Introduzione all’architettura di un DBMS

Architettura di un DBMS

- Un DBMS deve garantire una gestione dei dati:
  - efficiente
  - concorrente
  - affidabile
  - integra
  - sicura (protetta)
- Ciascuno degli aspetti precedenti è supportato dal DBMS da specifiche componenti, che complessivamente rappresentano l’architettura del sistema

Componenti di un DBMS

- **Efficienza**:
  - **File system**: gestisce l'allocazione dello spazio su disco e le strutture dati usate per rappresentare le informazioni memorizzate su disco
  - **Buffer manager**: responsabile per il trasferimento delle informazioni tra disco e main memory
  - **Query parser**: traduce i comandi del DDL e del DML in un formato interno (parse tree)
  - **Optimizer**: trasforma una richiesta utente in una equivalente ma più efficiente

Componenti di un DBMS

- **Affidabilità**:
  - **Recovery manager**: assicura che il DB rimanga in uno stato consistente a fronte di cadute del sistema
- **Concorrenza**:
  - **Concurrency controller**: assicura che interazioni concorrenti procedano senza conflitti
- **Integrità**:
  - **Integrity manager**: controlla che i vincoli di integrità (per esempio le chiavi) siano verificati
- **Sicurezza**
  - **Authorization manager**: controlla che gli utenti abbiano i diritti di accesso ai dati

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 B-tree o tabelle hash)
  - dati statistici (esempio il numero di tuple in una relazione) che sono usati per determinare la strategia ottima di esecuzione

Efficienza

- Finora abbiamo visto modelli di DBMS ad alto livello (**livello logico**)
  - e' il livello corretto per gli utenti del DB
- 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

Efficienza

- 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**)

Efficienza

- Cenni a:
  - supporti di memorizzazione
  - organizzazione di file
  - mapping di relazioni a file
  - clustering
  - strutture ausiliarie di accesso
  - gestione del buffer
  - esecuzione di interrogazioni

Efficienza- supporti di memorizzazione

- I dati memorizzati in una base di dati devono essere fisicamente memorizzati su un supporto fisico di memorizzazione
- Memoria primaria
  - memoria principale (main memory) e memorie più piccole e più veloci
  - i dati sono manipolati direttamente dalla CPU
  - accesso veloce ai dati
  - capacita' di memorizzazione limitata
  - volatile (il contenuto e' perso se va via la corrente o si ha una caduta di sistema)

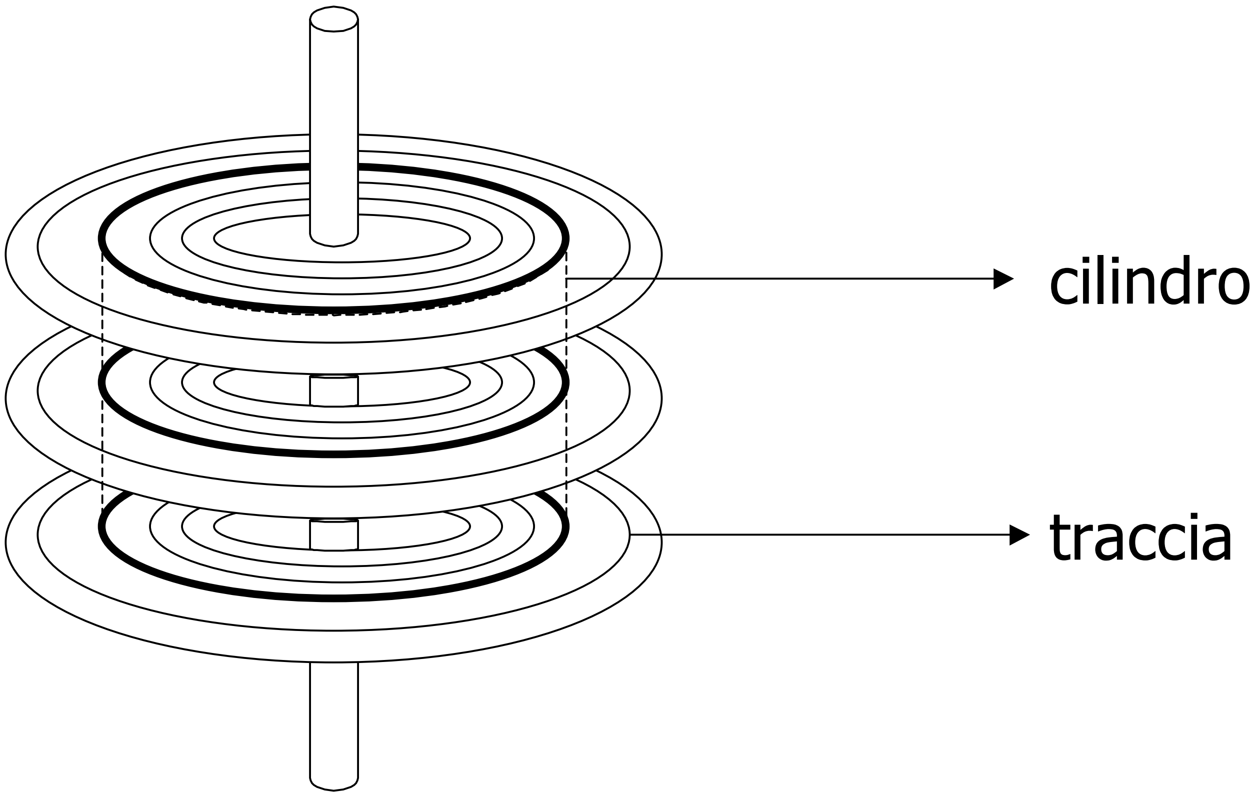
Efficienza- supporti di memorizzazione

- Memoria secondaria
  - Dischi magnetici, dischi ottici e nastri
  - capacità maggiore, costo inferiore e accesso più lento
  - necessità di trasferire i dati in memoria principale per elaborazione dalla CPU
  - non volatile
- Basi di dati in genere sono memorizzate su memoria secondaria (dischi magnetici)
  - troppo grosse per risiedere in memoria principale
  - maggiori garanzie di persistenza dei dati
  - costo per unità di memorizzazione decisamente inferiore

Efficienza - Dischi

- L'informazione è memorizzata su una superficie del disco in cerchi concentrici di piccola ampiezza, ognuno con un diametro distinto, detti tracce
- per i dischi a più piatti, le tracce con lo stesso diametro sulle varie superfici sono dette **cilindro**
- dati memorizzati su uno stesso cilindro possono essere recuperati molto più velocemente che non dati distribuiti su diversi cilindri

Efficienza- Dischi



Efficienza - Dischi

- I dati sono trasferiti tra il disco e la memoria principale in unità chiamate blocchi
  - un blocco è una sequenza di byte contigui memorizzati in una stessa traccia di un singolo cilindro
  - la dimensione del blocco dipende dal sistema operativo
  - il tempo di trasferimento di un blocco è il tempo impiegato dalla testina per trasferire un blocco nel buffer, una volta posizionata all'inizio del blocco
  - tale tempo è molto più breve del tempo necessario per posizionare la testina all'inizio del blocco (tempo di seek)

Efficienza - Organizzazione di file

- I dati sono generalmente memorizzati in forma di record
  - un record è costituito da un insieme di valori (campi) collegati
- un file è una sequenza di record
  - file con record a lunghezza fissa se tutti i record memorizzati nel file hanno la stessa dimensione (in byte)
  - file con record a lunghezza variabile sono pero' necessari per:
    - memorizzazione di tipi di record diversi nello stesso file
    - memorizzazione di tipi di record con campi di lunghezza variabile
    - memorizzazione di tipi di record con campi multivalore (es. basi di dati O-O o OR)

Efficienza - Organizzazione di file

- Un file può essere visto come una collezione di record
- Tuttavia, poiché i dati sono trasferiti in blocchi tra la MS e la MM, e' importante assegnare i record ai blocchi in modo tale che uno stesso blocco contenga record tra loro interrelati
- Se si riesce a memorizzare sullo stesso blocco record che sono spesso richiesti insieme si risparmiano accessi a disco

Efficienza - Mapping di relazioni a file

- Per DBMS di piccole dimensioni (es. per PC) una soluzione spesso adottata è di memorizzare ogni relazione in un file separato
- Per DBMS large scale una strategia frequente è di allocare per il DBMS un unico grosso file, in cui sono memorizzate tutte le relazioni
  - la gestione di questo file è lasciata al DBMS

Efficienza - 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

Efficienza – 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

Efficienza - 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)

Efficienza - 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

Efficienza - Strutture ausiliarie di accesso

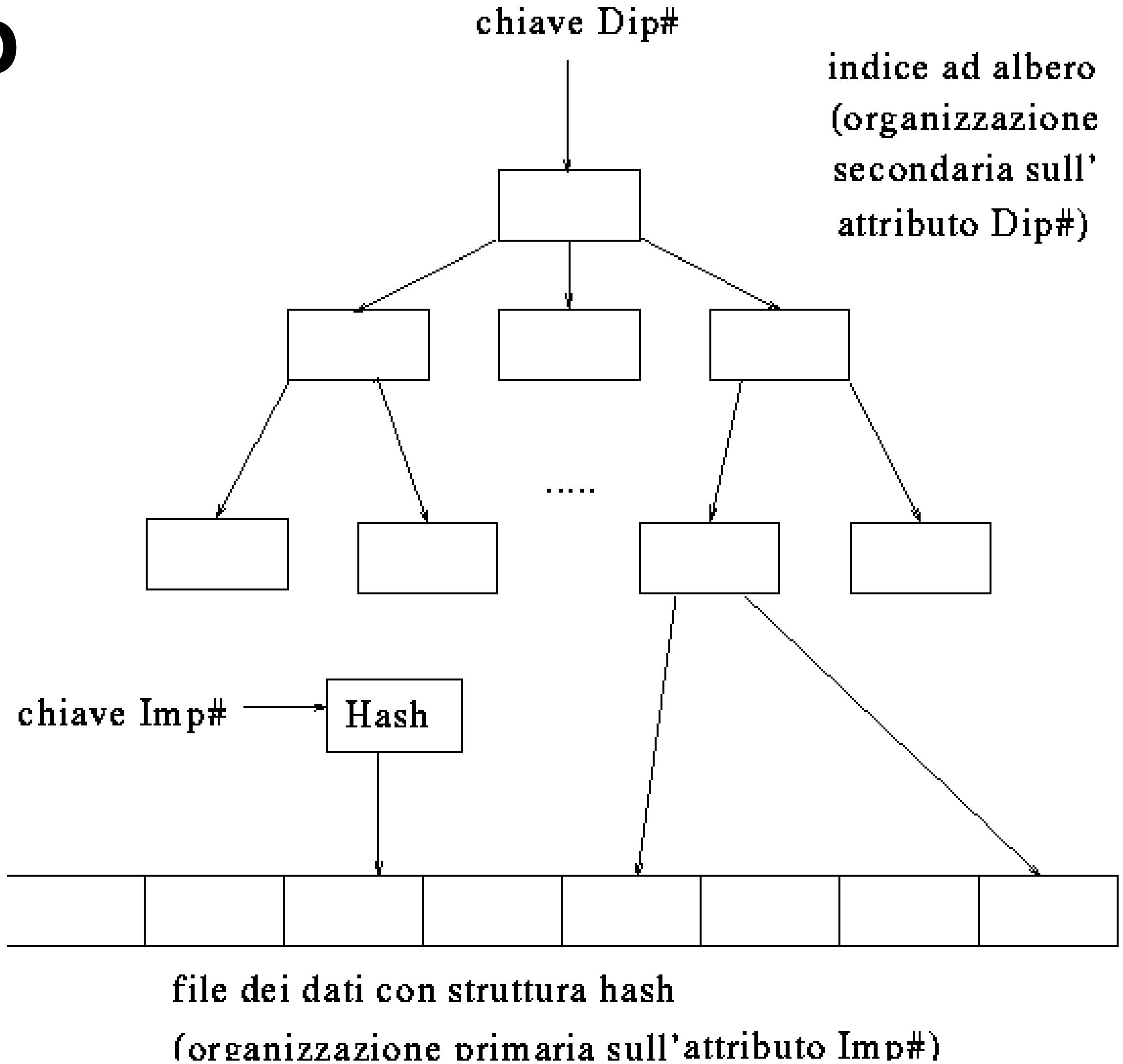
- Per effettuare la ricerca in modo più efficiente si può pensare di mantenere il file ordinato secondo il valore di una chiave di ricerca

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

Efficienza - 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

Efficienza - Strutture ausiliarie di accesso



Efficienza - Indici

- Idea base: associare al file dei dati una “tabella'’ nella quale l'entrata i-esima 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

Efficienza - 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

Efficienza - 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

Efficienza - 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)

Efficienza - 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

dirigente	
ingegnere	
segretaria	
tecnico	

7977	Verdi	dirigente	10-Dic-80	3000	?	10
7566	Rosi	dirigente	02-Apr-81	2975	?	20
7698	Biacchi	dirigente	01-Mag-81	2850	?	30
7369	Rossi	ingegnere	17-Dic-80	1600	500	20
7782	Neri	ingegnere	01-Giu-81	2450	200	10
7839	Dare	ingegnere	17-Nov-81	2600	300	10
7876	Adami	ingegnere	23-Set-81	1100	150	20
7900	Gianni	ingegnere	03-Dic-81	1950	?	30
7934	Milli	ingegnere	23-Gen-82	1300	150	10
7902	Fordi	segretaria	03-Dic-81	1000	?	20
7654	Martini	segretaria	28-set-81	800	?	30
7788	Scotti	segretaria	09-Nov-81	800	?	20
7521	Bianchi	tecnico	20-Feb-81	800	100	30
7499	Andrei	tecnico	20-Feb-81	800	?	30
7844	Turni	tecnico	08-Set-81	1500	?	30

Efficienza – indice sparso

dirigente	
segretaria	

7977	Verdi	dirigente	10-Dic-80	3000	?	10
7566	Rosi	dirigente	02-Apr-81	2975	?	20
7698	Biacchi	dirigente	01-Mag-81	2850	?	30
7369	Rossi	ingegnere	17-Dic-80	1600	500	20
7782	Neri	ingegnere	01-Giu-81	2450	200	10
7839	Dare	ingegnere	17-Nov-81	2600	300	10
7876	Adami	ingegnere	23-Set-81	1100	150	20
7900	Gianni	ingegnere	03-Dic-81	1950	?	30
7934	Milli	ingegnere	23-Gen-82	1300	150	10
7902	Fordi	segretaria	03-Dic-81	1000	?	20
7654	Martini	segretaria	28-set-81	800	?	30
7788	Scotti	segretaria	09-Nov-81	800	?	20
7521	Bianchi	tecnico	20-Feb-81	800	100	30
7499	Andrei	tecnico	20-Feb-81	800	?	30
7844	Turni	tecnico	08-Set-81	1500	?	30

Efficienza - 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

Efficienza - 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)

Efficienza - 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

Efficienza - 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

Efficienza - 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

Efficienza - 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)

Efficienza - Gestione del buffer

- L'obiettivo principale delle strategie di memorizzazione è di minimizzare gli accessi a disco
- Un altro modo è di mantenere più blocchi possibile in MM
- Si usa un buffer che permette di tenere in MM copia di alcune pagine di disco
- Il buffer è organizzato in pagine, che hanno la stessa dimensione dei blocchi

Efficienza - Gestione del buffer

- Quando una pagina della MS è presente nel buffer, il DBMS puo' effettuare le sue operazioni di lettura e scrittura direttamente su di essa
- I tempi di accesso alla MS sono dell'ordine di millesimi di secondo mentre quelli di accesso alla MM sono dell'ordine di milionesimi di secondo
- Accedere alle pagine nel buffer invece che alle corrispondenti pagine su disco influenza notevolmente le prestazioni

Efficienza - Gestione del buffer

- Il buffer manager di un DBMS usa alcune politiche di gestione che sono piu' sofisticate delle politiche usate nei SO:
  - le politiche di LRU non sempre sono le più adatte per i DBMS
  - per motivi legati alla gestione del recovery in alcuni casi un blocco non può essere trasferito su disco (in tal caso il blocco e' detto pinned)
  - per motivi legati alla gestione del recovery in alcuni casi e' necessario forzare un blocco su disco
  - Un DBMS e' in grado di predire meglio di un SO il tipo dei futuri riferimenti

Efficienza – Gestione del buffer: Esempio

- Operazione di join Impiegati |x| Dipartimenti (assumendo che le due relazioni siano in due file diversi)
- relazione Impiegati
  - una volta che una tupla della relazione e' stata usata non e' piu' necessaria
  - non appena tutte le tuple di un blocco sono state esaminate il blocco non serve piu' (strategia toss-immediate)

Efficienza – Gestione del buffer: Esempio

- relazione Dipartimenti
  - il blocco più recentemente acceduto sarà riferito di nuovo solo dopo che tutti gli altri blocchi saranno stati esaminati
  - la strategia migliore per il file Dipartimenti è di rimuovere l'ultimo blocco esaminato (strategia most recently used - MRU)
  - è però necessario eseguire il pin del blocco correntemente esaminato fino a che si siano esaminate tutte le tuple; quindi si può rendere il blocco unpinned

Efficienza - Esecuzione di interrogazioni

- Abbiamo visto finora come organizzare i dati in un DB
- Le decisioni sulle strutture da allocare sono determinate durante la progettazione fisica del DB
- La modifica di tali strutture in seguito può essere costosa
- Quindi quando una query è presentata al sistema occorre determinare il modo più efficiente per eseguirla usando le strutture disponibili

Efficienza - Esecuzione di interrogazioni

- Per interrogazioni complesse esistono più strategie possibili
- Anche se il costo di determinare la strategia ottima può essere alto, il vantaggio in termini di efficienza che se ne ricava è tale che in genere conviene eseguire l'ottimizzazione

Efficienza - Passi nell’esecuzione di un’interrogazione

- Parsing
  - Viene controllata la correttezza sintattica della query e ne viene generata una rappresentazione interna (parse tree)
- Trasformazioni algebriche
  - La query viene trasformata in una query equivalente ma più efficiente da eseguire (ci si basa sulle proprietà dell'algebra relazionale)
  - Esempi:
    - eseguire le operazioni di selezione prima possibile
    - Evitare join che rappresentano prodotti Cartesiani

Efficienza - Passi nell’esecuzione di un’interrogazione

- Selezione della strategia
  - Si determina in modo preciso come la query sarà eseguita (per esempio si determina che indici si useranno)
  - La scelta della strategia è fatta principalmente in base al numero di accessi a disco
- Esecuzione della strategia scelta
  - E' possibile eseguire alcuni dei passi a tempo di compilazione del programma (DB2 e System R usano questa strategia) o a tempo di esecuzione (Oracle usa questa strategia)

Efficienza - Perché ottimizzare le interrogazioni?

- Consideriamo le relazioni
  - Studenti(MatrS,Nome,Ind,AltreInfo)
  - Esami(Corso,MatrS,Voto,Data)
- Supponiamo di voler trovare il nome degli studenti e la data degli esami per gli studenti che hanno sostenuto BD con 30
  - SELECT Nome,Data
  - FROM Studenti NATURAL JOIN Esami
  - WHERE Corso ='BD' AND Voto =30

Efficienza - Perché ottimizzare le interrogazioni?

- Consideriamo un database con 2.000 studenti e 20.000 esami, di cui 500 di BD e di questi solo 50 con 30 (consideriamo solo la scansione sequenziale delle relazioni)
- Se si fa il prodotto cartesiano delle due relazioni, si ottiene una relazione temporanea con 40.000.000 tuple, da queste si estraggono poi le 50 tuple desiderate (costo proporzionale a 120.000.000 accessi)
- Se si selezionano i 50 esami di BD con 30 e poi si fa il join di questa relazione temporanea con Studenti si ha un costo proporzionale a 120.050

Selezione della strategia

- La strategia scelta dipende dalla dimensione di ogni relazione e dalla distribuzione dei valori nelle varie colonne
- Per selezionare la strategia si utilizzano stime del costo di esecuzione, basate su profili delle relazioni
- Quindi normalmente i DBMS mantengono statistiche per ogni relazione memorizzata

Efficienza - Selezione della strategia

- Esempi di statistiche:
  - n_r numero di tuple nella relazione r
  - S_r dimensione di una tupla della relazione r in byte (per tuple a lunghezza fissa, altrimenti si usano valori medi)
  - V(A, r) il numero di valori distinti che appaiono nella relazione r per l'attributo A
  - SA dimensione in byte di ciascun attributo A
  - NPAG(r), numero delle pagine di r
  - per ogni indice I: NLEAF(I), numero di foglie dell'indice I
- Le statistiche sono aggiornate in seguito al caricamento delle relazioni o alla creazione di un indice (ma l'utente puo' richiedere esplicitamente il loro aggiornamento)

Efficienza - Selezione della strategia

- Granularita' dell'ottimizzazione
  - L'ottimizzazione e' eseguita separatamente per ogni interrogazione
  - pertanto è bene formulare interrogazioni complesse e non scomporle in tante interrogazioni più semplici perché ciò fa perdere i vantaggi dell'ottimizzazione

Transazioni

- Per mantenere le informazioni consistenti è necessario controllare opportunamente le sequenze di accessi e aggiornamenti ai dati
- Gli utenti interagiscono con la base di dati attraverso programmi applicativi ai quali viene dato il nome di **transazioni**
- Una transazione si può interpretare come un insieme parzialmente ordinato di operazioni di lettura e scrittura; essa costituisce l'effetto dell'esecuzione di programmi che effettuano le funzioni desiderate dagli utenti

Transazioni

- Ogni transazione è eseguita o completamente (cioè effettua il commit), oppure per nulla (cioè effettua l'abort) se si verifica un qualche errore (hardware o software) durante l'esecuzione
  - Necessità di garantire che le transazioni eseguite concorrentemente si comportino come se fossero state eseguite in sequenza (controllo della concorrenza)
  - Necessità di tecniche per ripristinare uno stato corretto della base di dati a fronte di malfunzionamenti di sistema (tecniche di ripristino - recovery-)

Transazioni- Proprietà

- L'insieme di operazioni che costituiscono una transazione deve soddisfare alcune proprietà, note come proprietà ACID:
  - Atomicità
  - Consistenza
  - Isolamento
  - Durabilità

Transazioni - Proprietà

- **Atomicità**:
  - e' detta anche proprietà tutto-o-niente
  - tutte le operazioni di una transazione devono essere trattate come una singola unità: o vengono eseguite tutte, oppure non ne viene eseguita alcuna
  - l'atomicità delle transazioni e' assicurata dal sottosistema di ripristino (recovery)

Transazioni – Proprietà

- **Consistenza**:
  - una transazione deve agire sulla base di dati in modo corretto
  - se viene eseguita su una base di dati in assenza di altre transazioni, la transazione trasforma la base di dati da uno stato consistente (cioè che riflette lo stato reale del mondo che la base di dati deve modellare) ad un altro stato ancora consistente
  - l'esecuzione di un insieme di transazioni corrette e concorrenti deve a sua volta mantenere consistente la base di dati
  - il sottosistema di controllo della concorrenza (concurrency control) sincronizzale transazioni concorrenti in modo da assicurare esecuzioni concorrenti libere da interferenze

Transazioni - Proprietà

- **Isolamento**:
  - ogni transazione deve sempre osservare una base di dati consistente, cioè, non può leggere risultati intermedi di altre transazioni
  - la proprietà di isolamento è assicurata dal sottosistema di controllo della concorrenza che isola gli effetti di una transazione fino alla sua terminazione

Transazioni - Proprietà

- **Durabilità** (persistenza):
  - i risultati di una transazione terminata con successo devono essere resi permanenti nella base di dati nonostante possibili malfunzionamenti del sistema
  - la persistenza è assicurata dal sottosistema di ripristino
  - tale sottosistema può inoltre fornire misure addizionali, quali back-up su supporti diversi e journaling delle transazioni, per garantire la durabilità anche a fronte di guasti ai dispositivi i memorizzazione

Transazioni - Proprietà

- Le proprieta' ACID vengono assicurate utilizzando due insiemi distinti di algoritmi o protocolli, che assicurano:
  - l'atomicità dell'esecuzione
    - mantenere la consistenza globale della base di dati e quindi assicurare la proprietà di consistenza delle transazioni (anche concorrenti)
    - protocolli di controllo della concorrenza
  - l'atomicità del fallimento
    - assicura l'atomicita', l'isolamento e la persistenza
    - protocolli di ripristino

Transazioni – Modello flat

- Facciamo riferimento al modello di transazioni più semplice (transazioni flat), che prevede un solo livello di controllo a cui appartengono tutte le transazioni eseguite (è il modello usato nei DBMS commerciali)
- Tutte le istruzioni eseguite devono essere contenute tra le istruzioni BeginWork e CommitWork
  - l'istruzione BeginWork dichiara l'inizio di una transazione flat
  - l'istruzione CommitWork è invocata per indicare che il sistema ha raggiunto un nuovo stato consistente

Transazioni - Modello flat

- La transazione può terminare la propria esecuzione con successo (commit) e rendere definitivi i cambiamenti prodotti sulla base di dati dalle istruzioni eseguite tra BeginWork e CommitWork, oppure sarà disfatta (cioe' i suoi effetti saranno annullati) e tutti gli aggiornamenti eseguiti andranno persi (abort)
- In questo caso, si dice che viene eseguito il rollback della transazione

Transazioni - Modello flat

- I vari DBMS forniscono specifiche istruzioni SQL per supportare l'uso di transazioni flat
  - tali istruzioni sono invocate all'interno di programmi applicativi (nelle tre modalità viste in precedenza) e consentono al programmatore di identificare l'inizio e la fine di una transazione

Transazioni – PL/SQL

- Il primo statement SQL di un programma inizia implicitamente una transazione
- La transazione termina con i seguenti comandi:
  - COMMIT
  - ROLLBACK
  - Fine del programma, in assenza dei comandi precedenti

Transazioni – T-SQL

- Diversi modi per iniziare una transazione:
  - transazioni esplicite
  - transazioni autocommit
  - transazioni implicite
- due possibili risultati transazionali:
  - commit: tutte le modifiche effettuate dalla transazione vengono rese permanenti
  - rollback: nessuna modifica viene eseguita a causa di errori
- il rollback viene effettuato automaticamente nel caso di gravi errori, non gestiti a livello di applicazione (es. Si rompe il disco)

Transazioni – T-SQL: esplicite

- Vengono utilizzati comandi T-SQL per specificare l'inizio e la fine della transazione
- Inizio:
  - BEGIN TRANSACTION
- Fine:
  - COMMIT TRANSACTION
  - ROLLBACK TRANSACTION

Transazioni – T-SQL: esplicite, esempio

```
BEGIN TRANSACTION
UPDATE Impiegati
SET Stipendio = Stipendio * 1.2;
UPDATE Dipartimenti
SET Dirigente = 3
IF (@@ERROR = 0)
    BEGIN
        PRINT 'Transazione eseguita'
        COMMIT TRANSACTION
    END
ELSE
    BEGIN
        PRINT 'Errore'
        ROLLBACK TRANSACTION
    END
```

Transazioni – T-SQL: autocommit

- Default
- viene effettuato il commit di ogni statement T-SQL che non genera errori
- se viene generato almeno un errore, lo statement viene abortito
- in presenza di batch, l'autocommit rimane a livello di singolo statement
- l'autocommit viene annullato quando si comincia in modo esplicito una transazione
- viene ripristinato quando la transazione esplicita termina

Transazioni – T-SQL: implicite

- Simili alle esplicite, ma non si deve specificare l'inizio delle transazioni
- al termine di ogni transazione (COMMIT o ROLLBACK) viene automaticamente iniziata una nuova transazione
- Per attivare le transazioni implicite:
  - SET IMPLICIT_TRANSACTION ON
- Per disattivarle:
  - SET IMPLICIT_TRANSACTION OFF

Transazioni - JDBC

- Per default, JDBC esegue il commit di ogni statement SQL inviato al DBMS (auto-commit)
- è possibile disattivare l'autocommit tramite il metodo setAutoCommit
  - Connection con;
  - …
  - con.setAutoCommit(false);
- a questo punto ogni connessione diventa una transazione indipendente per la quale si può richiedere il commit o l'abort tramite:
  - con.commit();
  - con.abort();

Transazioni – JDBC: esempio

```
...
Connection ARS;
ARS =DriverManager.getConnection(ARS_URL,
                    "whitney", "secret");
ARS.setAutoCommit(false);
Statement selImp = ARS.createStatement ();
String stmt1 =
 ”UPDATE Impiegati SET Stipendio = 1000 WHERE
                           Cognome = ‘Rossi’";
String stmt2 =
 ”DELETE FROM Impiegati WHERE Cognome = ‘Verdi’";

selImp.executeUpdate (stmt1);
selImp.executeUpdate (stmt2);
ARS.commit();
ARS.close();  } }
```

Transazioni - SQLJ

- Autocommit: come in JDBC
- Comandi espliciti:
  - #sql {COMMIT};
  - #sql{ROLLBACK};

Transazioni - Controllo della concorrenza

- **Scopo**:
  - garantire l'integrità della base di dati in presenza di accessi concorrenti da parte di più utenti necessità di sincronizzare le transazioni eseguite concorrentemente

Transazioni – Concorrenza: esempio

- base di dati che organizza le informazioni sui conti dei clienti di una banca
- il sig. Rossi e' titolare di due conti: un conto corrente (intestato anche alla sig.ra Rossi) e un libretto di risparmio, i cui saldi sono Lit. 100.000 e Lit. 1.000.000
- con la transazione T1 il sig. Rossi trasferisce Lit. 150.000 dal libretto di risparmio al conto corrente
- contemporaneamente con la transazione T2 la sig.ra Rossi deposita Lit.500.000 sul conto corrente

Transazioni – Concorrenza: esempio

T1	T2
Read(Lr)	
Lr = Lr -- 150000	
	Read(Cc)
Write(Lr)	
	Cc = Cc + 500000
Read(Cc)	
	Write(Cc)
	Commit
Cc = Cc + 150000	
Write(Cc)	
Commit	

Transazioni – Concorrenza: esempio

- La somma depositata da T2 è persa (lost update) e non si ottiene l'effetto voluto sulla base di dati
  - l'esecuzione concorrente di più transazioni genera un'alternanza di computazioni da parte delle varie transazioni, detta interleaving
  - l'interleaving tra le transazioni T1 e T2 nell'esempio produce uno stato della base di dati scorretto
  - si sarebbe ottenuto uno stato corretto se ciascuna transazione fosse stata eseguita da sola o se le due transazioni fossero state eseguite l'una dopo l'altra, consecutivamente

Transazioni – Concorrenza: lock

- Idea base:
  - ritardare l'esecuzione di operazioni in conflitto imponendo che le transazioni pongano dei blocchi (lock) sui dati per poter effettuare operazioni di lettura e scrittura
  - due operazioni si dicono in conflitto se operano sullo stesso dato e almeno una delle due e' un'operazione di scrittura
  - una transazione può accedere ad un dato solo se ha un lock su quel dato
- esistono vari protocolli di locking
  - il più noto: two-phase locking

Transazioni- Gestione del ripristino

- Tre principali tipi di malfunzionamenti:
  - **malfunzionamenti del disco**: le informazioni residenti su disco vengono perse (rottura della testina, errori durante il trasferimento dei dati)
  - **malfunzionamenti di alimentazione**: le informazioni memorizzate in memoria centrale e nei registri vengono perse
  - **errori nel software**: si possono generare risultati scorretti e il sistema potrebbe essere in uno stato inconsistente (errori logici ed errori di sistema)
- Il sottosistema di recovery (ripristino) deve identificare i malfunzionamenti e ripristinare la base di dati allo stato (consistente) precedente il malfunzionamento

Transazioni – Ripristino: classificazione delle memorie

- Ai fini del ripristino, le memorie vengono classificate come segue:
  - Memoria volatile
    - le informazioni contenute vengono perse in caso di cadute di sistema
    - esempi: memoria principale e cache
  - memoria non volatile:
    - le informazioni contenute sopravvivono a cadute di sistema, possono pero' essere perse a causa di altri malfunzionamenti
    - esempi: disco e nastri magnetici
  - memoria stabile:
    - le informazioni contenute non possono essere perse (astrazione)
    - se ne implementano approssimazioni, duplicando le informazioni in diverse memorie non volatili con
    - probabilita' di fallimento indipendenti

Transazioni – Ripristino: modello astratto

- Una transazione è sempre in uno dei seguenti stati:
  - **active**: lo stato iniziale
  - **partially committed**: lo stato raggiunto dopo che e' stata eseguita l'ultima istruzione
  - **failed**: lo stato raggiunto dopo aver determinato che l'esecuzione non puo' procedere normalmente
  - **aborted**: lo stato raggiunto dopo che la transazione ha subito un rollback e la base di dati e' stata ripristinata allo stato precedente l'inizio della transazione
  - **committed**: dopo il completamento con successo

Transazioni – Ripristino: modello astratto

- E' importante che una transazione non effettui scritture esterne osservabili (cioè scritture che non possono essere ``cancellate'', ad es. su terminale o stampante) prima di entrare nello stato di commit
- dopo il rollback di una transazione, il sistema ha due possibilità:
  - rieseguire la transazione ha senso solo se la transazione è stata abortita a seguito di errori software o hardware non dipendenti dalla logica interna della transazione
  - eliminare la transazione se si verificano degli errori interni che possono essere corretti solo riscrivendo il programma applicativo

Transazioni – Ripristino: esempio

- T transazione che trasferisce Lit. 100000 dal conto A al conto B
- A = Lit. 1000000, B = Lit. 15000000
- dopo la modifica di A e prima della modifica di B, si verifica una caduta di sistema e i contenuti della memoria vengono persi
  - se si riesegue T: stato (inconsistente) in cui A = Lit. 800000 e B = Lit. 15.100000
  - se non si riesegue T : stato corrente (inconsistente) in cui A = Lit. 900000 e B = Lit. 15000000
- in entrambi i casi lo stato risultante è inconsistente

Transazioni – Ripristino: meccanismi di recovery con log

- Durante l'esecuzione di una transazione tutte le operazioni di scrittura sono registrate in un particolare file gestito dal sistema, detto file di log
- concettualmente, il log può essere pensato come un file sequenziale, nell'implementazione effettiva possono essere usati più file fisici
- ad ogni record inserito nel log viene attribuito un identificatore unico (LSN, log sequence number o numero di sequenza di log) che in genere è l'indirizzo logico del record
- a fronte di un malfunzionamento, si utilizzano le informazioni contenute nel file di log per riportare la base di dati in uno stato consistente, rieseguendo o disfacendo le operazioni eseguite
- tramite il log, il sistema può gestire qualsiasi malfunzionamento che non implichi la perdita di informazioni contenute in memoria non volatile

Protezione dei dati

- Gli obiettivi della protezione dei dati sono:
  - **segretezza**: protezione delle informazioni da letture non autorizzate
  - **integrità**: protezione dei dati da modifiche o cancellazioni non autorizzate
  - **disponibilità**: garanzie che non si verifichino casi in cui ad utenti legittimi venga negato l'accesso ai dati
- l'importanza assegnata a tali obiettivi varia a seconda del sistema considerato
- il sottosistema di controllo dell'accesso regola le operazioni che si possono compiere sulle informazioni e le risorse in una base di dati

Protezione dei dati

- Lo scopo e' limitare e controllare le operazioni che gli utenti effettuano, prevenendo accidentali o deliberate azioni che potrebbero compromettere la correttezza e la sicurezza dei dati
- le risorse sono costituite dai dati, memorizzati in oggetti a cui si vuole garantire protezione
- i soggetti sono agenti (utenti o programmi in esecuzione) che richiedono di poter esercitare privilegi (come lettura, scrittura o esecuzione) sui dati

Protezione dei dati

- I modelli per la protezione dei dati derivano dai modelli per la protezione di risorse in un sistema operativo
- differenze fondamentali tra i due ambienti:
  - maggior numero di oggetti da proteggere
  - diversi livelli di granularita' (relazione, tupla, o singolo attributo)
  - protezione di strutture completamente logiche (viste) invece di risorse reali (file)
  - diversi livelli architetturali con requisiti di protezione differenti
  - rilevanza non solo della rappresentazione fisica dei dati, ma anche della loro semantica

Protezione dei dati - SQL

- La politica di controllo degli accessi in SQL e' una politica di tipo **discrezionale**
- mediante apposite regole di autorizzazione, vengono stabiliti i diritti che ogni soggetto possiede sugli oggetti del sistema
- il meccanismo di controllo esamina le richieste di accesso accordando solo quelle che sono autorizzate da una regola
- gli utenti con privilegi delegabili su un oggetto possono concedere o revocare tali diritti di accesso sull'oggetto ad altri utenti, a loro discrezione

Protezione dei dati - SQL

- Le politiche di tipo discrezionale sono molto flessibili ma non forniscono alcun controllo sul flusso di informazioni nel sistema, cioe' non impongono restrizioni sull'uso che un utente fa dell'informazione, una volta acceduta
- l'amministrazione dei privilegi e' decentralizzata mediante ownership: quando un utente crea una relazione, riceve automaticamente tutti i diritti di accesso su di essa ed anche la possibilita' di delegare ad altri tali privilegi

Protezione dei dati – SQL

- Gli oggetti del modello sono relazioni (sia di base che viste)
- i privilegi previsti sono:
  - **alter**: aggiungere una nuova colonna ad una relazione
  - **index**: creare un indice su una relazione
  - **delete**: cancellare tuple da una relazione
  - **insert**: inserire tuple in una relazione
  - **select**: selezionare tuple da una relazione
  - **update**: modificare valori di attributi in tuple di una relazione

Protezione dei dati – SQL: delega

- Il comando di delega dei privilegi e' il comando di GRANT
- se il comando contiene la clausola GRANT OPTION il privilegio e' delegabile
- **Esempio**:

```
GRANT SELECT, UPDATE(PremioP)
ON Impiegati
TO Rossi
WITH GRANT OPTION
```

- le informazioni sui privilegi sono memorizzate nei cataloghi (SYSAUTH e SYSCOLAUTH)

Protezione dei dati – SQL: revoke

- Il comando **REVOKE** permette di revocare privilegi
- un utente può revocare solo privilegi che lui ha concesso
- quando si esegue una operazione di revoca, l'utente a cui i privilegi sono stati revocati perde tali privilegi, a meno che essi non gli provengano anche da altre sorgenti indipendenti da quella che effettua la revoca
- dopo un'operazione di revoca il sistema deve ricorsivamente revocare tutti i privilegi che non avrebbero potuto essere concessi se l'utente specificato nel comando di revoca non avesse ricevuto il privilegio revocato (*revoca ricorsiva*)

Protezione dei dati – SQL: autorizzazioni su viste

- Le viste permettono di supportare il controllo dell'accesso in base al contenuto
- **esempio**: per autorizzare un utente a selezionare solo le tuple della relazione Impiegati relative ad impiegati che non guadagnano più di due milioni di lire:
  - si definisce una vista che seleziona dalla relazione Impiegati le tuple che soddisfano tale condizione
  - si concede all'utente il privilegio di select sulla vista, invece che sulla relazione di base
- permettono di delegare privilegi su singole colonne di relazioni: basta definire una vista come proiezione sulle colonne su cui si vogliono concedere i privilegi
- permettono di delegare privilegi statistici (media, somma, ecc.)