Esperimenti su DB2

Tecnologia delle Basi di Dati

Anno accademico 2003-2004

Università di Roma 3

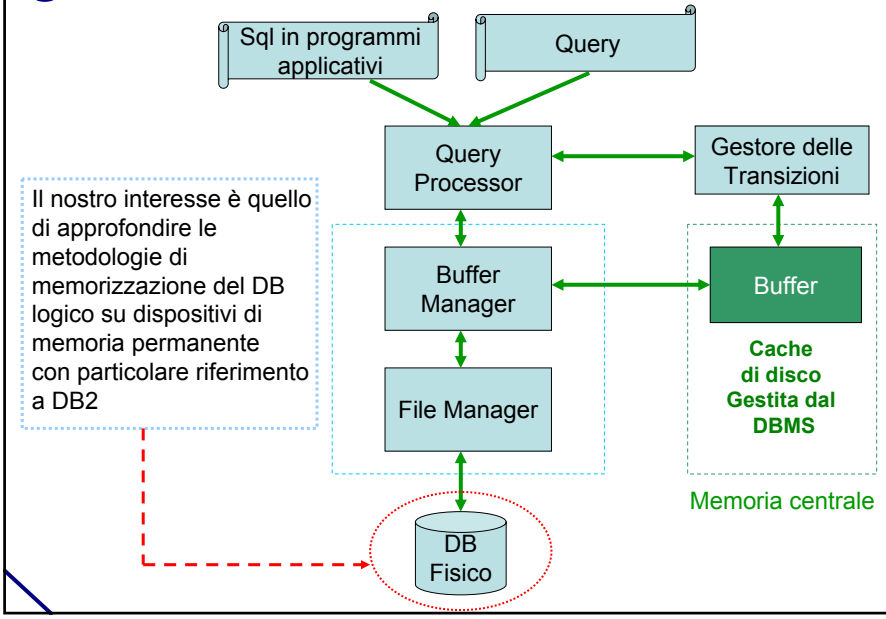
Realizzato da:

Daniele Nassuato

Alessandro Di Francesco

DB2. Information Management Software

Architettura di un DBMS



Collo di Bottiglia : Operazioni di I/O

- Poiché spesso le operazioni di I/O costituiscono il collo di bottiglia del sistema, si rende necessario ottimizzare l'implementazione fisica del DB, attraverso:

- Organizzazione efficiente delle tuple su disco
- Strutture di accesso efficienti
- Gestione efficiente dei buffer in memoria
- Strategie di esecuzione efficienti per le query

Prestazioni di una memoria:

$$\text{velocità di trasferimento} = \text{latenza} + \frac{\text{dimensione dati da trasferire}}{\text{tempo di accesso}}$$

• dato un indirizzo di memoria, le prestazioni si misurano in termini di tempo di accesso, determinato dalla somma:

latenza : tempo necessario per accedere al primo byte

tempo di trasferimento : tempo necessario per muovere i dati

Il DB Fisico

- A livello fisico un DB consiste di un insieme di file, ognuno dei quali viene visto come una collezione di pagine, di dimensione fissa (es: 4 KB)
- Ogni pagina memorizza più record (corrispondenti alle tuple logiche)
- A sua volta un record consiste di più campi, di lunghezza fissa e/o variabile, che rappresentano gli attributi
- I "file" del DBMS che qui consideriamo non corrispondono necessariamente a quelli del file system del sistema operativo

Casi limite:
- Ogni relazione del DB è memorizzata in un proprio file
- Tutto il DB è memorizzato in un singolo file

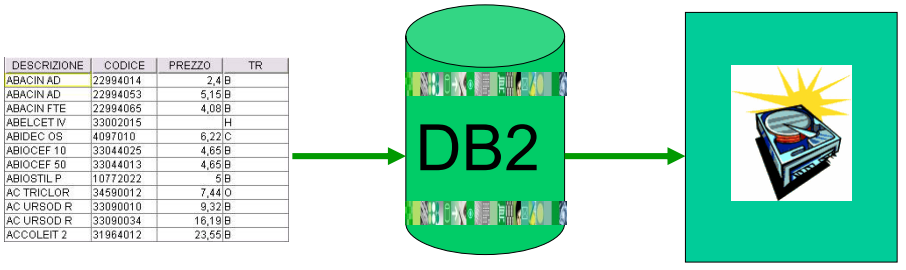
In pratica ogni DBMS a livello fisico adotta soluzioni specifiche più articolate e flessibili

Il modello di memorizzazione di DB2 : schema generale

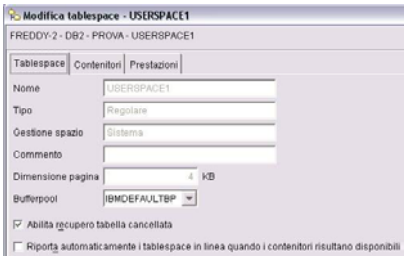
Come Db2 organizza i dati dei nostri DB su disco?

Introduciamo i concetti di:

- Tablespace
- Container
- Extent



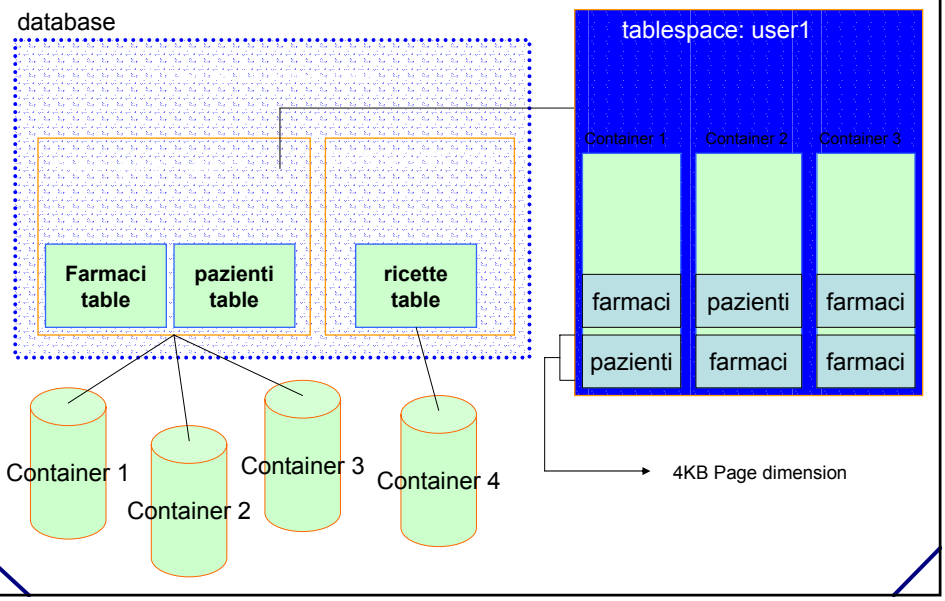
Il modello di memorizzazione di DB2 : panoramica



- Db2 organizza lo spazio fisico in Tablespace ognuno dei quali è una collezione di Container
- I Container possono essere memorizzati anche su supporti diversi

- Ogni Container è a sua volta suddiviso in Extent : questi rappresentano l'unità minima di l'allocazione su disco e sono costituiti da insiemi contigui di pagine da 4K (valore di default)
- La dimensione di un Extent dipende dal tablespace a cui è associato e viene fissato al momento della creazione
- Ogni relazione è memorizzata all'interno di un singolo tablespace, e un tablespace può contenere più relazioni
- Un Extent può contenere dati di una singola relazione

Il modello di memorizzazione di DB2 : Panoramica



Il modello di memorizzazione di DB2 : Table space

Un Table Space è una struttura di memorizzazione contenente :

- Tabelle
- Indici
- Large objects and long data.

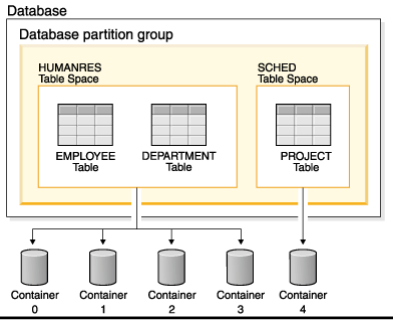
Risiedono nel database partition groups

Ci permettono di assegnare la posizione del database e dei Table Data direttamente nei Container

Ci permettono di avere una flessibilità di configurazione e di migliorare le prestazioni
È il database manager che si occupa di bilanciare i dati nei Container

(Un Container può essere un nome di directory , un device name, o un file name.)

- Tutti i contenitori sono utilizzati per memorizzare i dati.
- Il numero delle pagine che il database manager scrive in un container prima di utilizzare un differente container prende il nome di : Extent size
- Il database manager non comincia a scrivere sempre dal primo container



Il modello di memorizzazione di DB2 : Table space (2)

Un Database deve contenere Almeno Tre table Space :

- *Catalog table space*: contiene tutte le catalog tables di sistema del Db
  questo table space prende il nome di SYSCATSPACE
  non può essere eliminato
  (IBMCATGROUP is the default database partition group for this table space)

- Uno o più *user table spaces*: che contiene tutte le tabelle definite dall'utente
  di default ne viene creato uno con il nome di USERSPACE1 (IBMDEFAULTGROUP is the default database partition group for this table space)

- Uno o più Table Space temporanei : contiene le tabelle temporanee
  i Temporary table spaces possono essere *system temporary table spaces* or *user temporary table spaces*. Un Db deve avere minimo un temporary table space

NB. Ogni volta che si crea una tabella va specificato il table space a cui appartiene, or the results may not be what you intend.

Il modello di memorizzazione di DB2 : Table space (3)

- Il table's page size è determinato dalla grandezza delle righe(della tabella) e dal numero di colonne Il massimo spazio allocabile per una riga all'interno di una pagina è determinato al momento della creazione del table space.
  I possibili valori per page size sono:

  4 KB (the default) 8 KB 16 KB and 32 KB.

- Se vengono eseguite delle query in tabelle all'interno di table space che è stato definito con un page size più grande di 4 Kb (per esempio, ORDER BY on 1012 columns) alcune di queste falliranno : perché?

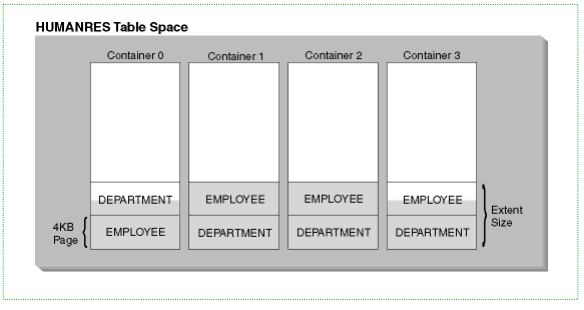
  Questo accadrà se non state definite dei temporary table spaces con un page size grande abbastanza.

  Importante! Creare temporary table spaces con page size più grande rispetto al page size maggiore all'interno degli user table space.(SMS)

Il modello di memorizzazione di DB2 : Container

- Tutti i Table space sono costituiti da un numero n di container (definiti dall'utente)
  un *container* descrive la locazione dove vengono memorizzati i dati
  una subdirectory in un file system è un esempio di container.
- Quando i dati sono letti dai containers di un table space, sono caricati in un area di memoria chiamata Buffer pool
  un *buffer pool* è associato ad uno specifico table space:

" thereby allowing control over which data will share the same memory areas for data buffering



Il modello di memorizzazione di DB2 : Sms e DMS

Esistono due tipologie di table space che si possono utilizzare contemporaneamente all'interno di un singolo DB:

- System managed space, in which the operating system's file manager controls the storage space.

- Database managed space, in which the database manager controls the storage space.

Il modello di memorizzazione di DB2 : Sms

In un SMS (System Managed Space) table space, è il manager del file system del sistema operativo che alloca e gestisce lo spazio in cui sono memorizzate le tabelle

- Il modello di memorizzazione tipicamente consiste in molti file (rappresentanti gli oggetti delle tabelle) salvati nello spazio del file system.

- L'utente decide la locazione dei file
- DB2(R) controlla i loro nomi
- il file system è responsabile della loro gestione

Ogni tabella ha almeno un SMS physical file associato

- in un SMS table space, un file è esteso una pagina alla volta.
  Se occorre migliorare le prestazioni, si può considerare di abilitare la funzione di multipage file allocation.
  Questo permette di estendere i file a più di una pagina alla volta

- Per ragioni di prestazioni se si memorizzano delle multidimensional (MDC) tables all'interno di un SMS table space è indispoensabile abilitare il multipage file allocation.

- (Run **db2empfa** to enable multipage file allocation)

Il modello di memorizzazione di DB2 : Sms (2)

Bisogna considerare due fattori chiave quando si progetta la creazione di un SMS:

- Il numero di container per il table space:
  È di estrema importanza identificare il numero di contenitori che si vogliono usare in quanto non sarà possibile aggiungere o eliminare container una volta creato il table space.
- la massima dimensione del table space che può essere stimata :

numero di containers * (dimensione massima del file system supportata dal SO)

- Per scegliere l'appropriato valore del numero di containers e di the extent size per il table space occorre comprendere:

- Le limitazioni che il SO impone alla dimensione logica del file system.
  per esempio, alcuni SO hanno 2 GB di limite. quindi, se si vogliono 64 GB table object, occorrono minimo 32 containers in questo tipo sistema.
  Quando si crea un table space si può specificare che i containers risiedano in differenti file systems

Il modello di memorizzazione di DB2 : DMS

- In un DMS (Database Managed Space) table space,
  Il Database Manager controlla lo spazio di memorizzazione

Il modello di memorizzazione consiste in un limitato numero di devices o files il cui spazio è gestito da DB2 :

The database administrator decide quali devices e file usare
DB2(R) gestisce lo spazio in questi devices e files

Il table space è implementazione di un file system speciale costruito allo scopo di venire incontro nel migliore dei modi ai bisogni del Db manager.

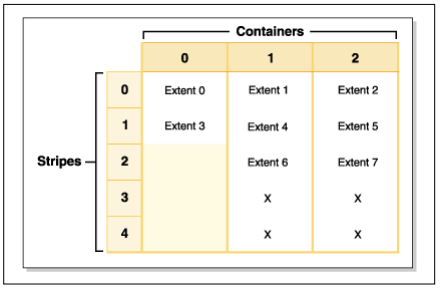
Un DMS table space contiene user defined tables and data che possono essere definiti come:

- A *regular* table space per memorizzare ogni table data e opzionalmente index data
- A *large* table space per memorizzare long field , LOB data e index data.

Il modello di memorizzazione di DB2 : DMS (2)

Quando si progetta un DMS table spaces e i suoi contenitori containers occorre considerare:

- Il database manager usa una rigatura (striping) per assicurarsi una distribuzione dei dati attraverso tutti i containers



- Al contrario dei SMS, nei DMS i containers non devono avere la stessa dimensione anche se sconsigliato poiché: risultano avere una rigatura differente abbassando le prestazioni.

se ogni container è full, DMS table space utilizza lo spazio libero disponibile di un altro container.

- Quando si usano device containers, il device deve possedere spazio sufficiente per la definizione di container, ogni device può avere soltanto un container definito su di esso, per evitare lo spazio sprecato occorre che la dimensione del device e del container sia la stessa (se per esempio, il device è allocato con 5 000 pages, e il container è definito per allocare 3000, 2 000 pages sul device non saranno utilizzabili

Il modello di memorizzazione di DB2 : DMS (3)

- di default, un extent in ogni container è riservato per l'overhead
- Per determinare la dimensione appropriata per un container si usa :

extent_size * (n + 1)

- The minimum size of a DMS table space is five extents
- The maximum size of regular table spaces is 64 GB for 4 KB pages; 128 GB for 8 KB pages; 256 GB for 16 KB pages; and 512 GB for 32 KB pages. The maximum size of large table spaces is 2 TB.

Il modello di memorizzazione di DB2 : Sms Vs DMS

Vantaggi in un SMS Table Space:

- Lo spazio non è allocato dal sistema prima che questo sia richiesto
- Creare un table space richiede minore lavoro iniziale perché non c'è bisogno di predefinire i containers

Vantaggi in un DMS Table Space:

- Lo spazio di un table space può essere incrementato aggiungendo o estendendo container (using the ALTER TABLESPACE statement)
  I dati esistenti possono essere automaticamente ri-bilanciati nel nuovo insieme di container per ottenere un'efficienza ottimale di I/O
- Una tabella può essere divisa in più table space a seconda dei dati che contiene :
  - Long field and LOB data
  - Indexes
  - Regular table data

Contiunua…….

Il modello di memorizzazione di DB2 : Sms Vs DMS (2)

- The location of the data on the disk can be controlled, if this is allowed by the operating system.
- If all table data is in a single table space, a table space can be dropped and redefined with less overhead than dropping and redefining a table.
- In general, a well-tuned set of DMS table spaces will outperform SMS table spaces.

In general, small personal databases are easiest to manage with SMS table spaces. On the other hand, for large, growing databases you will probably only want to use SMS table spaces for the temporary table spaces and catalog table space, and separate DMS table spaces, with multiple containers, for each table. In addition, you will probably want to store long field data and indexes on their own table spaces. If you choose to use DMS table spaces with device containers, you must be willing to tune and administer your environment.

Il modello di memorizzazione di DB2 : Sms Vs DMS (3)

Perché non usare sempre il file system?

Le prestazioni di un DBMS dipendono fortemente da come i dati sono organizzati su disco Intuitivamente, l'allocazione dei dati dovrebbe mirare a ridurre i tempi di accesso ai dati, e per far questo bisogna sapere come (logicamente) i dati dovranno essere elaborati e quali sono le relazioni (logiche) trai dati Tutte queste informazioni non possono essere note al file system

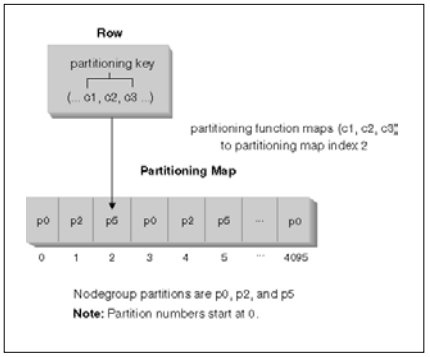
Esempi:

Se due relazioni contengono dati tra loro correlati(mediante join) può essere una buona idea memorizzarle in cilindri vicini, in modo da ridurrei tempi di seek

Il modello di memorizzazione di DB2 : Partition Map

La *partitioning map*, associata ad ogni database partition group, è usata dal database manager per determinare in quale partizione una data riga di dati deve essere memorizzata

Il partitioning map index prodotto dalla partitioning function per ogni riga in una tabella è usato come un indice nella partitioning map per determinare in quale partizione una data riga di dati deve essere memorizzata



http://publib.boulder.ibm.com/infocenter/db2help/topic/com.ibm.db2.udb.doc/admin/c0004126.htm

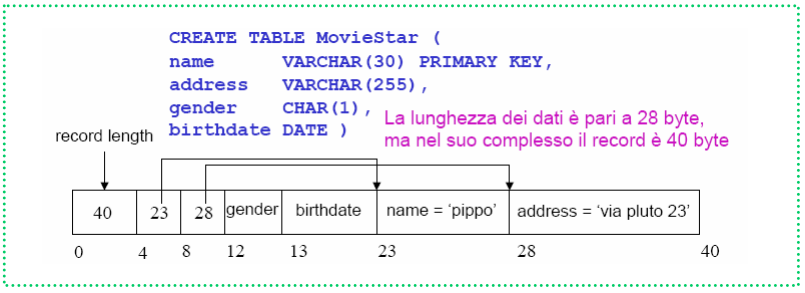
Il modello di memorizzazione di DB2 : rappresentazione dei valori

- Per ogni tipo di dati di SQL viene definito un formato di rappresentazione, ad es.:
  - Stringhe a lunghezza fissa: CHAR(n)
    Si usano n byte,eventualmente usando un carattere speciale per valori lunghi meno di n
    Esempio: se A è CHAR(5), 'cat' viene memorizzato come cat⊥⊥
  - Stringhe a lunghezza variabile: VARCHAR(n)
    Si allocano m+p byte, con m (≤ n) byte usati per gli m caratteri effettivamente presenti e p byte per memorizzare il valore di m (per n ≤ 254 p = 1)
    Esempio: se A è VARCHAR(10), 'cat' viene memorizzato in 4 byte come 3cat
  - DATE e TIME sono normalmente rappresentati con stringhe di lunghezza fissa
    DATE: 10caratteriYYYY-MM-DD; TIME: 8caratteriHH:MM:SS
  - Tipi enumerati: si usa una codifica intera
    Esempio: week = {SUN, MON, TUE, …, SAT} richiede un byte per valore
    SUN: 00000001, MON: 00000010, TUE: 00000011, …

Note that LOBs and LONG VARCHARs are not buffered by DB2's buffer pool

Il modello di memorizzazione di DB2 : Record Variabili

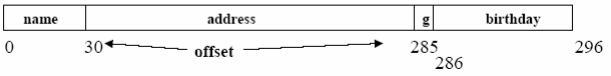
- Nel caso di record a lunghezza variabile si hanno diverse alternative,che devono considerare anche i problemi legati agli aggiornamenti che modificano la lunghezza dei campi(e quindi dei record)
- Una soluzione consolidata consiste nel memorizzare prima tutti i campi a lunghezza fissa, e quindi tutti quelli a lunghezza variabile; per ogni campo a lunghezza variabile si ha un "prefix pointer"che riporta l'indirizzo del primo byte del campo



Il modello di memorizzazione di DB2 : Record fissi

- Per ogni tipo di record nel DB deve essere definito uno schema (fisico) che permetta di interpretare correttamente il significato dei byte che costituiscono il record
- La situazione più semplice si ha evidentemente quando tutti i record hanno lunghezza fissa, in quanto,oltre alle informazioni logiche, è sufficiente specificare l'ordine in cui gli attributi sono memorizzati nel record (se differente da quello di default)

```
CREATE TABLE MovieStar (
name      CHAR(30) PRIMARY KEY,
address   CHAR(255),
gender    CHAR(1),
birthdate DATE )
```



Il modello di memorizzazione di DB2 : Creazione di un T.S.

All'atto della creazione di un tablespace è possibile specificare una serie di parametri, tra cui:

EXTENTSIZE: numero di blocchi dell'extent

BUFFERPOOL: nome del pool di buffer associato al tablespace

PREFETCHSIZE: numero di pagine da trasferire in memoria prima che vengano effettivamente richieste

OVERHEAD: stima del tempo medio di latenza per un'operazione di I/O

TRANSFERRATE: stima del tempo medio per il trasferimento di una pagina

Gli ultimi due parametri vengono usati dall'ottimizzatore

……

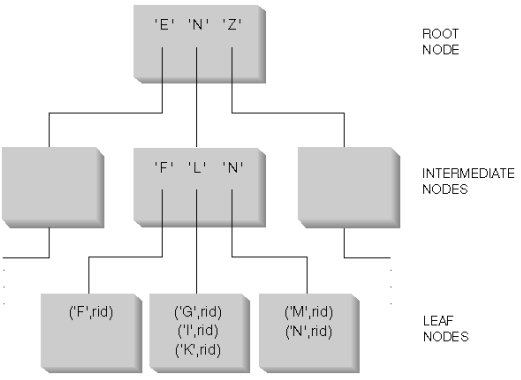
Indici su DB2 UDB

DB2 UDB Takes the Database Performance Triple Crown

DB2. Information Management Software

Struttura degli Indici su DB2 (1)

Il Database Manager di DB2 utilizza una struttura di tipo **B+TREE** per la memorizzazione degli indici.



Le foglie contengono coppie del tipo (K,P):

K-> è un valore chiave del campo su cui l' indice è costruito
P-> è un puntatore al record con valore di chiave K

Le foglie sono collegate in una lista per ottimizzare le ricerche su intervalli.

Nota: DB2 prevede un collegamento bidirezionale tra le foglie

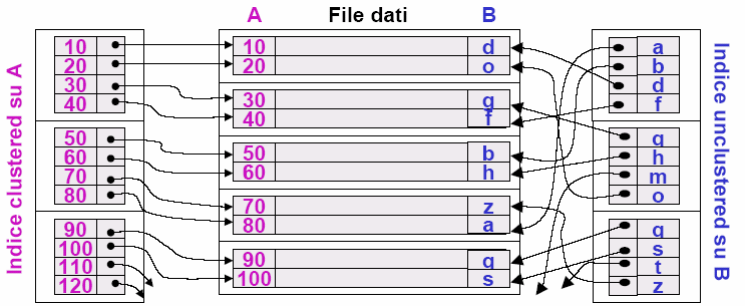
Struttura degli Indici su DB2 (2)

Un indice può essere definito come:

Univoco: deve essere costruito su un campo a valori non ripetuti (chiave relazionale)

Cluster: se è costruito sul campo su cui i record nel file dati sono mantenuti ordinati, altrimenti è detto ***unclustered***.

Ovviamente si può costruire al massimo un indice cluster per relazione, mentre si può costruire un arbitrario numero di indici *unclustered*

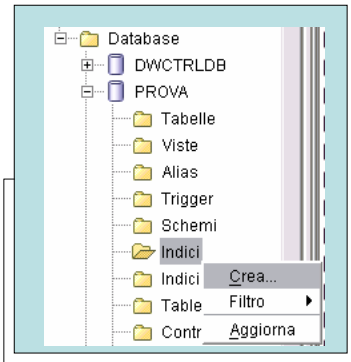


Creazione di un Indice su DB2

Al momento della creazione di una tabella DB2 crea automaticamente un indice Univoco sulla **chiave primaria** impostata.

L' aggiunta di ulteriori indici è estremamente semplice:

da Centro di Controllo



da Centro Comandi

In **SQL** la definizione di indici avviene mediante lo statement **CREATE INDEX** (ma non è standardizzato!) in DB2 (es):

```
CREATE UNIQUE INDEX MatrID
ON Studenti(Matricola) CLUSTER
```
indice univoco clusterizzato

```
CREATE INDEX DataNAscitaID
ON Studenti(datanascita)
ALLOW REVERSE SCANS
```
indice secondario con scansione inv.

```
CREATE INDEXS InfoID
ON Studenti(Cognome,Nome)
```
indice secondario multi attributo

NOTA: per creare automaticamente -> advisor di pianificazione

Osservazione su Indici Multi-Attributo

Un indice multi-attributo costruito su A1,A2,…An è efficace se in una Query si specificano valori per:

- tutti gli attributi
- i primi i < n

Attenzione: l'ordine di definizione è rilevante!

Ad esempio l'indice:

```
create index RicettaIndx
 on Ricette(cod_paziente,cod_cliente, data)
```

è utile solo per query che specificano valori per:

- tutti e tre gli attributi
- per `cod_paziente,cod_cliente`
- per il solo `cod_paziente`

se la clausola WHERE contenesse solo i predicati:

```
Data = '28-06-2002' AND cod_cliente = xxxxxxxxx
```

l'indice non verrebbe utilizzato dal DBMS!

Data Base di Esempio

TRE TABELLE su TABLESPACE SMS:

- **Farmaci (codice, descrizione, prezzo, tr)**

  <dimensione:**10985** record (≈ 38 byte), **528Kb, 132 pag.>**

  Indice su pk -> **PK_FARMACI 256Kb, 64 pag.**

- **Pazienti (codice_amministrativo, cf, nome, cognome, data_nascita)**

  <dimensione:**1443** record (≈ 63 byte),**108 Kb, 27 pag.>**

  Indice su pk -> **PK_PAZIENTI 36Kb,9pag.**

- **Ricetta (codice_paziente, codice_farmaco, data)**

  <dimensione:**10** record (≈29 byte)>

  Indice su pk -> **PK_RICETTA**

Vari altri indici creati per prova occasionalmente

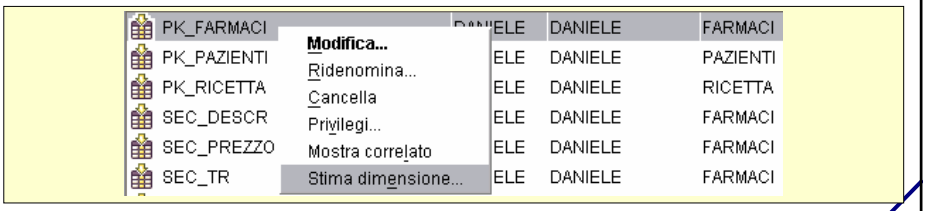
INDICI & CATALOGO(1)

ATTENZIONE: Dopo la creazione di un indice, o dopo grosse modifiche è indispensabile aggiornare il catalogo con il comando **RUNSTATS**! Altrimenti l'indice **non verrà utilizzato** dal DB.

da Centro Comandi

```
CONNECT TO PROVA;
RUNSTATS ON TABLE DANIELE.FARMACI ON ALL COLUMNS AND INDEX  DANIELE.PK_FARMACI ,
DANIELE.SEC_DESCR ,  DANIELE.SEC_PREZZO ,  DANIELE.SEC_TR ALLOW WRITE ACCESS ;
COMMIT WORK;
CONNECT RESET;
```

da Centro di Controllo



INDICI & CATALOGO(2)

Le **informazioni/statistiche** relative agli indici creati si trovano nella tabella:

SYSIBM.SYSINDEXES

Troviamo informazioni su: Numero di livelli dell' indice, numero di valori di chiave numero di pagine foglia dell' indice ecc. ecc.

Nel nostro caso:

```
select NAME,NLEAF,NLEVELS,FIRSTKEYCARD from SYSIBM.SYSINDEXES where creator = 'DANIELE'
```

NAME	NLEAF	NLEVELS	FIRSTKEYCARD
PK_FARMACI	64	2	10985
PK_PAZIENTI	9	2	1443
PK_RICETTA	1	1	3
SEC_DESCR	52	2	6654
SEC_PREZZO	26	2	2326
SEC_TR	19	2	37

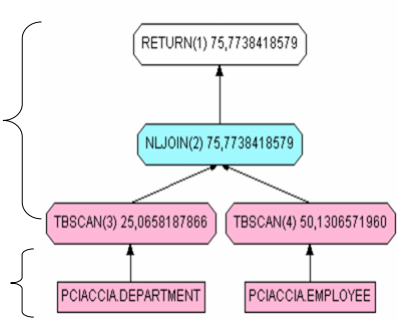
Piani di Accesso

Durante la fase di "**ottimizzazione basata sui costi**" di una query, facendo anche uso delle informazioni statistiche sui dati, viene scelto il modo "**più economico**" per eseguire la query, ovvero il modo che complessivamente presenta un costo minimo tra le alternative possibili.

Ogni modo di eseguire una query è un **piano di accesso**, e si compone di una serie di **operatori** connessi ad albero.

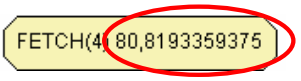
Gli **altri nodi** sono operatori che agiscono su 1 o 2 insiemi di tuple in input e producono 1 insieme di tuple in output

Le **foglie** del piano di accesso sono le relazioni di base presenti nella query.



Costi sul Piano di Accesso

Ad ogni operatore del piano è associato un **Costo (complessivo del sottoalbero).**



- Tale costo è l'utilizzo delle risorse totali stimato necessario all'esecuzione del plan di accesso per un'istruzione.
- Il costo viene ricavato da una combinazione fra il costo della CPU (in numero di istruzioni) ed I/O (in numero di ricerche e trasferimenti pagina).
- L'unità del costo è il ***timeron***.
  Un timeron non è equiparabile al tempo impiegato effettivamente ma fornisce una stima approssimativa delle risorse richieste (costo) dal Database Manager per eseguire due plan della stessa interrogazione.

Query su Indice Primario

select * from FARMACI where CODICE = '22534123'

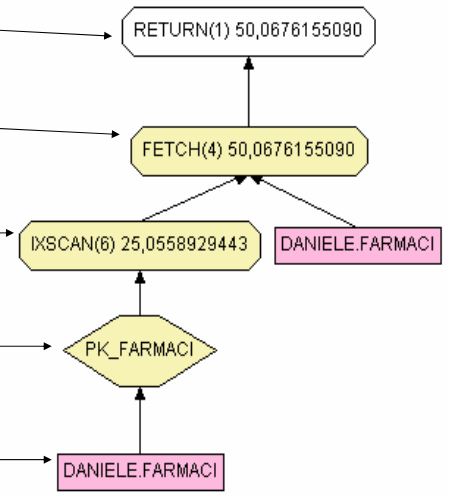
Rappresenta la restituzione dei dati all'utente

Partendo dai RID recupera le righe dalla tabella

Compie una ricerca nell'indice di una tabella producendo un flusso ordinato di righe

Indici, visualizzati come diamanti

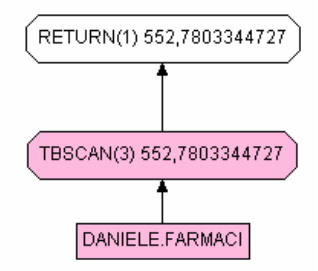
Tabelle, visualizzate come rettangoli



Query su Indice Secondario

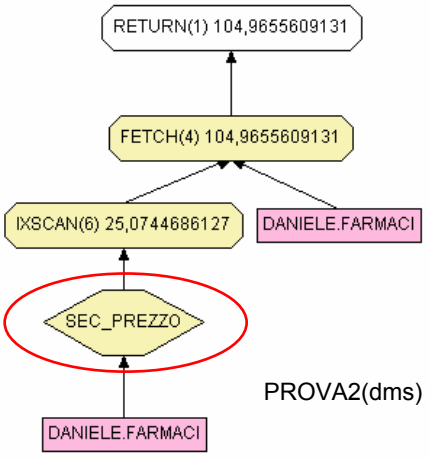
select * from FARMACI where PREZZO = 0.6

(Risultato senza indice)



PROVA2(dms) = 221

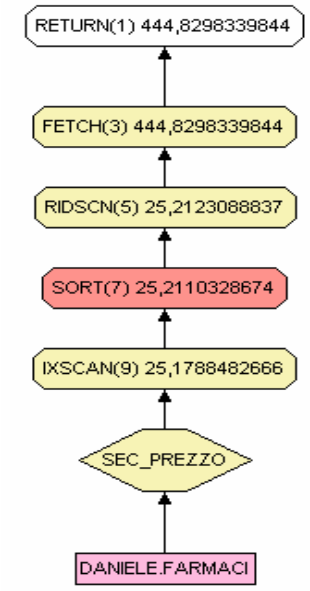
(Risultato con indice)



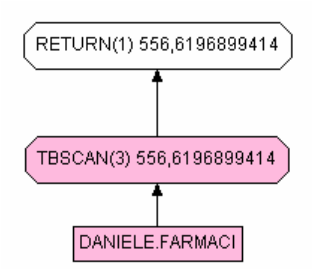
PROVA2(dms) = 42

Query su intervallo

select * from FARMACI
where PREZZO<10 and PREZZO>5

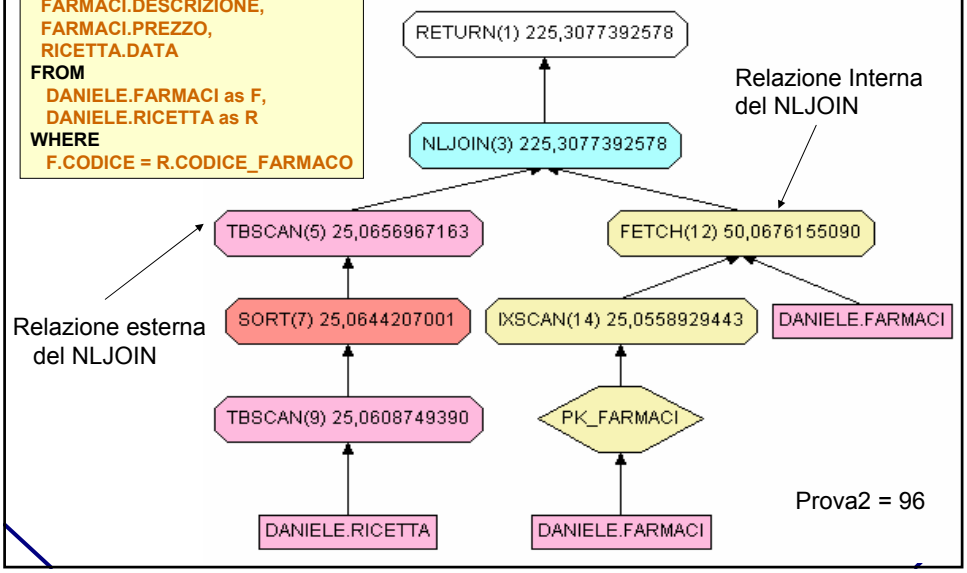


select * from FARMACI where PREZZO/10 = 1



JOIN (1)

```
SELECT
  FARMACI.DESCRIZIONE,
  FARMACI.PREZZO,
  RICETTA.DATA
FROM
   DANIELE.FARMACI as F,
   DANIELE.RICETTA as R
WHERE
   F.CODICE = R.CODICE_FARMACO
```

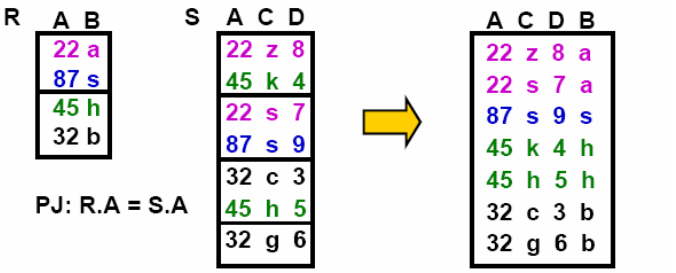


Prova2 = 96

IL NESTED LOOP JOIN(1)

Date 2 relazioni in input **R** e **S** tra cui sono definiti i predicati di join **PJ**, e supponendo che **R** sia la **relazione esterna**, l'algoritmo opera come segue:

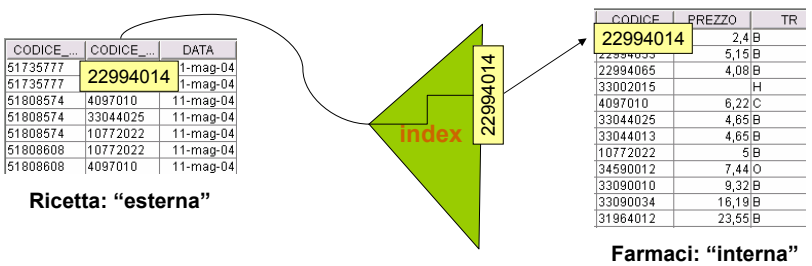
Per ogni tupla **ta** in R**:**
{ Per ogni tupla **tb** in S**:**
{ se la coppia(**ta**, **tb**) soddisfa PJ
allora aggiungi(**ta**, **tb**)al risultato**} }**



OSSERVAZIONE:
L'ordine con cui vengono generate le tuple del risultato coincide con l'ordine eventualmente presente nella relazione esterna -> può influenzare l' OT.

IL NESTED LOOP JOIN(2)

Data una tupla della relazione esterna **Ricetta**, la **scansione completa** della relazione interna **Farmaci** può essere sostituita da una **scansione basata su un indice** costruito sugli attributi di join di **Farmaci**:



Ricetta: "esterna"

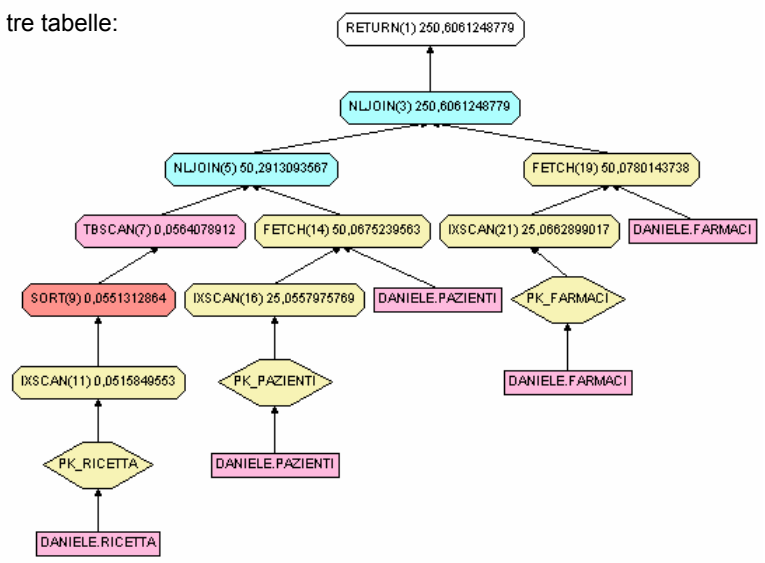
Farmaci: "interna"

PJ: F.CODICE = R.CODICE_FARMACO

L 'accesso alla relazione interna mediante indice porta in generale a ridurre di molto i costi di esecuzione del Nested Loops Join

JOIN (2)

Join sulle tre tabelle:



JOIN (3)

Oltre al **Nested Loop Join** **DB2 UDB** può utilizzare anche:

Il Merge Join:

applicabile quando entrambi gli insiemi di tuple in input sono ordinati sugli attributi di join

L' Hash Join:

non richiede né la presenza di indici né input ordinati, e risulta particolarmente vantaggioso in caso di relazioni molto grandi

MATERIALIZZAZIONE vs PIPELINE(1)

Per eseguire un **Piano di Accesso** possiamo pensare di agire in 2 modi:

1. **"Valutazione per Materializzazione":** consiste nel procedere bottom-up secondo il seguente schema:

   • Si calcolano innanzitutto i risultati degli operatori al livello più basso dell' albero e si memorizzano tali risultati in relazioni temporanee.

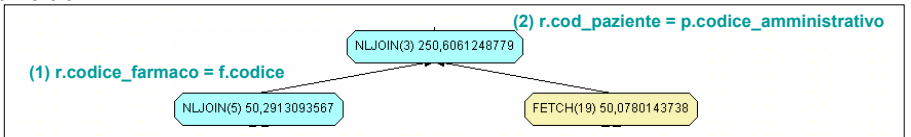
   • Si procede quindi in modo analogo per i livelli superiori fino alla radice

E' **altamente inefficiente**, in quanto comporta la **creazione**, **scrittura** e **lettura** di molte **relazioni temporanee**.

Relazioni che, se la dimensione dei risultati intermedi eccede lo spazio disponibile in memoria centrale, devono essere gestite su disco.

MATERIALIZZAZIONE vs PIPELINE(2)

2. **"Valutazione in Pipeline":** consiste nel **eseguire più operatori in pipeline**, ovvero non aspettare che termini l'esecuzione di un operatore per iniziare l'esecuzione di un altro.



1. Si inizia eseguendo il primo join (1). Appena viene prodotta la prima tupla dell' output, questa viene passata in input al secondo join (2) che può quindi iniziare la ricerca di matching tuples e incominciare a produrre il risultato finale.
2. Quando il join (2) avrà scandito tutta la relazione interna, chiederà al join (1) di passargli un'altra tupla che nel frattempo avrà prodotto ecc. ecc.

Molto più efficiente ma richiede una gestione più complessa.

Nota: Non tutti gli operatori possono lavorare in pipeline (es. SORT)

FONTI

- Guida in Linea di DB2 UDB
- Administrator Guide -> publib.boulder.ibm.com