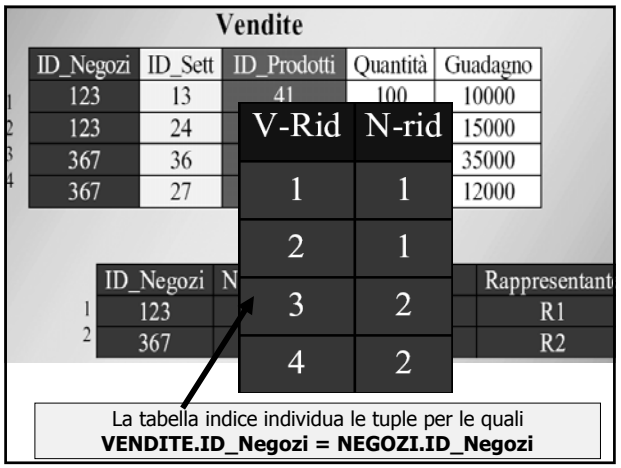
OBIETTIVO:

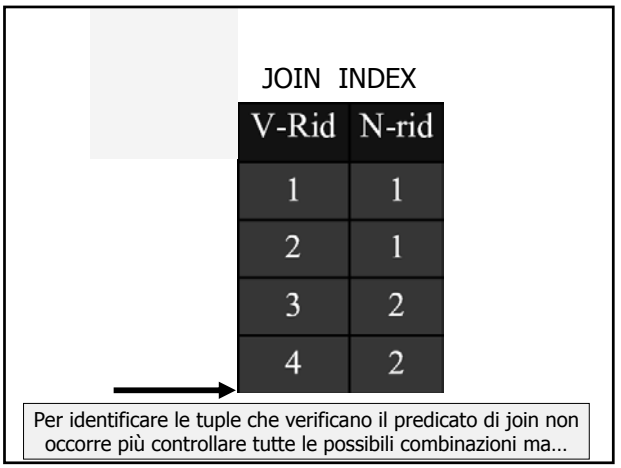
cercare di identificare a priori le tuple che soddisfano il predicato di join e creare un indice apposito

Join Index

FILOSOFIA:

- I Join Index migliorano le performance perché precalcolano e memorizzano i RID delle tuple corrispondenti nelle due relazioni.
- A quel punto:
  per verificare quali tuple soddisfano le condizioni di join non è necessario verificare tutte le possibili combinazioni, ma è sufficiente scandire l'indice.





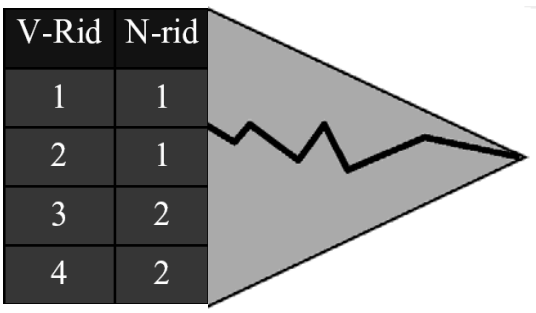
Join Index

DEF.

Il Join Index tra due relazioni R1 e R2 memorizza coppie di RID che soddisfano la condizione di join

c1 <operatore> c2

tra due attributi c1 e c2 appartenenti rispettivamente a R1 e R2.



In presenza di condizioni di selezione aggiuntive, i Join Index vengono associati ad un B^+-Tree in modo da evitarne la scansione completa

(si riduce il numero di RID da considerare)

FACT TABLE Vendite

RID	chiaveN	chiaveD	ChiaveP	Qtà	Incasso	.....
1	1	1	1	13	1500	.....
2	2	1	2	143	2334	.....
3	3	2	3	23	344	.....
4	2	2	1	21	2334	
5	1	2	1	25	3331	

DIM. TABLE Negozi

RID	chiaveN	Negozio	Città	regione	.....
1	1	coop1	Bologna	Emilia	.....
2	2	coop2	Roma	Lazio	.....
3	3	coop3	Roma	Lazio	.....

Se si pone la condizione negozio='coop1' allora si può evitare la scansione completa dell'indice trovando i RID della DT Negozi per cui è valida la condizione e poi si accede al Join Index

RID-NEGOZI	RID-VENDITE
1	1
1	5
2	2
2	4
3	3

Join Index

- Notevole miglioramento delle performance:
  - Vengono precalcolati e memorizzati i RID delle tuple corrispondenti nelle due relazioni
  - Per verificare quali tuple soddisfano le condizioni di join non è più necessario verificare tutte le possibili combinazioni ma basta scandire l'indice (eventualmente con l'ausilio di un B^+-Tree)

Join Index

- Tecnica che non viene adoperata nei sistemi operazionali: ha un costo di aggiornamento troppo alto perché
  - → quando una nuova tupla viene aggiunta ad una delle due tabelle è necessario aggiornare l'indice trovando tutte le tuple che soddisfano la condizione di join
- Tecnica accettabile per il DW in cui l'aggiornamento dei dati avviene periodicamente (fuori linea ad intervalli di tempo stabiliti)

Star Join Index e Bitmapped Join Index

Esigenza

Le Interrogazioni OLAP

Esempio: *VENDITE*(*Settimane.Mese*, *Prodotto.Fornitore*, *Negozio.Città*;*Prodotto.Tipo*="Sport",*Negozio.Stato*="I").*Quantità*

```
SELECT Settimane.Mese, Prodotti.Fornitore, Negozi.Città,
       SUM(vendite.Quantità)
FROM Vendite, Negozi, Settimane, Prodotti
WHERE Vendite.ID_Negozi = Negozi.ID_Negozi
AND Vendite.ID_Settimane = Settimane.ID_Settimane
AND Vendite.ID_Prodotto = Prodotti.ID_ Prodotti
AND Prodotti.Tipo = "Sport" AND Negozi.Stato = "I"
GROUP BY Settimane.Mese, Prodotti.Fornitore, Negozi.Città
```

Join

Esigenza

- Esigenza operativa:

  le interrogazioni OLAP coinvolgono solitamente più tabelle dimensionali (oltre alla Fact Table) → necessità di più operazioni di JOIN tra ogni riga della Fact e le corrispondenti righe delle DT (Star Join Query)

- Soluzioni proposte:
  - STAR JOIN INDEX (generalizzazione del join index)
  - BITMAPPED JOIN INDEX (mix tra bitmap e join index)

Star Join Index

Star Join Index

- Lo STAR JOIN INDEX generalizza il concetto del Join Index (che coinvolge due tabelle) al Join Multi-Tabella agganciando insiemi di RID appartenenti a tabelle diverse

  (come è appunto il caso del join tra una FACT TABLE ed n DIMENSION TABLE).

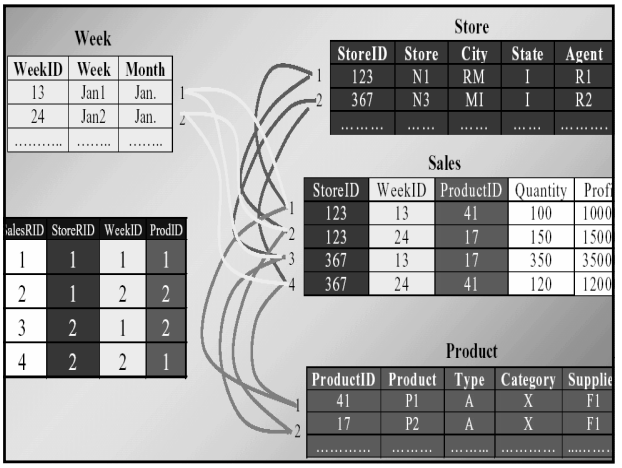
Star Join Index

- **DEF.**

  Uno STAR JOIN INDEX tra una FT e le n DT memorizza gli (n+1) RID delle n DT e 1 della FT, corrispondenti alle tuple che soddisfano le n condizioni di join $d_i=ft_i$

Chiave primaria della i-esima DT

Corrispondente i-esima chiave esterna nella FT



Star Join Index

- TENERE PRESENTE:

  Essendo creati su di una lista ordinata di colonne, la struttura dell'indice dipende dall'ordinamento delle colonne stesse

→ciò influenza i tempi di accesso all'indice che pertanto dipendono molto dall'ordinamento delle colonne

Star Join Index

- Sono molto efficienti quando le query coinvolgono tutte o solo le colonne iniziali dell'indice
- Per avere efficienza sempre, occorrerebbe avere tanti star join index quante sono le possibili permutazioni dell'insieme delle dimensioni

  (il numero cresce esponenzialmente al crescere del numero delle dimensioni)

Star Join Index

- Tali inconvenienti possono essere superati utilizzando un altro tipo di indicizzazione che costituisce una sorta di ibrido tra i bitmap index e gli star join index:

  → Bitmapped join index ←

  che rappresentano la soluzione più evoluta di join multi-tabella

Bitmapped Join Index

Bitmapped Join Index

- **DEF.**

  Un BJI sugli attributi c_R della relazione R e c_s della relazione S è una matrice B di bit con:
  - tante righe quante sono le tuple di R
  - tante colonne quante sono le tuple di S.

  Il bit $B_{i,j}$ è posto a 1 se la i-esima riga della relazione R e la j-esima riga della relazione S soddisfano la condizione di join.

FACT TABLE Vendite

RID	chiaveN	chiaveD	ChiaveP	Qtà	Incasso	.....
1	1	1	1	13	1500	.....
2	2	1	2	143	2334	.....
3	3	2	3	23	344	.....
4	2	2	1	21	2334	
5	1	2	1	25	3331	

DIM. TABLE Negozi

RID	chiaveN	Negozio	Città	regione	.....
1	1	coop1	Bologna	Emilia	.....
2	2	coop2	Roma	Lazio	.....
3	3	coop3	Perugia	Umbria	.....

Bitmapped Join Index →

RID-VENDITE	1	2	3
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0
5	1	0	0

Bitmapped Join Index

- Per risolvere il join tra le due tabelle, è sufficiente scorrere le righe dell'indice individuando la colonna che presenta il valore 1

Bitmapped Join Index

MORALE

"Il BJI è un Bitmap Index che si occupa di mappare le condizioni di join."

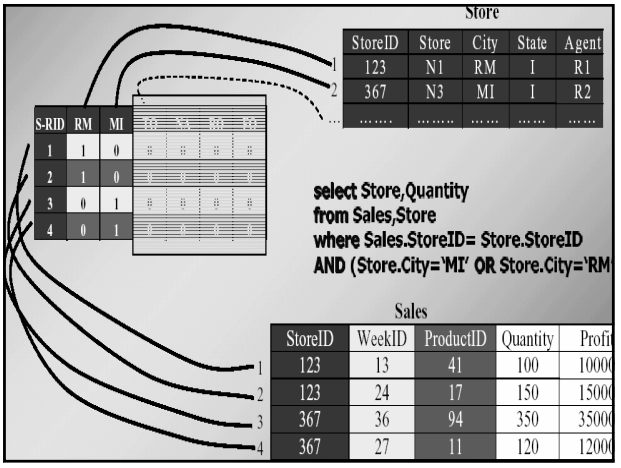
INOLTRE…

"Pur realizzando il join tra due tabelle, il BJI è usabile anche per query che prevedono join multipli."

Bitmapped Join Index

- **Condizioni di selezione aggiuntive:**

  Se ci sono condizioni di selezione che restringono il numero di RID da considerare, basta restringere la ricerca nell'indice considerando solo i vettori di bit che corrispondono a valori che soddisfano le suddette condizioni



Bitmapped Join Index

- Pur realizzando un join tra due tabelle, il bitmapped join index può essere generalizzato al multi-join

  (come del resto è nei casi reali in cui la Fact Table è in multi-join con varie Dimension Table).

Bitmapped Join Index

Sia data la seguente interrogazione (STAR JOIN QUERY):

SELECT distinct FT.m, $DT_1.a_1$, $DT_2.a_2$,....,$DT_n.a_n$

Condizioni di Join

WHERE $FT.a_1=DT_1.a_1$ AND $FT.a_2=DT_2.a_2$ AND… $FT.a_n=DT_n.a_n$

AND $DT_1.b_1=val_1$ AND $DT_2.b_2=val_2$ AND … $DT_n.b_n=val_n$

Condizioni aggiuntive

Bitmapped Join Index

Data l'interrogazione precedente, supponiamo di avere:

- un BJI per ogni condizione di join $FT.a_i=DT_i.a_i$
- un bitmap index per ogni attributo dimensionale coinvolto (es. $DT_i.b_i$)

→ E' possibile combinare l'uso dei **Bitmap Index** con l'uso dei **Bitmapped Join Index** ed è possibile determinare un piano di esecuzione dell'interrogazione come descritto di seguito.

PIANO DI ESECUZIONE

dell'interrogazione

1. Per ogni DT si identificano i RID che soddisfano le condizioni sugli attributi dimensionali, tramite il Bitmap Index
2. Per ogni BJI si caricano solo i vettori di bit corrispondenti ai suddetti RID; eseguendo un OR bit a bit si ottiene il vettore RID_i che soddisfa tutte le condizioni relative ad una DT
3. Le tuple della FT che soddisfano l'interrogazione vengono determinate eseguendo un AND bit a bit tra gli n vettori precedentemente creati

Bitmapped Join Index

La sequenza di esecuzione è concettualmente simile a quella da adottare disponendo di uno star join index. Ma:

- il numero di star join necessari a risolvere in modo efficiente interrogazioni comprendenti condizioni arbitrarie di join cresce esponenzialmente con il numero di DT
- il numero di BJI cresce linearmente poiché l'insieme potenza delle condizioni è ottenuto applicando l'operatore AND ai diversi vettori di bit

RID	val_1	val_2	...	val_i	...	val_n
1	1	0	...	0	...	0
2	0	0	...	0	...	1
3	0	1	...	0	...	0
4	0	0	...	1	...	0
5	0	0	...	1	...	0

INDICE Bitmap sull'attributo $DT_i.b_i$

RID della DT_i

Si trovano i RID che soddisfano la condizione sull'attributo dimensionale, ossia $DT_i.b_{i\,=}\ val_i$

PASSO 1:

Per ogni DT si accede all'indice Bitmap e si identificano i RID che soddisfano le condizioni sugli attributi dimensionali.

RID	val_1	val_2	...	val_i	...	val_n
1	1	0	...	0	...	0
2	0	0	...	0	...	1
3	0	1	...	0	...	0
4	0	0	...	1	...	0
5	0	0	...	1	...	0

INDICE Bitmap sull'attributo $DT_i.b_i$

$DT_i.b_{i\,=}\ val_i$

RID	1	2	3	4	5	....
1	0	0	0	1	0	....
2	0	0	0	1	0	...
3	0	0	1	0	0	....
4	1	0	0	0	0	
5	0	0	0	0	1	
6	0	1	0	0	0	...

$\leftarrow DT_i$

Bitmap Join Index per ogni condizione $FT.a_i = DT_i.a_i$

FT

PASSO 2 A:

Per ogni BJI si caricano solo i vettori di bit corrispondenti ai suddetti RID.

RID	val1	val2	...	val i	...	valn
1	1	0	...	0	...	0
2	0	0	...	0	...	1
3	0	1	...	0	...	0
4	0	0	...	1	...	0
5	0	0	...	1	...	0

INDICE Bitmap sull'attributo $DT_i.b_i$

$DT_i.b_{i\ =}\ val_i$

Bitmap Join Index per ogni condizione $FT.a_i = DT_i.a_i$

RID	1	2	3	4	5	....
1	0	0	0	1	0	....
2	0	0	0	1	0	...
3	0	0	1	0	0	....
4	1	0	0	0	0	
5	0	0	0	0	1	
6	0	1	0	0	0	...

RID	val1	val2	...	val i	...	valn
1	1	0	...	0	...	0
2	0	0	...	0	...	1
3	0	1	...	0	...	0
4	0	0	...	1	...	0
5	0	0	...	1	...	0

INDICE Bitmap sull'attributo $DT_i.b_i$

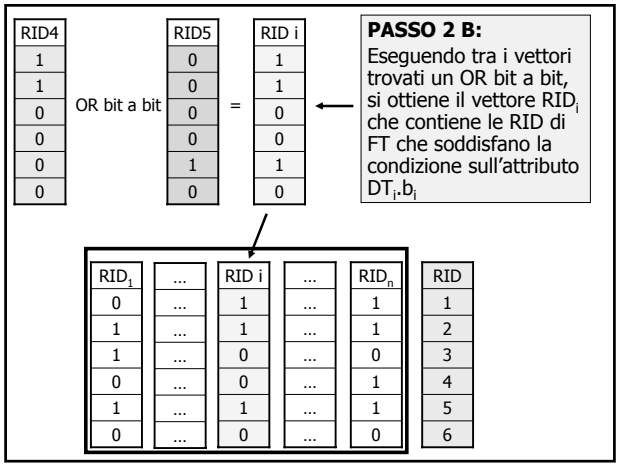
$DT_i.b_{i\ =}\ val_i$

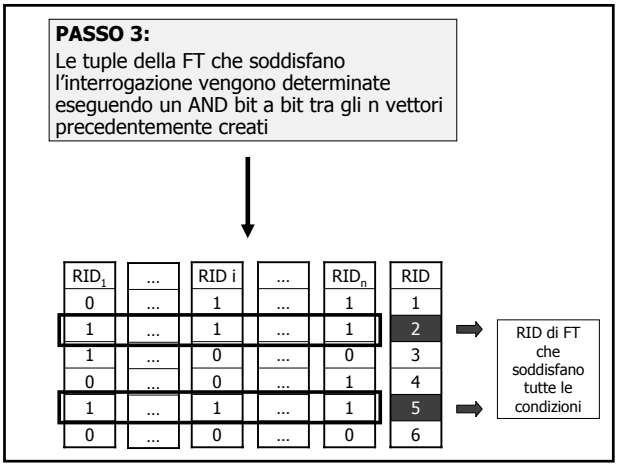
Bitmap Join Index per ogni condizione $FT.a_i = DT_i.a_i$

RID	1	2	3	4	5	....
1	0	0	0	1	0	....
2	0	0	0	1	0	...
3	0	0	1	0	0	....
4	1	0	0	0	0	
5	0	0	0	0	1	
6	0	1	0	0	0	...

RID4
1
1
0
0
0
0

RID5
0
0
0
0
1
0





Bitmapped Join Index

- Il numero di BJI è lineare nel numero degli attributi dimensionali
- Negli schemi a stella i BJI vengono creati tra le chiavi primarie delle DT e le corrispondenti chiavi esterne della FT.
- La dimensione di un BJI può essere elevata essendo dipendente dal numero di tuple presenti nelle tabelle R e S.

Considerazioni generali

La scelta degli indici

- Non si può prescindere dalla conoscenza delle caratteristiche del DBMS su cui si andrà ad operare.
  - FONDAMENTALITA' dell'OTTIMIZZATORE

La scelta degli indici

- OTTIMIZZATORE:
  - Modulo preposto alla valutazione ed al confronto dei possibili piani di esecuzione per le interrogazioni
- PIANO DI ESECUZIONE:
  - Indica la sequenza delle operazioni da svolgere per risolvere una interrogazione

La scelta degli indici

- Come nell'OLTP, anche nell'OLAP i possibili piani di esecuzione sono fortemente condizionati da:
  - Caratteristiche dell'ottimizzatore
  - Tipologie di indici
- A differenza dell'OLTP, ci sono maggiori vincoli (lo star schema non lascia molte chance: es. che senso ha fare un join tra due Dimension Table che si risolverebbe in un prodotto cartesiano?)

La scelta degli indici

- Comunque sia, il progettista non può disconoscere:
  - Tipi di indici presenti nel sistema
  - Quanti indici possono essere utilizzati contemporaneamente
  - Quali sono le "linee guida" secondo cui opera l'ottimizzatore
- Tenendo presente che siamo in un contesto OLAP ben diverso da quello OLTP

La scelta degli indici

- Esempio:
  - se il DBMS è in grado di usare più indici contemporaneamente
  → conviene creare un ampio insieme di indici sugli attributi dimensionali
  - se il DBMS fa ampio uso di indici sulle chiavi esterne della Fact Table
  → conviene creare indici sui singoli attributi della chiave primaria
  - se il DBMS usa indici su attributi su cui è espressa una condizione di selezione
  → conviene evitare di creare l'indice se ha scarsa selettività

La scelta degli indici

- Problema reale: conoscere in profondità le possibilità e le capacità del DBMS che si intende utilizzare.
- Praticamente (APPROCCIO EMPIRICO):
  - Se non si conoscono le tecniche e le performance medie del sistema utilizzato, si consiglia di procedere tramite un esame dei piani di esecuzione generati in risposta ad un insieme di interrogazioni di prova.
  - Poi si sperimenta il data mart col carico di lavoro effettivo, eventualmente apportando modifiche allo schema iniziale degli indici.

La scelta degli indici

- OTTIMIZZATORI BASATI SU REGOLE
  - Si tiene conto della struttura dei dati
  - Si tiene conto della struttura dell'interrogazione
  - Si tiene conto degli indici
  - NO a considerazioni statistiche sui dati

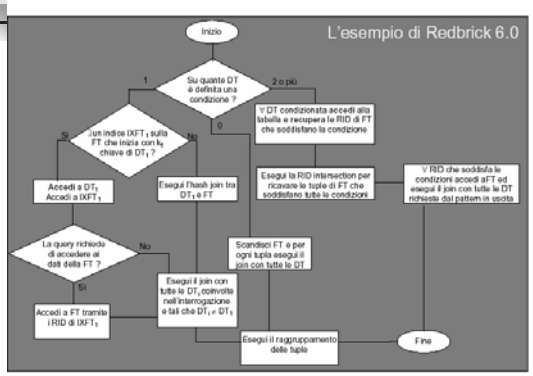
  → Sempre il solito piano indipendentemente dalla cardinalità delle tabelle e della distribuzione dei valori degli attributi

La scelta degli indici

- OTTIMIZZATORI BASATI SUI COSTI
  - Si tiene conto del costo stimato di esecuzione sulla base di STATISTICHE (memorizzate in appositi CATALOGHI) che considerano:
    - Cardinalità delle relazioni
    - Cardinalità degli attributi
    - Distribuzione dei valori degli attributi stessi

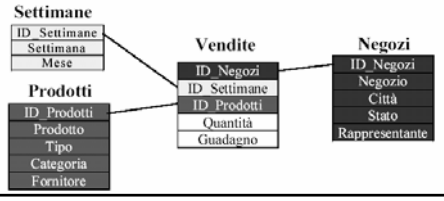
  → TEORICAMENTE si ottengono delle soluzioni più accurate

Si consiglia di creare indici su..



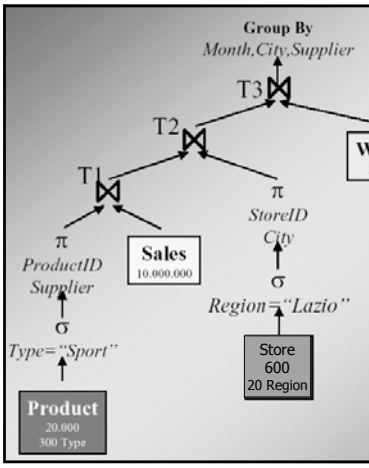
Esempio: *VENDITE*(*Settimane.Mese*, *Prodotto.Fornitore*, *Negozio.Città*;*Prodotto.Tipo*="Sport",*Negozio.Stato*="I").*Quantità*

SELECT Settimane.Mese, Prodotti.Fornitore, Negozi.Città,
SUM(vendite.Quantità)
FROM Vendite, Negozi, Settimane, Prodotti
WHERE Vendite.ID_Negozi = Negozi.ID_Negozi
AND Vendite.ID_Settimane = Settimane.ID_Settimane
AND Vendite.ID_Prodotto = Prodotti.ID_ Prodotti
AND Prodotti.Tipo = "Sport" **AND** Negozi.Stato = "I"
GROUP BY Settimane.Mese, Prodotti.Fornitore, Negozi.Città



Example: *SALES*(*Week.Month*,*Store.City*;
Product.Type="Sport",*Store.Region*="Lazio").*Quantity*

```
SELECT Week.Month,Product.Supplier,Store.City,SUM(Sales.Quantity)
FROM Sales, Store, Week, Product
WHERE Sales.StoreID = Store. StoreID
AND Sales.WeekID = Week.WeekID
AND Sales.ProductID = Product.ProductID
AND Product.Type = "Sport" AND Store.Region = "Lazio"
GROUP BY Week.Month, Product.Supplier,Store.City
```



- E' preferibile anticipare le operazioni di selezione e proiezione
- L'ordine di esecuzione dei join dipende dalla selettività dei predicati relativi alle diverse DT
- Non è conveniente eseguire join tra DT

Indice	Scansione	Indice	Spazio
B⁺-Tree I. su Prodotti.Tipo	704	66	22
B⁺-Tree I. su Negozi.Regione	22	20	4

Piano di accesso (costo espresso in pagine)

Selezione su prodotti (B⁺-Tree):	66
Join Prodotti-Vendite (Nested-Loop):	273,438
Selezione su Negozi (B⁺-Tree):	20
Join con Negozi (Nested-Loop):	912
Join con settimane (Nested-Loop):	1,845,928
Totale (Pagine)	2,120,364
Assumendo throughput = 12 Mbyte/Sec dimensione di pagina = 1Kbyte	173 (sec)

Indice	Scansione	Indice	Spazio
Bitmap I. su Prodotti.Tipo	704	799	735
Bitmap I. su Negozi. Stato	22	18	2
B⁺-Tree I. su Prodotti.Tipo	704	66	22
B⁺-Tree I. su Negozi.Stato	22	20	4
Join I. su Prodotti-Vendite	65,975,950	10,834	39,295
Join Bitmap I.su Prodotti-Vendite (Tipo)	65,975,950	398,814	367,432

Piano di accesso (costo espresso in pagine)

Selezione su prodotti (B⁺-Tree):	66
Join Prodotti-Vendite (Join I.)	10,834
Selezione su Negozi (Bitmap)	18
Join con Negozi (Sort-Merge):	12,786
Join con settimane (Sort-Merge):	512
Totale (Pagine)	24,216
Assumendo throughput = 12 Mbyte/Sec dimensione di pagina = 1Kbyte	2 sec

La scelta degli indici

1. Riferirsi ai criteri di ottimizzazione del DBMS di appoggio per scegliere gli indici più idonei tenendo presenti i vincoli di spazio
2. Effettuare Test di valutazione
3. Procedere per raffinamenti successivi apportando modifiche allo schema di indici iniziale
4. NON POSSIAMO PRESCINDERE dal rispettare il vincolo dello spazio.

La scelta degli indici

- Il problema della scelta degli indici può essere posto in modo molto simile a quello della materializzazione delle viste e può essere affrontato in due fasi successive:
  - Individuazione dell'insieme di indici utili
  - Selezione del miglior sottoinsieme

La scelta degli indici

- Individuazione dell'insieme di indici utili
  - L'utilità di un indice dipende dalle caratteristiche del DBMS e del suo ottimizzatore

La scelta degli indici

- Selezione del miglior sottoinsieme
  - Considerando che abbiamo vincoli di spazio, solo un sottoinsieme di indici potrà essere effettivamente creato

Indicizzazione delle DT

- Poiché gli attributi delle DT vengono usati principalmente per filtrare le tuple da recuperare nella fact table

→ risulta utile che vengano indicizzati (liste di RID o indici BITMAP)

→ la scelta dipende dalle caratteristiche dell'attributo: maggiore è la cardinalità dell'attributo, maggiore è l'utilità delle liste di RID

Indicizzazione delle DT

- La scelta degli attributi da indicizzare deve tener conto del workload (prevedibile da progetto e/o ricavabile dalla frequenza d'uso che se ne fa che può essere analizzata tramite il file di log)

MAGGIORE è il numero di interrogazioni che esprimono condizioni su un attributo → MAGGIORE è la necessità di indicizzare quell'attributo.

Indicizzazione delle DT

- Solitamente sulla chiave primaria delle DT, viene costruito un indice di tipo B^+-Tree:
  - visto il suo frequente utilizzo per l'aggancio con la Fact Table
  - Per agevolare l'aggiornamento periodico dei dati per il controllo dell'integrità

Indicizzazione delle DT

- Si ricordi che nel caso di uno schema tipo snowflake, è necessario indicizzare non solo le chiavi primarie delle DT ma anche le corrispondenti chiavi esterne per favorire le interrogazioni che coinvolgono anche le DT secondarie.

Indicizzazione delle DT

- Un possibile criterio che dipende dalle caratteristiche dell'attributo da indicizzare

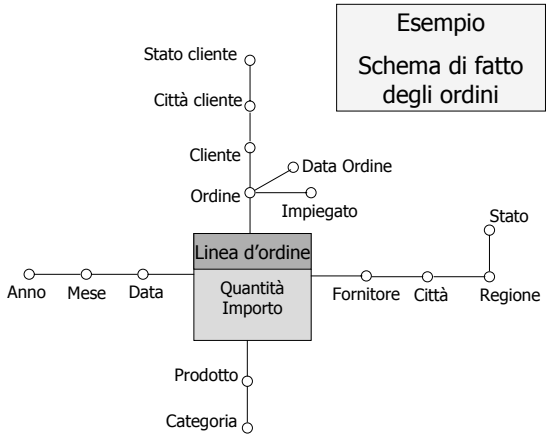
  MAGGIORE è la cardinalità del dominio dell'attributo

➔ MAGGIORE è l'utilità degli indici a liste di RID (es. B^+-Tree)

Indicizzazione delle DT

Schema per effettuare la scelta del tipo di indice da usare in base alle caratteristiche dell'attributo e delle selezioni che su di esso insistono

Numero dei dati da recuperare		Valori distinti	
		MOLTI	POCHI
Selettività interrogazioni	ALTA	B^+-Tree	B^+-Tree Bitmap
	BASSA	B^+-Tree Bitmap evoluti	Bitmap



Stima della dimensione di alcuni degli indici costruibili sullo schema degli ordini.

Attributo	Valori distinti	Tuple	Tipo indice	Dimensione (MB)
Stato cliente	25	1500000	B^+-Tree	8,6
Stato cliente	25	1500000	Bitmap	4,5
Impiegato	1000	1500000	B^+-Tree	8,6
Impiegato	1000	1500000	Bitmap	179,8

Si noti come la dimensione degli indici bitmap cresca in funzione del numero di valori distinti dell'attributo e risulti proibitiva per l'indice sull'attributo impiegato.

Indicizzazione della FACT TABLE

- Le esigenze di indicizzazione riguardano principalmente le operazioni di join

→coinvolgimento degli attributi che compongono la chiave della FT

- L'indice sulla chiave primaria (essendo composta di più attributi) non è sempre utilizzabile nell'esecuzione dei join

Indicizzazione della FACT TABLE

- IMPORTANZA DELL'ORDINE DELLE CHIAVI NELL'INDICE:
  - un join tra una DT ed una FT può usare proficuamente l'indice su FT solo se la chiave esterna di join con la DT è la prima a comparire nell'indice

    → → →

    nella prima posizione di un indice composto dev'essere collocato l'attributo maggiormente utilizzato nelle selezioni.

Indicizzazione della FACT TABLE

- E' consigliabile creare più indici di join tra coppie di tabelle piuttosto che un unico indice di join multi-tabella (star index) → maggiore flessibilità d'uso
- Quando è possibile, conviene utilizzare i Bitmapped Join Index
- In alternativa, qualora il DBMS non metta a disposizione strutture ad indice specializzate per le operazioni di join o quando non possiamo disporre di spazio a sufficienza, possono essere costruiti più indici di tipo B^+-Tree per indicizzare le singole componenti della chiave della FT.

Indicizzazione della FACT TABLE

- Talvolta può essere necessario fare indicizzazioni su misure invece che su attributi: data la natura numerica delle stesse si può utilizzare un tipo specifico di indice (bit sliced: dimensioni ridotte ed elevata flessibilità nell'esecuzione delle interrogazioni)
- Dato l'alto numero di tuple della FT, ogni indice costruito su di esse avrà dimensioni ragguardevoli: → occorre valutare bene la convenienza di definire indici sulle misure.

Indicizzazione della FACT TABLE

Attributo	Valori distinti	Tuple	Tipo indice	Dimensione (MB)
Stato cliente	25	1500000	B^+-Tree	8,6
Stato cliente	25	1500000	Bitmap	4,5
Impiegato	1000	1500000	B^+-Tree	8,6
Impiegato	1000	1500000	Bitmap	179,8
Chiave FT	6000000	6000000	Star Index	126,7

Allocazione fisica

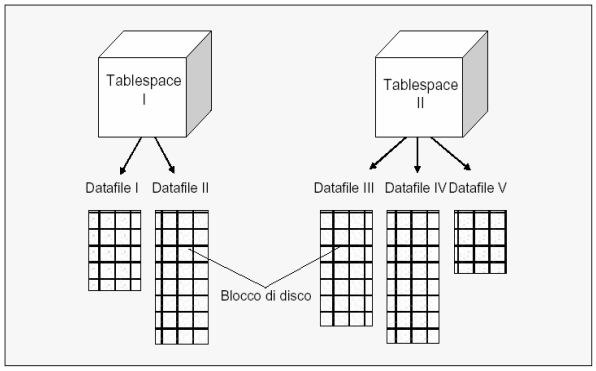
Allocazione fisica dei dati

- La progettazione fisica di un data mart non si conclude con la definizione del piano degli indici, ma contempla un insieme di parametri che nel loro complesso determinano lo schema fisico.

Allocazione fisica dei dati

- La progettazione fisica di un data mart non si conclude con la definizione del piano degli indici, ma contempla un insieme di parametri che nel loro complesso determinano lo schema fisico.
- Lo spazio su disco occupato da un database è normalmente suddiviso in:
  1. **Tablespace:** suddivisione logica dei dati in unità contenenti tipologie uniformi di informazioni.
  2. **Datafile:** file contenente parte delle informazioni di un tablespace.
  3. **Blocco dati**: unità di informazione letta o scritta dal DBMS.
- Organizzazione, dimensione, e disposizione di questi elementi può incidere notevolmente sulle prestazioni

Allocazione fisica dei dati



I blocchi di disco

- Ogni DBMS permette di specificare la dimensione dei blocchi di disco, ossia il numero di byte letti/scritti in modo atomico ad ogni accesso.
- La dimensione di un blocco varia tra 0,5 e 256 KB.
  - ✓ Blocchi piccoli sono più adatti per tabelle con tuple di dimensione ridotta e interrogazioni che coinvolgono poche tuple (sistemi OLTP)
  - ✓ Blocchi grandi sono più adatti per tabelle con tuple di dimensione elevata e interrogazioni che coinvolgono molte tuple (sistemi OLAP)

Per i sistemi operazionali si utilizza normalmente una dimensione di blocco di disco compresa tra i 2 e gli 8 Kbyte, per i sistemi di data warehousing si consiglia di utilizzare blocchi compresi tra i 32 e i 64 Kbyte.

I tablespace

- Il livello di astrazione dei tablespace permette di razionalizzare l'organizzazione dei dati e funge da punto di partenza per il miglioramento delle prestazioni e della tolleranza ai guasti ottenibile tramite una coerente allocazione dei relativi datafile.
- I fattori da prendere in considerazione sono:
  - ✓ L'omogeneità dei dati
  - ✓ La quantità dei dati
  - ✓ Le tipologie di accesso

Tablespace	*Descrizione*
SYSTEM	Tablespace per il dizionario dati
DATA1..., *n*	Tablespace per i dati
AGG1..., *n*	Tablespace per i dati aggregati
INDEX1..., *m*	Tablespace per gli indici
INDEX_AGG1..., *m*	Tablespace per gli indici sui dati aggregati
RBS	Tablespace di rollback standard
RBS_LOAD	Tablespace di rollback utilizzato durante il caricamento
TEMP	Tablespace per i dati temporanei
TEMP_LOAD	Tablespace per i dati temporanei necessari al caricamento
TOOLS1..., *h*	Tablespace per stored procedure e trigger

I datafile

- Un'intelligente allocazione dei datafile sui dischi permette di:
  - ✓ Migliorare le prestazioni grazie all'accesso parallelo a più dischi
  - ✓ Aumentare la resistenza ai guasti

Verso il parallelismo……..

Verso il parallelismo……

- Vista la grande quantità di dati da memorizzare, è pensabile di avere a disposizione più unità disco.
- Un'efficiente allocazione dei datafile sui dischi permette di:
  - Migliorare le prestazioni possibilmente grazie all'accesso parallelo ai dischi durante le interrogazioni
  - Aumentare la resistenza ai guasti dato che l'eventuale guasto di un disco coinvolgerà solo una parte dei dati (!?!?!)

Parallelismo

- Le operazioni di I/O determinano in larga parte il costo delle interrogazioni poiché il tempo di accesso al disco è di diversi ordini di grandezza superiore al costo di esecuzione di un'operazione della CPU
- Il tempo di I/O può essere ridotto utilizzando più dischi in parallelo.

Parallelismo

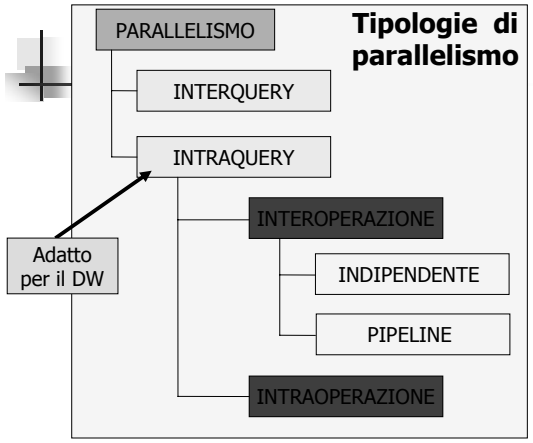
The Database Problem

- Large volume of data — use disk and large main memory
- I/O bottleneck (or memory access bottleneck)
  - Speed(disk) << speed(RAM) << speed(microprocessor)
- Predictions
  - (Micro-) processor speed growth : 50 % per year
  - DRAM capacity growth : 4× every three years
  - Disk throughput : 2× in the last ten years
- Conclusion : the I/O bottleneck worsens

Parallelismo

- L'accesso in parallelo è possibile in presenza di più dischi fissi purché ognuno abbinato ad un proprio controller.
- Il parallelismo non può essere comunque effettuato senza criterio:
  - Lo sfruttamento ottimale del parallelismo (in presenza di più dischi) è ottenibile solo modellando in modo adeguato lo schema logico-fisico del database.

Tipologie di parallelismo



Parallelismo

- INTERQUERY:

Esecuzione parallela di più query al fine di incrementare il numero di transazioni per unità di tempo (throughput)

MA

nei sistemi di DW, il fattore critico non è dato dal numero di accessi contemporanei al sistema bensì dalla complessità delle singole interrogazioni

➔ Non utilizzato nel contesto dei DW.

Parallelismo

- INTRAQUERY:

Si ottimizza lo schema al fine di massimizzare la velocità di ogni singola interrogazione.

Si memorizzano su dischi distinti, le porzioni di dati necessarie ad una singola interrogazione (declustering) così da parallelizzare l'accesso ai dati ed eventualmente consentire l'esecuzione da parte di più processori.

Parallelismo

- INTRAQUERY: può essere attuato
- a livello di tabella, memorizzando in dischi separati la FACT TABLE e le relative DT in modo da favorire la lettura parallela dei dati durante le join.
- a livello di frammento, partizionando la FACT TABLE in più frammenti allocati su dischi diversi e quindi leggibili in parallelo.

Parallelismo

- INTRAQUERY (i meccanismi):
  - I meccanismi per sfruttare il parallelismo Intraquery sono:
    - *Per l'interoperazione*
      - Parallelismo Indipendente *(indipendent parallelism)*
      - Parallelismo di Iteratori *(pipelined parallelism)*
    - *Per l'intraoperazione*
      - Parallelismo di partizionamento *(partitioned parallelism)*
    - Questi tre tipi di parallelismo possono anche coesistere

PARALLELISMO INTRAQUERY

- Inter-operation
  - ⇒ p operations of the same query in parallel

- **INTEROPERAZIONE**:
- Esecuzione parallela dei diversi operatori relazionali che compongono l'interrogazione

Parallelismo Interoperazione (1)

- Parallelismo Indipendente *(indipendent parallelism)*:
  - Si eseguono in parallelo gli operatori indipendenti:
  - Es.

$R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4$

si possono eseguire parallelamente

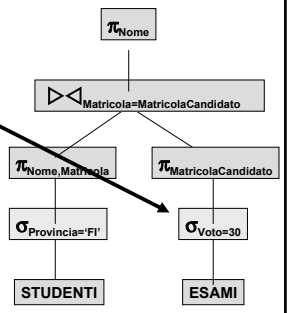
$(R_1 \bowtie R_2)$ e $(R_3 \bowtie R_4)$

e dopo si fa la giunzione tra i risultati.

Parallelismo Interoperazione (1)

Indipendent Parallelism

- Non c'è dipendenza tra gli operatori che possono eseguiti in parallelo.
- Non ci sono interferenze tra processori.

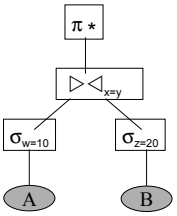


Parallelismo Interoperazione (2)

- Parallelismo di Iteratori *(pipelined parallelism)*:

```
Select *
from   A, B
where A.x = B.y  and  A.w=10  and  B.z=20
```

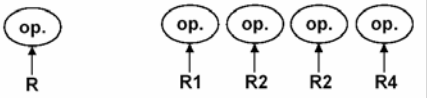
Le ennuple soddisfacenti le condizioni vengono determinate in parallello ed inviate a mano a mano al processo di livello superiore che ne fa la join.
Il vantaggio è che il risultato intermedio non viene materializzato, risparmiando memoria ed accessi disco e l'inizio dell'attività dell'operatore successivo viene anticipata.



PARALLELISMO INTRAQUERY

- **INTRAOPERAZIONE**:
  - utilizzo di algoritmi paralleli per l'esecuzione del singolo operatore relazionale (sono stati studiati algoritmi per realizzare tutti gli operatori relazionali e per l'ordinamento)

- Intra-operation
  - the same operation in parallel on different data partitions



INTRAOPERAZIONE

- Si basa sulla decomposizione di un operatore in un insieme di sub-operatori indipendenti, detti "istanze di operatore"
- La decomposizione può essere fatta usando il partizionamento statico e/o dinamico.

Parallelismo Intraoperazione

- Parallelismo di partizionamento *(partitioned parallelism)*:
  - Si possono valutare in parallelo i singoli operatori di un piano di accesso partizionando i dati, lavorando su ogni partizione in parallelo e combinando alla fine i risultati.
  - All'uopo sono stati studiati tutta una serie di algoritmi paralleli per tutti gli operatori relazionali nonché per l'ordinamento

INTRAOPERAZIONE

- L'operazione di Selezione può essere decomposta in più operazioni di Selezione, ognuna operante su una differente partizione dei dati.
- Se la relazione è partizionata sull'attributo di selezione, le proprietà di partizionamento possono essere usate per "guidare" la query sulle istanze giuste

Resistenza ai guasti Fault tolerance

- Capacità del sistema di operare anche a fronte di guasti di una parte dei suoi componenti (es. dischi)
- La necessità deriva dall'esigenza di mantenere disponibili informazioni rilevanti per il controllo aziendale e storiche difficilmente ricostruibili (periodo molto più ampio di quello coperto dai sistemi operazionali)

Resistenza ai guasti Fault tolerance

- Buona norma: mantenere una copia fuori linea (es. su nastro magnetico o disco ottico) almeno delle viste primarie → le secondarie e gli indici si possono ricostruire da queste (non particolari problemi se non quelli di tempo!)
- La presenza di più dischi (oltre al vantaggio delle operazioni di I/O parallele) riduce la percentuale di dati persi in caso di guasto.
- Anche se, statisticamente, all'aumentare dei dischi, si riduce il tempo medio tra due guasti.

Resistenza ai guasti
Fault tolerance

- Per cui:
  - Si vogliono usare più dischi per permettere l'accesso parallelo ai dati
  - Si rende necessario introdurre ridondanze nei dati al fine di ridurre il rischio di perdita di informazioni
  - DATA STRIPING
  - DATA REDUNDANCY

Resistenza ai guasti
Fault tolerance

- DATA STRIPING:
  - Tecniche per la distribuzione di un insieme di dati su più dischi al fine di permetterne la lettura (scrittura) parallela.
- DATA REDUNDANCY:
  - Tecniche che, mediante memorizzazione ridondante di dati, riducono la probabilità di perdita dei dati.
- CONCETTI ALLA BASE DEI DISCHI RAID (Redundancy Array of Inexspensive Disks)

Resistenza ai guasti
Fault tolerance

- RAID 1: mirrored disk
  dati in duplice copia su dischi distinti
  → perdita di un dato si verifica solo se entrambi i dischi si guastano contemporaneamente

Resistenza ai guasti
Fault tolerance

- RAID 0+1: stripe of mirrors
  dati distribuiti su insiemi di dischi che vengono replicati, realizzando così sia il mirroring che il data striping

Resistenza ai guasti
Fault tolerance

- RAID 5: block-interleaved distributed parity
  dati e parità distribuiti su più dischi; in tal caso non si ha mirroring, perché non viene memorizzata una copia vera e propria dei dati ma si utilizzano tecniche di correzione dell'errore (bit di parità per ogni unità di dati)

Resistenza ai guasti
Fault tolerance

- Le prime due soluzioni offrono un'elevata affidabilità ma presentano anche il costo più elevato fra le architetture RAID.
- La terza soluzione presenta ottime prestazioni in fase di lettura e scrittura di grandi quantità di dati.
- Comunque sia, la capacità dell'I/O parallelo dipende dal numero di controller disco utilizzati

→ Il parallelismo totale si ottiene solo in presenza di un controller distinto per ogni unità a disco.

DATA WAREHOUSE

- Riferimento bibliografico:

  "DATA WAREHOUSE", *Golfarelli e Rizzi*, Mc Graw-Hill, 2005 II edizione

FINE