

Base di dati: è un insieme organizzato di grandi dati, persistente, condiviso e gestito dal DBMS

Catalogo: contiene la descrizione centralizzata dei dati

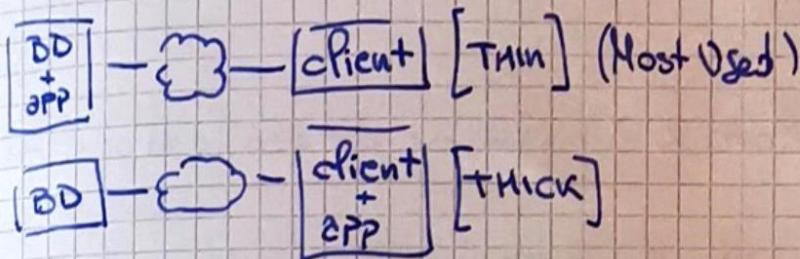
Il DBMS garantisce:

- privacy
- efficacia
- efficienza
- affidabilità

I DBMS (relazionali) sono sistemi transazionali

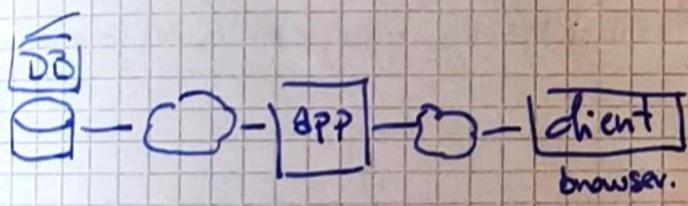
Architetture distribuite:

- Two Tier (client-server)



- 3 livelli

si ha un client (browser) e 2 server, uno applicativo ed uno per la gestione dei dati



- Si possono connettere sistemi eterogenei
- client-thin
- Scalabilità sul numero di client (add/paw-up a server)

Modello Relazionale

Proposto da Codd nel 1970, si basa sul concetto di relazione matematica con una variante.

Relazione matematica:

- Struttura posizionale
- Non c'è ordinamento fra le $n-u$ tuple

Siano $D_1 \dots D_n$ i domini, una relazione è un sottoinsieme del prodotto cartesiano tra i domini

Nella relazione di Codd ad ogni Dominio si associa un nome e la struttura smette quindi di essere posizionale

Le relazioni si rappresentano naturalmente con le tabelle.

- I valori di ogni colonna sono tra loro omogenei
- le righe sono diverse tra loro
- le intestazioni delle colonne sono diverse tra loro

Nel modello relazionale i riferimenti fra dati di relazioni diverse sono ottenuti tramite valori uguali in tuple diverse.

Definiamo:

Schemi di Relazione:

un nome R con un insieme di attributi $A_1 \dots A_n$

Schema di Base di Dati:

insieme di schemi di Relazione.

Tupla:

Una tupla su un insieme di Attributi X è una funzione che associa ad ogni $A \in X$ un valore del dominio di A

$t[A]$ denota il valore della tupla t sull'attributo A

Relazione su uno schema:

$R(x) = \text{insieme di tuple su } x$

Base di dati su uno schema:

$R = \{R_1(x_1) \dots R_n(x_n)\} = \text{insieme di relazioni } R = \{r_1, \dots, r_n\}$ (con r_i relazione su R_i)

NULL:

- val. sconosciuto
- val. inconsistente
- val. non interessante

Vincoli di Integrità:

Si associano alla base di dati delle proprietà che ne esprimono la "correttezza".

I vincoli di integrità:

- permettono una descrizione più accurata della realtà
- contribuiscono alla "qualità" dei dati
- si usano (più sa DBMS) nelle interrogazioni

Li definiamo come:

- un predicato che associa ad ogni istanza della base di dati il valore "vero" o "falso"

Esistono di 2 tipi:

- intrarelazionali
- interrelazionali

Intervarazionali

- Vincoli di tupla: esprimono condizioni sui valori di una singola tupla, indipendentemente dalle altre
- Vincoli di dominio: idem, ma coinvolgono un solo attributo

Chiave

insieme di attributi che identificano le tuple di una relazione

Formalmente diciamo che: un insieme K di attributi è superchiave per r se r non contiene 2 tuple distinte t_1, t_2 tali che $t_1[K] = t_2[K]$

K si dirà chiave se è una superchiave minima (Non contiene un'altra superchiave)

Poiché una relazione contiene tuple tutte distinte, ogni relazione ha almeno una chiave.

Una chiave con valori NULL perde di significato, definiamo quindi la chiave primaria: Chiave su cui non sono ammessi valori NULL (PK)

I vincoli di chiave fanno parte delle dipendenze funzionali (Ne parliamo dopo)

Interrelazionali

Informazioni di relazioni diverse si possono correlare attraverso valori comuni (quegli delle PK)
Tali correlazioni devono essere "coerenti".

Vincoli di integrità referenziale:

Un vincolo di integrità referenziale farà gli attributi X di R_1 e un'altra relazione R_2 imponga ai valori su X in R_1 di comparire come valori della (PK) di R_2

Nel caso di vincoli su più attributi l'ordine dei suddetti è significativo.

Si può rendere il vincolo meno restrittivo accettando valori NULL negli attributi di X ; in tal caso il vincolo sarà solo tra i valori di X diversi da NULL e la PK di R_2

Reazione alla violazione di un vincolo

• intervarazionale:

■ rifiuto dell'operazione

• interrelazionale:

- Se si producono violazioni sulla tabella interna a seguito di modifiche:

■ si rifiuta l'operazione

- Si può voler fare un'azione sulla tabella esterna ("master") come: eliminare una n-tupla o modificare l'attributo riferito.

In questo caso si possono attuare soluzioni differenti:

- Eliminazione/Modifica in cascata sulla tabella interna (cascade)

- Viene posto a NULL il valore dell'attributo referente (set null)

- Viene assegnato un valore default all'attributo referente (set default)

- La cancellazione non viene permessa (no action)

ALGEBRA E CALCOLO

Intervogazione: Operazioni di lettura sul DB (può richiedere l'accesso a più tabelle)

Abbiamo 2 semantiche principali:

• Operazionale:

- Si specificano le modalità di generazione del risultato
- Per SQL, definita con l'algebra relazionale

• Dichiavativa:

- Si specificano le proprietà del risultato
- Per SQL, si usa il Calcolo relazionale

La semantica di SQL è dichiavativa (indipendenza dai dati); Il metodo operazionale è il modo in cui il DBMS esegue le istruzioni SQL

DBMS

Il DBMS contiene un modulo specifico detto Query Processor (QP) al cui interno è definito il processo esecutivo delle operazioni.

Una parte del QP si occupa di ottimizzare la query prima dell'esecuzione (Ottimizzazione algebrica e poi basata sui costi)

Prima di parlarne definiamo il "profilo" di una relazione, che si usa nell'ottimizzazione sui costi. Contiene info riguardo:

- cardinalità delle relazioni
- dimensione delle tuple
- dimensione dei valori
- numero di valori distinti degli attributi
- valore min. e max. di ciascun attributo.

Ottimizzazione Algebrica:

basato su un metodo evристico e sulla nozione di equivalenza, ovvero:

Due espressioni si dicono equivalenti: se producono lo stesso risultato, qualunque sia l'istanza attuale del database.

Algebra relazionale

Operatori:

- Insiemistici: unione, intersezione, differenza
- Su relazioni:
 - ridenominazione (ρ)
 - selezione (σ)
 - proiezione (π)
 - Join (naturale, prodotto cartesiano, theta-Join, equi-Join)

Si possono applicare gli operatori insiemistici solo a relazioni definite sugli stessi attributi.

- Ridenominazione,

- operatore monadico

$$P_{y_1 \dots y_n} \leftarrow x_1 \dots x_n \quad (r)$$

- Selezione, (equivalente del where) \rightarrow decomposizione orizzontale

- operatore monadico

$\sigma_E(r)$ con E espressione booleana che determina l'etupla da selezionare

- Proiezione, (equivalente del select) \rightarrow decomposizione verticale.

- operatore monadico

$$\text{sia } y \subseteq x, \quad \pi_y(r(x))$$

Se y è superchiavi di r , la proiezione ha tante tuple quante ne ha r , altrimenti può averne meno.

- Prodotto cartesiano:

combina tutte le tuple di una relazione con quelle di un'altra

- Join Naturale: (\bowtie)

produce una relazione tale che:

- lo schema ha l'unione degli argomenti

• le tuple sono ottenute compiendo le tuple delle due relazioni per valori uguali degli attributi comuni

Rapporto con la proiezione:

$$\pi_{x_1}(R_1 \bowtie R_2) \leq R_1 \text{ con } R_1(x_1), R_2(x_2)$$

$$(\pi_{x_1}(R_1)) \bowtie (\pi_{x_2}(R_2)) \geq R, \text{ con } R(x), x = x_1 \cup x_2$$

Se tentiamo un join naturale tra relazioni senza attributi in comune, otteniamo il prodotto cartesiano

- Theta-Join

$$R_1 \bowtie_F R_2, \text{ con } F \text{ condizione di join.}$$

Se F contiene solo l'operatore " $=$ ", parliamo di equi-join

Equivalenza di espressioni:

due espressioni sono equivalenti; se producono lo stesso risultato indipendentemente dall'istanza attuale del DB.

Equivalenti importanti:

- pushing selections down (fare le selezioni il prima possibile)

- pushing projections down

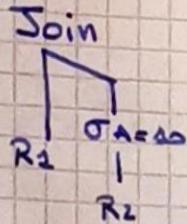
entrambe riducono notevolmente il costo dell'operazione.

Internamente, le interrogazioni sono rappresentate come alberi in cui:

- le foglie sono dati;
- i nodi interni sono operatori.

es:

$$R_1 \text{ Join } \sigma_{A=40} (R_2) \Rightarrow$$



Procedura evistica di ottimizzazione:

- 1) Selezioni congiuntive \rightarrow selezioni atomiche
- 2) pushing selections down
- 3) tra le selezioni, far prima le più selettive
- 4) formare join combinando selezioni e prodotti cartesiani
- 5) pushing projections down (inserire anche di nuovo se d'aiuto)

Calcolo Relazionale

è una famiglia di linguaggi dichiarativi basati sul calcolo dei predicati del primo ordine.

2 versioni principali:

- calcolo su domini
- calcolo su esempi

Calcolo su domini

$$\{ A_1 : x_1, \dots, A_n : x_n \mid f \} \text{ con:}$$

- A_i = nome Attributo
- x_i = nome variabile
- f = formula

es: Trovare impiegati con stipendio > 40

$$\{ Matricole:m, Nome:n, Età:e, Stipendio:s \mid$$

Impiegati: (Matricole:m, Nome:n, Età:e, Stipendio:s)

$$\wedge s > 40 \}$$

and

Calcolo su tuple

$\{ \underbrace{x_1, x_2, \dots, x_n}_{\text{Target-list}} | \underbrace{x_i(R_1), \dots, x_j(R_m)}_{\text{range list}} | f \}$

Esempio:

$\{ i.* | i(\text{Impiegati}) \text{ } i.\text{stipendio} > 40 \}$

Nel calcolo relazionale si possono anche utilizzare i quantificatori esistenziali (\exists) ed universali (\forall).

Non sono obbligatori ma in alcune situazioni risultano necessari.

Nel calcolo sui domini, al contrario che in algebra, è possibile scrivere espressioni senza senso poiché esse sono dipendenti dal dominio, mentre non lo sono in algebra relazionale.

Nel calcolo sui domini (non si può esprimere l'unione, poiché ogni variabile nel risultato ha un solo range).

Calcolo e algebra sono "equivalenti":

- per ogni operazione di calcolo indipendente dal dominio esiste un'equivalente in algebra
- per ogni operazione in algebra esiste un'equivalente in calcolo.

Calcolo ed algebra non possono esprimere ogni tipo di operazione; ad esempio non è esprimibile la chiusura transitiva.

Questo poiché tale operazione potrebbe risultare in un'operazione con join infiniti.

Estensioni all'algebra:

- Join esterno ($=\Delta$, $\Delta =$, $=\Delta =$)

Tutte le tuple dell'operando affiancate al simbolo " $=$ " vengono mantenute e, più dove non sia fatto il join vengono riempite con Null.

- Proiezione generalizzata:

$\Pi_{F_1, \dots, F_n}(E)$, con F_1, \dots, F_n espressioni aritmetiche su attributi di E e costanti.

Esempio:

$\Pi_{\text{cliente}, \text{credito-spese}}(\text{Conto}) \Rightarrow$

| cliente | credito-spese |
|---------|---------------|
| ⋮ | ⋮ |

Operatori aggregati:

- $\text{Sum}_{\text{att}}(\epsilon)$
- $\text{Count}_{\text{att}}(\epsilon)$
- $\text{Count-distinct}_{\text{att}}(\epsilon)$
- $\text{Max}_{\text{att}}(\epsilon)$

Raggruppamento:

$\text{att}^G \text{sum}_{\text{att}}(\epsilon)$ att = attributo raggruppamento. (possono essere molti pluri)

Divisione

Utile per le interrogazioni di tipo "universale".

Siano $r(R)$ ed $s(S)$ relazioni con $R \sqsupseteq S$, $r \div s$ è una relazione su $R - S$ e le tuple t che vi appartengono sono tali che:

- $t \in \Pi_{R-S}(r)$
- $\forall t' \in s, \exists t'' \in r :$
 - $t'[S] = t''[S]$
 - $t''[R-S] = t$

Esempio:

Clienti che hanno un conto in ogni filiale di Disa

$\Pi_{C, BrN}(\text{deposito} \bowtie \text{account}) \div \Pi_{BrN}(\sigma_{Br=Disa}(\text{branch}))$

Relazioni derivate:

Sono relazioni il cui contenuto è funzione del contenuto di altre relazioni (ed è definito per mezzo di interrogazioni).

Ne esistono 2 tipi:

- Viste materializzate
- Viste virtuali (o solo viste)

• Materialized view:

- Sempre disponibili, ma

- ridondanti:

- appesantiscono gli aggiornamenti

• Viste (virtuali):

- un'interrogazione su una vista viene eseguita "ricalcolando" la vista ogni volta.
All'atto della chiamata, al nome della vista si sostituisce la sua definizione; questo rende più semplice scrivere codice ma non ne cambia la complessità.

Progettazione di un DB

Lo sviluppo di sistemi SAV in generale ha un suo ciclo di vita:

- 1) Raccolta e analisi dei requisiti
- 2) progettazione
- 3) Realizzazione
- 4) Validazione e collaudo
- 5) Funzionamento

Concentriamoci sulla progettazione, che include:

- progettazione dei dati
- progettazione delle applicazioni

Per la realizzazione di un buon progetto necessitiamo di un linguaggio/modello per descriverlo:

Per descrivere il ciclo di vita vi sono molti modelli; il più vecchio è il waterfall. Esso consta di fasi uniche e non ripetibili:

1) Acquisizione ed analisi dei requisiti.

1.1) Acquisizione: tramite interviste, documentazione apposita, normative, precedenti realizzazioni, etc. Non è un'attività standardizzabile.

1.2) Analisi: inizia con le prime acquisizioni e spesso indirizza verso altre. Si usano linguaggi per definire i requisiti, come ad esempio UML

Nell'intervento con l'utente è consigliato:

- Standardizzare la struttura delle frasi;
- Separare le frasi sui dati da quelle sulle funzioni;
- Costruire un glossario dei termini:
 - unificare i termini (individuare i sinonimi)
 - rendere esplicito il riferimento tra termini
- Riorganizzare le frasi per concetti.

2) Progettazione

Si vuole progettare per livelli d'astrazione, ovvero:

- Livello concettuale: requisiti espressi in una descrizione adatta all'analisi esterna
- Livello logico: Evidenzia l'organizzazione dei dati dal punto di vista del loro contenuto descrivendo la struttura di ciascun record ed i collegamenti tra di essi.

- Livello Fisico: Il DB è ora visto come un insieme di blocchi fisici su disco. Si decidono allocazione dei dati e modalità di memorizzazione.

Ognuna di queste fasi genera un diverso schema della base di dati, ognuno con il proprio linguaggio.

Modello Concettuale:

I dati sono rappresentati in modo più vicino al modo di pensare umano, prevedendo anche rappresentazioni grafiche.

I requisiti raccolti sono specificati in modo:

- Formale: non-ambiguo
- Integro: la descrizione si riferisce alla totalità dell'ambiente.

Il modello E-R: (1976)

- Costrutti base:

- Entità
- Relazione
- Attributo

- Altri costrutti:

- Identificatore
- Generalizzazione
- (...)

Entità

Classe di oggetti dell'applicazione con proprietà comuni ed esistenza autonoma.

Un'occorrenza di un'entità è un elemento di tale classe (l'oggetto, non un valore dei dati ad essa legato).

Ogni entità ha un nome che l'identifica univocamente (meglio se singolare) significativo.

Relazione

Legame logico rilevante fra due o più entità.

È rappresentata univocamente da un nome (preferibilmente un sostantivo singolare).

Un'occorrenza di relazione è un t-upla di occorrenze di entità, una per ciascuna delle t entità coinvolte.

Attributi

Proprietà elementare di un'entità o di una relazione. Associasi ogni occorrenza di entità/relazione un valore appartenente al dominio dell'attributo.

- Attributi composti:

Raggruppamento di attributi di un'entità/relazione che fanno affinità nel significato o nell'uso.

Cardinalità di relazione

Coppia di valori associata ad ogni entità che partecipa ad una relazione; specificano il numero minimo (0/1) e massimo (1/N) di occorrenze della relazione cui ciascuna occorrenza di entità può partecipare.

Si suddividono, con riferimento alle cardinalità massime, in:

- uno a uno
- uno a molti
- molti a molti

Si può associare una cardinalità anche agli attributi, per specificare:

- opzionalità
- attributi multivalue.

Identificatore di una entità

Identifica univocamente le occorrenze di un'entità, costituito da:

- attributi dell'entità
 - identificatore interno (o chiave)
- (attributi +) chiave di entità esterna raggiunta tramite relationship.
 - identificatore esterno.

Ogni entità ha almeno un identificatore (anche più di uno), un'identificazione esterna è possibile solo attraverso una relazione in cui l'entità da identificare partecipa con cardinalità (1,1)

Generalizzazione

Mette in relazione una o più entità E_1, \dots, E_n con un'entità E che le comprende come casi particolari.

E si dice generalizzazione; E_1, \dots, E_n si dicono specializzazioni.

- Ogni proprietà di E è significativa per E_1, \dots, E_n
- Ogni occorrenza di E_1, \dots, E_n è occorrenza anche di E

Le proprietà dell'entità genitrice sono ereditate (e non rappresentate esplicitamente) dalle entità figlie.

Si fanno due tipi di generalizzazione:

Totale:

Ogni occorrenza del genitore è
occorrenza almeno una figlia.
(altrimenti parziale)

Esclusiva:

Ogni occorrenza del genitore è al più
occorrente di una figlia. (Altrimenti soprapposta)

È sempre possibile passare da generalizzazione soprapposta ad esclusiva.

Lo schema E-R da solo non è quasi mai sufficiente a rappresentare tutti i dettagli di un'applicazione, si associa quindi una documentazione di supporto.

Documentazione

- Dizionario dei dati
 - Entità
 - Relazioni
- Regole aziendali
 - Vincoli d'integrità
 - Possibili derivazioni

Strategie di progettazione:

Top-down:

Si parte da uno schema iniziale che viene successivamente raffinato e integrato per mezzo di primitive che lo modificano fino ad ottenere lo schema E-R finale.

Bottom-up:

Si suddividono le specifiche fino ad ottenere una componente minima di cui si dà l'E-R. Tali schemi vengono fusi ed integrati fino ad ottenere lo schema finale.

Soltanamente si utilizza un approccio "misto".

VINCOLI

Clausola check

La clausola check permette di restringere i domini e specificare predicati che devono essere soddisfatti ogni volta che un valore viene assegnato ad una variabile in quel dominio. Vale al livello di tabella e si esprime con "check (condizione)".

Esempio:

:
Sesso char Not NULL check (Sesso in ('M', 'F'))
:

Asseverazioni

Specifica vincoli al livello di schema del DB (cioè tra più tabelle); il vincolo è allo stesso livello della definizione delle tabelle.

Si specifica con:

Create assertion "Name"
check (condizione)

ad ogni assezione è associata una politica di controllo, che può essere:

- Immediato (dopo ogni modifica) (rollback parziale)
 - Differito (dopo una transazione) (rollback)
- Lo si determina con il costrutto:
`set constraints [Name] [immediate | deferrable]`

Trigger

Sono regole attive che si basano sul paradigma Evento - Condizione - Azione

Evento

- Normalmente un insert / update / delete
- Quando accade, il trigger è attivato

Condizione

- Predicato che determina se l'azione del trigger deve essere eseguita
- Il trigger è considerato

Azione

- Sequenza di update SQL o una procedura
- Il trigger è eseguito.

Ogni trigger è caratterizzato da:

- nome
- target (tabella controllata)
- modalità (before / after)
- evento (insert / update / delete)
- granularità (statement-level / row-level)
- alias dei valori
- Azione
- timestamp di creazione

Before

- trigger considerato e possibilmente eseguito prima dell'evento
- può solo modificare i valori "new" in modalità row-level
- Si usa per verificare (ed eventualmente modificare) una modifica al DB prima che avvenga.

After

- trigger considerato ed eseguito dopo l'evento

• Statement-level (for each statement),

il trigger viene considerato e possibilmente eseguito solo una volta per ogni statement, indipendentemente dal numero di tuple coinvolte.

• row-level,

trigger considerato e possibilmente eseguito una volta per ogni tupla modificata.

• referencing:

- row-level: si hanno due variabili (old, new) che rappresentano il valore precedente e successivo alla modifica di una tupla

- statement-level: si hanno due tabella (old table / new table) che contengono i valori precedenti e successivi delle tuple modificate dallo statement.

old / old table non presenti: con insert

new / new table non presenti: con delete.

In caso di conflitto fra trigger:

- prima i before (in ordine di T.timestamp)
- modifica e verifica vincoli d'integrità
- after triggers (in ordine di Timestamp)

Privilegi d'accesso:

E' possibile specificare chi e come può utilizzare il DB (o parte di esso)

"System" ha tutti i privilegi; il creatore di una risorsa ha tutti i privilegi su di essa.

Ogni privilegio è caratterizzato da:

- | | |
|---------------------|-------------------|
| - risorsa | - azione permessa |
| - utente concedente | - trasmissibilità |
| - utente ricevente | |

In SQL i tipi di privilegio sono

- | | |
|----------|--|
| - insert | - select |
| - update | - references (permette di definire vincoli d'integrità verso la risorsa) |
| - delete | - usage (permette l'utilizzo in una definizione) |

Si utilizzano i comandi grant e revoke per concedere/togliere privilegi.

con la clausola "with grant option" si permette all'utente di trasmettere il privilegio

Attraverso le viste si può concedere una visione "parziale"

Procedure

Si possono creare Stored procedures, esse fanno parte dello schema e lo standard prevede che abbiano funzionalità limitate, composta da un solo comando SQL

SQL e linguaggi

Al fine di gestire realtà più complesse, è possibile scrivere applicazioni contenenti anche comandi SQL (in C, Java, PL/SQL, Delphi, ...)

Tecniche principali:

- SQL immerso (SQL "statico")
- SQL dinamico
- Call level interface (CLI)

SQL immerso:

Le istruzioni SQL sono "immense" nel programma redatto nel linguaggio "ospite"

Un precompilatore (legato al DBMS) analizza il programma e sostituisce i comandi SQL con chiamate alle API del DBMS

Ogni coppia linguaggio - s.o. - DBMS ha il suo precompilatore.

- EXEC SQL denota le porzioni di interesse del precompilatore.
- Le variabili del programma, precedute da ":" possono essere utilizzate come parametri delle istruzioni SQL.
- Sqlda è la struttura di comunicazione con il DBMS
- sqrcode = 0 se connessione okay, altrimenti errore.

Un problema si riscontra nell'impedimente mismatch:

I linguaggi lavorano su singoli variabili/oggetti, SQL fa operazioni su tabelle e restituisce tabelle.

Attraverso i cursori (cursori) si può trasmettere al programma una tupla alla volta.

⚠️ usare troppi cursori (specie semidificati) pregiudica la capacità ottimizzatrice del DBMS

SQL Dinamico

Permette di eseguire istruzioni SQL costruite dal programma (o ricevute in input!)

Tali operazioni possono essere eseguite immediatamente (execute immediate SQLSTATEMENT) oppure prima preparate (prepare COMMANDNAME from SQLSTATEMENT) e poi eseguite tante volte quante si desidera.

CL1

indica interface che permettono di inviare al DBMS richieste per mezzo di parametri, trasmessi a funzioni.

Indipendente dal DBMS, permette l'accesso ADPS eterogenei.

DA MODELLO CONCETTUALE
A SCHEMA LOGICO

Vogliamo tradurre lo schema concettuale, insieme alle info sul carico applicativo, in uno schema logico.

Non è immediato: Bisogna semplificare la traduzione e tenere conto dei requisiti di prestazione.

Ristrutturazione:

1) eliminazione delle generalizzazioni:

a) accorpamento delle figlie nella genitrice:

conviene se gli accessi a madre e figlie sono contestuali

b) accorpamento della genitrice nelle figlie

conviene se gli accessi sono solo alle figlie e sono distinti dall'una all'altra

c) sostituzione della generalizzazione con relazioni:

conviene se si effettuano accessi separati alle figlie e alla madre.

2) Eliminazione attributi multivalore.

Si può:

• ripetere le tuple con valore diverso nell'attributo

• dimensionare l'attributo al massimo numero di attributi possibile

• creare una relazione apposita per l'attributo.

} Spezza memoria

3) Analisi ed eventuale eliminazione delle ridondanze

Una ridondanza è un'informazione significativa, ma derivabile da altre.

Derivabile da:

- altri attributi della stessa entità

- attributi di altre entità

- associazioni derivabili dalla composizione di altre associazioni (presenta di cicli)

Si decide se mantenere una ridondanza in base ad una valutazione del costo delle operazioni

Per farci consideriamo degli indicatori:

- spazio: numero di occorrenze previste

- tempo: numero occorrente visite per portare a termine un'operazione

Abbiamo quindi una tanda dei volumi ed una degli accessi.

In base ad esse consideriamo il numero necessari di accessi per le operazioni con e senza ridondanza; sceglieremo in base a questo se mantenere o meno la ridondanza.

4) partizionamento / accoppiamento di entità e relazioni

Al fine di rendere le operazioni più efficienti, secondo il principio che:

- gli accessi si riducono:

- separando attributi di un concetto che vengono accediti separatamente;
- unendo attributi di ~~un~~ concetto a diversi accediti insieme.

a) partizionamento di entità

b) accoppiamento entità - relazioni

c) partizionamento relazioni

Traduzione verso il modello relazionale,

- le entità diventano relazioni sugli stessi attributi

- le relazioni diventano entità sugli identificatori delle entità coinvolte (+ i propri)

Nel caso di relazioni con cardinalità (1,1) si può accoppare all'entità

QUALITÀ DELLE
RELAZIONI

Dipendenze funzionali (FD)

Esprime un legame semantico tra due gruppi di attributi di uno schema di relazione R

Una FD è una proprietà di R, deve essere definita esplicitamente.

Forma Normale

Ne esistono di vari tipi, ognuna garantisce l'assenza di determinati difetti in R.

Attraverso la normalizzazione si può portare lo schema R in una determinata forma normale.

Le FD in dettaglio

Sia R relazione su $R(x)$

y, z due sottoinsiemi non vuoti di x

esiste una FD in R da y a z se, $\forall t_1, t_2$ di R con valori uguali su y , t_1 e t_2 hanno valori uguali anche su z . Si dirà:

$y \rightarrow z$.

Ad ogni chiave K di R , corrisponde una FD $K \rightarrow x$

$y \rightarrow A$ si dice non banale se A non appartiene a y

$y \rightarrow z$ " " se nessun attributo in z appartiene ad y

Teoria delle dipendenze

Sia F un insieme di FD definite su $R(z)$ e sia $X \rightarrow Y$:

- si dice che F implica $X \rightarrow Y$ ($F \Rightarrow X \rightarrow Y$) se, per ogni istanza di R che verifica F , risulta verificata anche $X \rightarrow Y$.
- si dice chiusura di F (F^+) l'insieme di tutte le dipendenze funzionali implicite da F . Un'istanza di R che verifica F , verifica anche F^+

Superchiavi

Dato $R(z)$ e un insieme F di FD, un insieme di attributi $K \subseteq z$ si dice superchiave di R se la FD $K \rightarrow z$ è implicata da F ($K \rightarrow z$ è in F^+)

Se nessun sottoinsieme proprio di K è superchiave di R , K si dice chiave di R .

Per il calcolo di F^+ si utilizzano le regole di Armstrong, che sono corrette e complete.

Regole di Armstrong

1) Riflessività: se $Y \subseteq X \Rightarrow X \rightarrow Y$

2) Addittività: Se $X \rightarrow Y \Rightarrow XZ \rightarrow YZ \forall Z$

3) Transitività: Se $X \rightarrow Y, Y \rightarrow Z \Rightarrow X \rightarrow Z$

• correttezza: applicandole ad un insieme F di FD, si ottengono solo le dipendenze logicamente implicite da F

• completezza: applicandole ad F si ottengono tutte le dipendenze logicamente implicite da F

• minimalità: ignorando anche solo una regola di Armstrong, quelle che rimangono non sono più complete.

Regole derivate:

4) Regola di Unione:

$$\{x \rightarrow y, x \rightarrow z\} \Rightarrow x \rightarrow yz$$

5) Regola di pseudotransitività

$$\{x \rightarrow y, wy \rightarrow z\} \Rightarrow wx \rightarrow z$$

6) Regola di decomposizione:

$$z \subseteq y, x \rightarrow y \Rightarrow x \rightarrow z$$

Equivalenza:

Dati due insiemi di FD F e G , essi sono equivalenti se $F^+ = G^+$, ovvero $\forall x \rightarrow y \in F^+$, $x \rightarrow y \in G$ e viceversa.

Il calcolo di F^+ è molto costoso, esposto ci interessa solo verificare se F^+ contiene una certa dipendenza. Come alternativa si può calcolare la chiusura di un insieme di attributi X ; è dimostrato che $x \rightarrow y$ è in F^+ se $y \subseteq X^+$.

X^+ rappresenta l'insieme degli attributi che dipendono da X secondo F .

Esempio:

$$F = \{A \rightarrow B, B \rightarrow E, BC \rightarrow D, E \rightarrow C\}$$

$$A^+ = B (A \rightarrow B), E (B \rightarrow E), C (E \rightarrow C), D (BC \rightarrow D), A$$

Da 2 dipendono tutti gli attributi, quindi A è superchiave (ed anche chiave).

Possiamo quindi dire che F e G sono equivalenti se:

$$\forall x \rightarrow y \in F, y \subseteq X^+ \text{ in } G \text{ e}$$

$$\forall z \rightarrow w \in G, w \subseteq Z^+ \text{ in } F$$

La chiusura di un insieme di attributi si può utilizzare per verificare se:

- Una FD è implicata da F
- Un insieme di attributi è superchiave o chiave.

Ogni insieme F può essere portato in forma standard eliminando le ridondanze a destra e sinistra, nonché le dipendenze ridondanti.

Attributi estranei

In una FD $AX \rightarrow B$, A è estraneo se $X^+ \not\subseteq B$, ovvero se X da solo determina B .

FD Ridondanti:

Data $X \rightarrow A$ essa è ridondante se, eliminata da F , $X^+ \ni A$ comunque (si deduce $X \rightarrow A$ dalle altre).

Copertura minima:

Una copertura minima è un insieme di FD tale che:

- Tutte le FD sono semplici
- Non ci sono attributi estranei
- Non ci sono FD ridondanti.

Forme Normali e Normalizzazione

BCNF

Una relazione r è in BCNF se, per ogni FD non banale $X \rightarrow Y$, X è superchiave di r . Tutte le proprietà devono quindi essere direttamente associate alla chiave.

Teorema:

Se F non contiene alcuna $X \rightarrow Y$ non banale con X non superchiave di R , allora nemmeno F la contiene.

Per verificare BCNF:

- Si minimizza F , si verifica che i membri a sinistra siano solo superchiavi.

Se R non è in BCNF, possiamo decomporla in più relazioni in BCNF sulla base delle FD.

Si possono però generare problemi.

Decomposizione senza perdita:

$r \in R$ si decomponga senza perdita su X_1, X_2 se il join naturale delle proiezioni di r su X_2 ed X_1 è uguale ad r .

Algoritmo decomposizione: $U = \text{tot. att.}$ (Varia a seconda dell'ordine di valutazione delle FD)

$\left\{ \begin{array}{l} \text{if } (\exists X \rightarrow A \text{ EF : } X \text{ non superchiave } R) \\ \quad \left\{ \begin{array}{l} \text{Sostituisce } R \text{ con } R_1 \text{ con att. } U-A, R_2 \text{ con att. } X \cup A \end{array} \right\} \end{array} \right.$

Decomponi (R_1, F_{U-A})

Decomponi ($R_2, F_{X \cup A}$)

}

Tale algoritmo genera una decomposizione tale che R_1, \dots, R_n sono in BCNF e non si ha perde sul join.

La proiezione di F su $X \subseteq U$ (F_X) è l'insieme di FD $Z \rightarrow Y$ in F^+ che coinvolgono solo attributi di X .

Conservazione delle dipendenze:

Una decomposizione conserva le di dipendenze se ciascuna di esse coinvolge attributi che compaiono tutti insieme in uno degli schemi decomposti.

Sia X un sottoinsieme degli attributi di R :

La decomposizione di R in due tabelle con attributi X ed Y e' senza perdita di dipendenze se $(F_X \cup F_Y)^+ = F^+$.

Terza Forma Normale

R è in 3FN se per ogni FD (Non banale) $X \rightarrow Y$:

- X è superchiave di R
- Y è contenuto in almeno una chiave di R .

Può generare anomalie (ridondante...)

La verifica di 3NF è un problema NP-completo.

3NF è sempre raggiungibile senza perdite e conservando le dipendenze.

Decomposizione:

Si decompone in BCNF; sia N l'insieme di FD non presentate, $\forall X \rightarrow A \in N$, aggiungiamolo schema $X \rightarrow A$ con le FD relative ad $X \rightarrow A$

2° metodo

Esempio:

1) $R(ABCDEF)$, FD: $A \rightarrow C$, $B \rightarrow E$, $C \rightarrow F$, $F \rightarrow G$

↓

$R_1(ABCDE)$, $R_2(CF)$, $R_3(FG)$ (Anche BCNF)

2) $R(ABCD)$, FD: $A \rightarrow BC$, $B \rightarrow A$, $C \rightarrow D$

↓

$R_1(ABC)$ $R_2(CD)$ (No BCNF)
chiave o A o B.

Il 1° metodo garantisce prima il join e poi conserva le dipendenze, il secondo fa l'opposto.

Se dopo il 2° metodo le relazioni hanno solo una chiave, lo schema è anche BCNF

Altre forme normali

Se in R c'è più di una chiave, ognuna di esse è detta chiave candidata; una verrà nominata **principale**.
Se un'attributo fa parte di una chiave candidata è detto **primo**, altrimenti **non-primo**.
Una dipendenza $X \rightarrow Y$ è detta completa se rimuovendo qualsiasi attributo da X essa non sussiste più. In caso contrario è detta **parziale**.

1NF

Il dominio degli attributi deve essere composto solo di valori atomici ed il valore di qualsiasi attributo in una tupla sia un valore singolo del dominio.

2NF

R è in 2NF se ogni attributo non primo di R è funzionalmente dipendente in modo completo da ogni chiave di R.

Possono esistere dipendenze tra attributi non primi.

GESTIONE DELLE TRANSAZIONI

I DBMS possono essere monostanti o multistanti, anche se la maggior parte è multistante.

In questo secondo caso vi saranno più accessi da gestire.

Transazione:

Identifica un'unità elementare di lavoro svolta da un'applicazione, cui si vogliono associare particolari caratteristiche di correttezza, robustezza ed isolamento

oppure:

Un programma in esecuzione che forma un'unità logica di collaborazione sul DB

oppure:

un insieme di operazioni da considerare indivisibile, corretto anche in presenza di concorrenza e con effetti definitivi.

Un sistema che mette a disposizione un meccanismo per la definizione e l'esecuzione di transazioni è detto **transazionale**.

Una transazione è caratterizzata da un inizio (`begin-transaction`) una fine (`end-transaction`) e al cui interno deve essere eseguito una e una sola volta o "commit work" oppure "rollback work".

Le proprietà di una transazione sono:

- Atomicità
 - Consistenza
 - Isolamento
 - Durata
- } "ACID"

Atomicità, (gestore dell'affidabilità)

Una transazione non può lasciare la base di dati in uno stato intermedio:

questo pre-commit → annullamento operazioni scritte

questo post-commit → (se necessario) ripetizione operazioni scritte.

I guasti sono molti vari.

Consistenza, (gestore integrità a tempo di esecuzione)

Vengono rispettati i vincoli d'integrità. Se lo stato iniziale era corretto, lo stato finale deve essere corretto.

All'inizio e alla fine il DB è in stato consistente, durante l'esecuzione questo non è garantito.

Isolamento, (gestore della concorrenza)

La transazione non risente degli effetti delle altre esecuzioni concorrenti.

Perciò una transazione non espone i suoi stati intermedi.

Durabilità, (gestore dell'affidabilità)

Gli effetti di una transazione andata in commit non vanno perduti.

Una transazione è sempre in uno dei seguenti stati:

- active: dopo il begin-transaction, posso fare operazioni R/W
- partially-committed: dopo end-transaction; il controllore dell'affidabilità garantisce che i cambiamenti effettuati verranno memorizzati anche in caso di errore di sistema. Poi si passa a
- committed
- failed: stato raggiunto dopo aver determinato che l'esecuzione non può procedere normalmente.
- aborted: vi è stato un rollback ed il DB è stato riportato allo stato pre-transazione.

I dispositivi di memoria secondaria sono organizzati in blocchi di lunghezza fissa, le uniche operazioni sono la lettura e la scrittura di un blocco.

La memoria principale è organizzata in pagine.

• Buffer

Poiché i programmi possono accedere solo alla memoria principale, serve un'intervazione fra memoria principale e secondaria che limiti il più possibile gli accessi alla secondaria.

Il Buffer è un'area di memoria centrale, condivisa tra le transazioni, organizzata in pagine di dimensione uguale o multiplo di quella dei blocchi di memoria secondaria.

Esso è gestito dal Buffer-Manager che:

- Riceve le richieste di lettura/scrittura
- Le esegue usando il Buffer o accedendo se necessario alla memoria secondaria.

Le pagine vengono mantenute fin quando il Buffer non è pieno e non si possono inserire altre pagine (Pagine 85-20)

• Interfaccia

esegue le primitive:

- fix: richiesta di una pagina
- setDirty: comunica un'avvenuta modifica alla Pagina.
- unfix: la transazione ha concluso l'utilizzo della pagina.
- force: trasferisce in modo sincrono una pagina in memoria secondaria.

Gestione dell'affidabilità

Abbiamo 3 tipi fondamentali di malfunzionamenti:

- del disco (rottura della testina/errore nel trasferimento dati)
- di alimentazione (info. in memoria centrale e registri periferici)
- del software (errori logici e disistema)

Il gestore dell'affidabilità gestisce:

- esecuzione dei comandi transazionali
- recovery dopo i guasti:
 - warm/cold restart
- Assicura atomicità e durata
- Usa il log, ovvero:

Un archivio permanente che registra le operazioni svolte.

Ai fini del recupero, la memoria si divide in:

- volatile: persa in caso di caduta di sistema (non persistente)
- non-volatile, resistente a cadute di sistema ma comunque vulnerabile (persistente)
- stabile: invulnerabile (virtualmente)

Log

È un file sequenziale gestito dal controllore dell'affidabilità, scritto in memoria a stabile.

Riporta tutte le operazioni in ordine

Serve "a ricostruire" le operazioni. Si trovano dei record di sistema (Checkpoint e Dump) che servono a farsi che la ricostruzione debba partire dall'inizio dei tempi.

I record nel log sono o di sistema o di transazione.

Checkpoint

Serve a fare "il check" della situazione, in coordinamento con il gestore del buffer.

Registra quali transazioni sono attive in un dato istante, e dunque conferma che le altre o non sono iniziate, o sono finite. (Le transazioni che hanno fatto commit trasferiscono i dati in memoria di massa).

Nell'esecuzione del Checkpoint:

- si sospende l'accettazione delle operazioni di commit, abort.
- forza della scrittura in memoria di massa delle pagine del buffer modificate da chi ha fatto commit.
- forza della scrittura sul log di un record contenente gli id delle transazioni attive.
- ripresa normale esecuzione.

Dump

Copia completa ("di riserva") del DB:

- prodotta mentre il sistema non è operativo
- salvato in memoria stabile.
- Un record di Dump nel log indica il momento in cui il dump è stato effettuato.

Undo e Redo sono operazioni idempotenti.

Modifiche al Log

• Write-ahead-log:

- si scrive la parte BS del record del log prima di effettuare l'operazione sul DB:
permette di disfare operazioni di transazioni senza commit avendo un valore corretto in memoria stabile.

• Commit-Precedenza:

- si scrive la parte AS del record di log prima del commit:
permette di eseguire il redo di una transazione che ha fatto commit ma le cui pagine non sono state ancora trascritte in memoria di massa

Nella pratica si usano delle regole semplificate, in cui BS ed AS sono scritti insieme:

- WAL: record di log scritti prima dei record del DB
- Commit-precedenza: log scritto prima dell'operazione di commit.

Esito di una transazione

è determinato quando viene scritto il record di commit nel log:

- un guasto prima del commit porta ad un undo di tutte le azioni
- un guasto dopo non deve avere conseguenze: se necessario si effettuano i redo.

Scritture nel DB

immediata:

- Il DB contiene valori AS di transazioni uncommitted
- Richiede undo delle operazioni di transazioni uncommitted al momento del guasto.
- Non richiede redo

Differita:

- Il DB non contiene valori AS di transazioni uncommitted
- Undo non serve, non essendoci scritture prima del commit
- Richiede Redo

Mista:

- La scrittura avviene sia in modalità immediata che differita
- Richiede sia Undo che Redo

La si preferisce perché lascia più libertà di scrittura al Buffer anche se gestisce "peggiori" gli errori: perché ~~tutti~~ sono molti vari.

Il commit si considera effettuato quando viene scritto nel log il corrispondente record.

Il Rollback cancella tutte le operazioni tra begin e abort, dopodiché scrive un record di Abort nel log.

- Guasti "soft": si perde la memoria centrale (quindi anche i buffer) ma non la secondaria.
 - ripresa a caldo.
- Guasti "Hard": si perde anche la memoria secondaria (parte del DB)
 - ripresa a freddo.

Modello Fail-Stop

- 1) L'individuazione del guasto forza l'avresto totale delle transazioni.
- 2) Il S.O. viene riavviato
- 3) si avvia una ripresa
- 4) Al termine della ripresa le transazioni ripartono, ma il buffer è vuoto.

Ripresa a Caldo

- Trovare l'ultimo checkpoint.
- insieme UNDO: transazioni attive ma non committed
- insieme REDO: transazioni attive ma committed
- Ripercorrere il log all'indietro disfacendo tutte le azioni delle transazioni in UNDO
- Ripercorrere in avanti facendo i REDO

Le transazioni attive da inserire nel checkpoint si possono fissare:

- rifiutando nuovi commit (si usa in Pratica)
- rifiutando nuovi begin ed aspettando i commit delle transazioni iniziate.

Le operazioni derivanti dal rollback possono essere eseguite durante il recovery (UNDO) oppure essere eseguite al momento dell'abort ed essere poi inserite nel REDO durante il recovery.

Ripresa a Freddo:

- Ci si porta al più recente dump e si ripristina la parte di dati deteriorata
- si eseguono le operazioni sul log in avanti alla parte deteriorata fino al guasto
- si effettua una ripresa a caldo.

CONTROLLO DELLA CONCERNZA

Anomalie:

- Perdita di aggiornamento (W-W)
- Lettura sporca (R-W / W-W con abort)
- Lettura inconsistente (R-W)
- Aggiornamento fantasma (R-W)
- Inserimento fantasma (R-W)

Schedule:

Una sequenza di R/W relativa all'insieme delle transazioni concorrenti in un certo istante. Le operazioni compaiono in ordine temporale di esecuzione sul DB.

Lo scheduler esegue il controllo della concorrenza, decide quindi se accettare o rifiutare le operazioni via via richieste.

Assumiamo che il commit/abort sia noto a priori:

- possiamo rimuovere le transazioni abortite dalla schedule
- non possiamo trattare alcuna anomalia (lettura sporce)

Schedule Seziale:

Le operazioni di ogni transazione vengono svolte in sequenza

Schedule serializzabile:

Uno schedule S è equivalente ad uno schedule seziale S'

VSR

Uno schedule è viewserializzabile se è view-equivalente ad uno schedule seziale, ovvero se hanno la stessa relazione "legge-da" e le stesse scritture finali su ogni oggetto.

Decidere la viewserializzabilità di uno schedule è un problema NP-completo, non posso usare questa definizione nella pratica.

Conflitti:

Due operazioni sono in conflitto se operano sullo stesso oggetto e almeno una di esse è una scrittura.

CSR

2 schedule sono conflict-equivalenti se hanno le stesse operazioni: ogni coppia di operazioni in conflitto compare nello stesso ordine in entrambi.

Uno schedule è conflict-serializzabile se è conflict-equi. ad uno schedule seziale.

$CSR \Rightarrow VSR$

$VSR \not\Rightarrow CSR$

Uno schedule è CSR se il suo grafo dei conflitti è aciclico.

il grafo si costruisce inserendo un arco orientato da t_i a t_j se un'operazione di t_i è in conflitto con una di t_j e deve precederla.

Anche questo non è praticabile, sia per come andrebbe aggiornato sia per l'esistenza di basi di dati distribuite.

In pratica si utilizzano tecniche di scheduling che garantiscono a priori CSR

Lock

Tutte le letture / scritture sono precedute da un lock e seguite da un unlock

Per aumentare la concorrenza è possibile avere lock condiviso sulle letture ed un lock esclusivo sulle scritture.

Tutte le letture sono precedute da r-lock e seguite da unlock

" " Scritture sono precedute da w-lock e seguite da unlock

Se prima si legge e poi si scrive si può richiedere subito un lock esclusivo oppure fare l'upgrade al momento della scrittura)

Il lock manager riceve le richieste di lock e concede o meno il lock in base ai lock precedentemente concessi.

Se la risorsa non è concessa, la transazione richiedente è posta in attesa fin quando la risorsa non diventa disponibile.

2PL

Locking a 2 fasi:

Garantisce a priori CSR. (Non è vero il contrario)

Utilizza i lock su letture e scritture, inoltre una transazione, dopo aver rilasciato un lock, non può acquisirne altri finché non ha rilasciato tutti quelli precedentemente acquisiti.

(crescente)

(calante)

Abbiamo quindi una prima fase di acquisizione ed una seconda di rilascio.

2PL risolve: perdita di aggiornamento, aggiornamento fantasma, letture inconsistenti

Presenta altre anomalie:

- letture sporche
- Deadlock

2PL stretto

I lock possono essere rilasciati solo dopo il commit.

elimina il rischio di letture sporche.

Time Stamp

Alternativa al 2PL

Timestamp: id che definisce un ordinamento totale sugli eventi di un sistema.

- Ogni transazione ha il suo timestamp.
- Lo schedule è accettato solo se riflette l'ordinamento totale dei timestamp.

Due contatori, RTM e WTM per ogni oggetto

read(x, ts) è accettata solo se $ts \geq WTM(x)$

write(x, ts) è accettata solo se $ts \geq RTM(x), ts \geq WTM(x)$

TS \Rightarrow CSR

TS non causa deadlock (avr.) ma uccide e rilancia spesso transazioni (molto costoso)

Conviene il 2PL

Risoluzione del deadlock

Corrisponde ad un ciclo nel grafo delle attese (il deadlock)

3 tecniche:

- 1) Timeout: Dopo un passo di tempo in attesa, una transazione riceve risposta negativa alla richiesta di lock e viene normalmente abortita.
- 2) Rilevamento deadlock: Ricerca di cicli nel grafo delle attese.
- 3) Prevenzione deadlock: Uccisione di transazioni "ospette".

Come scelgo chi uccide?

- Politiche interrompendi: Un deadlock si può risolvere uccidendo la transazione che possiede la vissorsa.

Criterio extra: uccide le transazioni con meno lavoro svolto. Geneva Starvation, visibile con il restart delle transazioni con il vecchio timestamp e la precedenza alle più anziane.

- Politiche non interrompendi: Una transazione può essere uccisa solo nel momento in cui effettua una nuova richiesta.

Il DBMS Garantisce privacy, efficienza, efficacia e persistenza.

Per far ciò, ha bisogno di poter gestire sofisticamente la memoria secondaria, nonché efficacemente, per questo necessita di strutture fisiche opportune.

Il file-system è il componente del SO. che si occupa della memoria secondaria.

I DBMS ne sfruttano le funzionalità per la gestione dei blocchi. Il DBMS gestisce i file allocati come se fossero un'unico spazio al cui interno costruire strutture fisiche con cui implementare le relazioni.

Prestazioni di memoria:

Dato un indirizzo d'accesso, le prestazioni di memoria si misurano in termini della somma del tempo di spostamento della testina, latenza e tempo di trasferimento (complessivo).

In generale il tempo di accesso in memoria secondaria è 3/4 ordini di grandezza maggiore rispetto a quelli in memoria centrale. Nelle applicazioni "I/O Bound" il costo dipende esclusivamente dal numero di accessi in memoria secondaria.

In memoria secondaria, l'accesso a blocchi contigui ha un minor costo (spostamento + latenza = 0)

Il buffer-manager, oltre a gestire i buffer, gestisce un directory che:

- Ha per ogni pagina file fisico en un solo blocco (mem. 2)
- 2 variabili distinte che indicano:
 - quanti programmi usano il file
 - Se la pagina è sporca o meno (setDirty)

Approfondiamo le operazioni di interfaccia del buffer manager:

Fix:

Cerca ^{la} pagina richiesta nel buffer; se c'è restituisce l'indirizzo, altrimenti cerca una pagina libera:

- Se la trova ci carica i dati dalla mem. secondaria e restituisce l'indirizzo
- Altrimenti vi sono 2 politiche:
 - Steal: Una pagina occupata viene scritta in mem2 e poi liberata per la richiesta
 - no-steal: L'operazione è posta in attesa.

Al fine di ottimizzare le scritture, il buffer manager può far partire le scritture in modo:

- Sincrono (force)
- Asincrono (anticipare/posticipare per coordinate)

I file sono organizzati logicamente in record, mappati nei blocchi di memoria secondo.
Le tuple di una relazione sono in blocchi contigui.

A volte si affiancano tuple di R. diverse per favorire i join.

L'dimensione del blocco dipende dal file-system, quella della tuple dall'applicazione.
Se la dimensione delle tuple è fissa, la struttura può essere semplificata, inoltre per tuple
molto grandi è possibile spezzarle su più blocchi.

Fattore di Blocco:

numero record per blocco:

- L_R : dimensione record

- L_B : dimensione blocco

$$\left[\frac{L_B}{L_R} \right] \text{ (parte intera)}$$

Lo spazio residuo può essere utilizzato (record "spanned") o meno ("unspanned")

Operazioni sulla pagina:

- Inserimento/Modifica tuple (può richiedere allocazione di nuovo spazio)
- Cancellazione
- Accesso ad una tuple/ ad un suo campo

I DBMS, attraverso i S.O., si occupano di fare l'accesso alle strutture fisiche che contengono i dati.

Strutture:

Le tuple organizzate all'interno dei blocchi dei file costituiscono le strutture primarie.
Esistono blocchi contenenti le strutture secondarie, che non contengono dati ma favoriscono l'accesso agli stessi.

Strutture primarie

Possono avere un'organizzazione simile alla sequenziale, che può essere:

- seriale (ordinamento fisico ma non logico)
- ordinato (ordinamento delle tuple coerente con quello di un campo)
- array

Seriale:

anche nota come file heap

Inserimenti in coda / al posto di record cancellati (periodiche riorganizzazioni)

Ordinata:

Permettono ricerche binarie a costo dell'ordinamento.

Con accesso casuale:

File basati con accesso molto efficiente.

L'hash su file funziona esattamente come su array, ma si sfrutta l'organizzazione in blocchi che "ammortizza" le probabilità di collisione.

Si utilizza l'hashing ad indirizzamento aperto per la gestione delle collisioni; in questo modo al momento della ricerca si potrebbe dover fare più di un accesso a blocchi.

Strutture ad albero (non sequenziali)

Dette anche indici, sono basate sull'uso di puntatori; ma l'accesso è in base ai valori di uno o più campi. Si usano sia come primarie che come secondarie.

Un indice I di un file f , è un altro file con record contenenti pseudochiave e indirizzo, ordinato secondo i valori della chiave.

- Indice primario: su un campo sul cui ordinamento è basata la memorizzazione.
- Indice secondario: su un campo con ordinamento diverso da ↑
- Indice denso: ha 1 record per ciascun record del file
- Indice sparso: ha un numero di record < record del file.

Un indice primario può essere sia denso che sparso, mentre un indice secondario deve essere denso. Ogni file ha al più un indice primario, etanti indici secondari quanti vuole.

Un indice ha:

- Accesso diretto efficiente
- Scansione sequenziale ordinata efficiente
- Modifiche della chiave, inserimenti, eliminazioni inefficienti

Indici multilivello:

Poiché gli indici sono file, è possibile costruire indici sugli indici per evitare di fare ricerche fra blocchi diversi.

Le strutture di indice basate su strutture ordinate sono poco flessibili in presenza di elevata dinamicità.

Soltanente gli indici sono memorizzati e gestiti come B-tree (alberi di ricerca bilanciati).

Nel B-tree, al Nodo può corrispondere un blocco, e l'albero viene mantenuto bilanciato attraverso il riempimento parziale e la riorganizzazione periodica.

B+tree:

foglie collegate in una lista

B-tree:

I nodi intermedi possono avere puntatori direttamente ai dati:

SQL permette di definire degli indici con il comando

create index [unique] "Name" on "Table Name(AttributeList)"

Nei DBMS relazionali le strutture sono solitamente:

- Primaria:

- disordinata

- ordinata

- hash

- indici

- Organizzazione sequenziale se la struttura è ordinata (statico)
 - B-tree (dinamico)

Ottimizzazioni in base ai costi

Si valutano il numero di accessi in memoria e la dimensione dei risultati intermedi.

I DBMS implementano gli operatori dell'algebra relazionale al livello abbastanza basso.

I due operatori fondamentali sono:

- accesso diretto

- scansione

Per ottimizzare in base ai costi occorre sapere determinare:

- operazione da svolgere (accesso / scansione)

- il metodo con cui svolgerla (quale metodo di join? ...)

- ordine delle operazioni

Accesso diretto:

Possibile solo con gli indici o con le strutture hash
indice:

Efficace per le interrogazioni sulla "chiave dell'indice" (anche su intervalli)

Per predicati congiuntivi (si sceglie il più selettivo e poi si verifica sugli altri dopo la lettura)

Per predicati disgiuntivi (servono indici su tutti, ma conviene usarli se molto selettivi e facendo attenzione ai duplicati)

Hash

Praticamente identico, ma non è efficace sulle interrogazioni su intervalli.

Scansione:

esprime la ricerca su una relazione. Ha complessità media $O(n)$ e necessita che tutti i blocchi siano trasferiti nel buffer.

Nel caso di scansione su un attributo ordinante, si arriva a $O(\log n)$.

Alcune query richiedono l'ordinamento delle tuple (ottimizzando il join). Si può usare quicksort se la relazione è tutta nel buffer, altrimenti: il mergesort (a più vel).

Join

Esistono molti metodi di join:

- Nested Loop
- Merge Scan
- Hash based

Nested Loop:

per ogni tupla esterna si esaminano tutte le tuple di quella interna per verificare la condizione di join. Il costo dipende dal numero di accessi e dalle dimensioni della tabella interna.

Formula costo

Se R_i non sta tutta nel buffer:

$$\text{Record}_e \cdot \frac{1}{\text{nr. record}_i} + \text{peso record}_e$$

↓
numero record nel
buffer.

Se R_i sta nel buffer

$$\text{peso record}_e + \text{peso record}_i$$

Merge Scan:

Si ordinano le tabelle sugli attributi di join

Si trova l'elemento della 2^a da cui partire rispetto al primo elemento della prima. Il costo è l'ordinamento.

Si usa per il join naturale e per l'equi-join.

Hash

Si usa solo per join naturale ed equi-join, la funzione di hashing h si usa per partizionare le relazioni S ed R.

Tali partizioni sono inserite in tabelle aggiuntive.

Ha bisogno quindi di memoria aggiuntiva, ma è migliore di ~~un~~ nested loop nel caso di equijoin.

Costo: $3(br + bs) + r \cdot s$ esperimenti di blocco

Al costo di una query contribuiscono molti fattori, tra cui è predominante l'accesso al disco.

Per semplicità consideriamo un unico fattore t come tempo per accedere al blocco.

Per ottimizzare le query, sia le selezioni che i join vengono fatti nell'ordine che garantisce la minor dimensione del risultato intermedio

Processo di ottimizzazione

- Si costruisce l'albero di decisione
- Si valuta il costo di ciascun piano d'esecuzione
- Si sceglie il piano dal costo minore.

Si trova sempre una buona soluzione, non necessariamente ottima