

TEORIA BD

domenica 27 giugno 2021

15:35

3

Qualità del Servizio

Le prestazioni vengono analizzate in riferimento a un certo carico di lavoro – workload, attraverso un’attività detta **tuning**, in cui si analizzano le prestazioni del sistema e si cercano di adottare correttivi. È svolto in fase di progettazione, quando il sistema non è ancora in produzione.

Il **workload** indica quali sono le interrogazioni o gli aggiornamenti più importanti e con che frequenza vengono eseguiti e quali sono le prestazioni desiderate/necessarie per tali interrogazioni e aggiornamenti e per il sistema in generale.

Gestore delle Strutture di Memorizzazione

Le informazioni di una BD sono memorizzate su disco.

Le **prestazioni di una memoria** si misurano in **tempo di accesso**:

$$\text{Tempo di accesso} = \text{latenza} + \frac{\text{dimensione dati}}{\text{velocità di trasferimento}}$$

latenza: tempo necessario per accedere al primo byte;

tempo di trasferimento: tempo necessario per muovere i dati.

Un DB, a causa della sua dimensione, risiede normalmente su dischi.

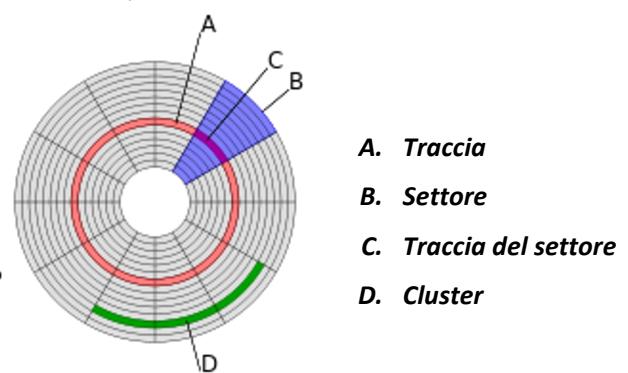
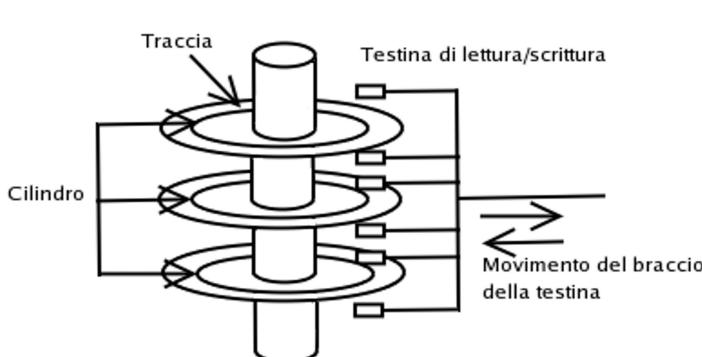
I dati devono essere trasferiti in memoria centrale per essere elaborati dal DBMS; il trasferimento non avviene in termini di singole tuple, bensì di **blocchi** (pagine)

Gestore dei File

Un **hard disk** (HD) è un dispositivo elettromeccanico per la conservazione di informazioni sotto forma magnetica, su supporto rotante a forma di piatto su cui agiscono delle **testine** di lettura/scrittura.

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

Dati memorizzati su uno stesso cilindro possono essere recuperati molto più velocemente che non dati distribuiti su diversi cilindri.

Ogni traccia è divisa in **settori**, le più piccole unità di dati che possono essere lette o scritte;

Per leggere un settore:

- il braccio si posiziona sopra la traccia,
- i dati vengono letti o scritti quando il settore passa sotto la testina.

Le **prestazioni** delle memorie possono essere classificate in:

- **Interne**: dipendono da caratteristiche meccaniche e tecniche di memorizzazione e codifica dei dati;
- **Esterne**: dipendono da tipo di interfaccia e file system.

Le **prestazioni interne** dipendono principalmente dal tempo di **latenza**,

Esso è composto da:

- **Command Overhead Time**: tempo necessario a impartire comandi al drive (dell'ordine di 0.5 msec e trascurabile).
- **Seek Time** (Ts): tempo impiegato dal braccio a posizionarsi sulla traccia desiderata; dai 2 ai 20 msec, e quindi molto determinante.
- **Settle Time**: tempo richiesto per la stabilizzazione del braccio
- **Rotational Latency** (Tr): tempo di attesa del primo settore da leggere; da 2 a 11 msec, anch'esso importante.

I dati sono trasferiti tra il disco e la memoria principale in unità chiamate **blocchi**.

Un blocco (o pagina) è una **sequenza contigua di settori su una traccia**, e costituisce l'unità di I/O per il trasferimento di dati da/per la memoria principale. **Pagine piccole** comportano un maggior numero di operazioni di I/O. **Pagine grandi** richiedono più spazio in memoria per essere caricate.

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 di seek e dipende dalla dimensione della pagina e dal transfer rate.

PostgreSQL organizza lo spazio fisico in **tablespace**, ciascuno corrispondente a una posizione su disco in cui vengono memorizzati i file corrispondenti a oggetti dei database. Il file corrispondente ad una tabella è memorizzato in un singolo tablespace, ma un tablespace può contenere più relazioni.

L'uso dei tablespace permette al sistema di decidere dove memorizzare i vari oggetti, ad esempio memorizzando sullo stesso cilindro dati che vengono acceduti insieme.

L'uso dei tablespace permette quindi di raggiungere le migliori prestazioni nel caso di DB di grandi dimensioni.

Abbiamo compreso che le prestazioni di un DBMS dipendono fortemente da come i dati sono organizzati su disco, però queste informazioni non possono essere note al file system, quindi la gestione viene lasciata ai sistemi di gestione dati.

I dati sono generalmente memorizzati in forma di **record**: un **insieme di valori collegati**.

Ogni valore è formato da uno o più byte e corrisponde ad un **campo** del record.

Una collezione di nomi di campi a cui sono associati i tipi corrispondenti costituisce un **tipo di record**.

Per ogni tipo di record nel DB deve essere definito uno **schema** (fisico) che permetta di **interpretare** correttamente il **significato dei byte** che costituiscono il record.

Un **file** è una **sequenza di record**.

Un file è detto **file con record a lunghezza fissa** se tutti i record memorizzati nel file hanno la stessa dimensione (in byte); altrimenti, parliamo di **file con record a lunghezza variabile**.

Modi per memorizzare blocchi su disco:

- I blocchi del file sono allocati in blocchi di disco contigui (Allocazione Continua)
- Ogni blocco di un file contiene un puntatore al successivo blocco del file (Allocazione Concatenata)

I file che contengono i record dei dati costituiscono l'**organizzazione primaria dei dati**.

Il termine organizzazione primaria di un DBMS indica quindi l'**insieme di tutti i file contenenti i record corrispondenti alle tuple contenute nelle tabelle rappresentate a livello logico**.

Gestore del Buffer

Un aspetto significativo per minimizzare gli accessi a disco è mantenere più blocchi possibile nel buffer della memoria principale.

Il gestore del buffer si occupa di dirigere questi aspetti, ad esempio decidendo quale blocco sostituire in caso di buffer pieno; la gestione del buffer è fondamentale dal punto di vista prestazionale.

Il buffer è organizzato in pagine, che hanno la stessa dimensione delle pagine/blocchi su disco.

A fronte di una richiesta di una pagina, ad esempio durante una operazione di lettura (SELECT) il **Buffer Manager** (BM) opera come segue:

- Se la pagina è già nel buffer, viene fornito al programma chiamante l'**indirizzo della pagina** corrispondente.
- Se la pagina non è in memoria:
 - Il BM seleziona una pagina nel buffer per la pagina richiesta:
 - Se la pagina è libera viene portata dal disco alla memoria centrale.
 - Se la pagina prescelta dal buffer è occupata, questa viene riscritta su disco solo se è stata modificata e non ancora salvata su disco e se nessuno la sta usando.

A questo punto il BM può leggere la pagina da disco e copiarla nella pagina del buffer selezionata, rimpiazzando così quella prima presente

Indici

Le organizzazioni dei file viste non permettono di ottenere prestazioni soddisfacenti se si eseguono ricerche con condizioni su attributi che non sono quelli utilizzati per organizzare i record nel file.

Per ovviare a questi limiti si creano delle strutture ausiliarie di accesso, chiamate anche **indici**, che **forniscono cammini di accesso alternativi ai dati**, per localizzare velocemente i dati di interesse. Le strutture ausiliarie di accesso permettono di eseguire in maniera **più efficiente operazioni di ricerca**.

Si usa il termine **chiave di ricerca** per indicare **un attributo o insieme di attributi** usati per la ricerca (anche diversi dalla chiave primaria).

Un indice è un **multi-insieme di coppie** del tipo (ki, ri) dove:

- ki è un valore per la chiave di ricerca su cui l'indice è costruito,
- ri può assumere diverse forme, per il momento assumiamo che sia un **puntatore a un record** con valore di chiave ki

Vantaggi dell'uso degli indici:

- l'indice occupa uno spazio minore rispetto al file dati
- L'uso di indici rende l'esecuzione delle interrogazioni più efficiente (ma rende più costosi gli aggiornamenti)

Indice primario se la chiave di ricerca è chiave anche per la relazione, altrimenti si dice **indice secondario**.

Per evitare di duplicare inutilmente i valori di chiave, la soluzione più comunemente adottata per gli indici secondari consiste nel raggruppare tutte le coppie con lo stesso valore di chiave in una **lista di puntatori**.

Le diverse tecniche di indice differiscono nel modo in cui organizzano l'insieme di coppie (ki, ri):

- **Indici ordinati** (anche chiamati **indici ad albero**): le coppie(ki, ri) vengono **mantenute esplicitamente**. Le coppie vengono salvate in un file, ordinate rispetto ai valori di chiave ki . Le coppie (ki, ri) sono contenute in nodi (quindi blocchi) foglia e le foglie sono collegate a lista mediante puntatori (PID) per favorire **ricerche di tipo range**.
- **Indici hash**: esiste una **funzione hash** h per cui, se una coppia (ki, ri) appartiene all'indice allora $h(ki) = ri$. Le coppie non vengono memorizzate su un file apposito. L'uso di indici ad albero ha lo svantaggio di richiedere la scansione di una struttura dati, memorizzata su disco, per localizzare i dati; Un indice hash è **clusterizzato** se i **record** che condividono lo stesso valore per la chiave di ricerca sono **memorizzati in posizioni adiacenti** nel file dei dati. Indici hash consigliati per selezioni con **condizioni di uguaglianza**.

1

Transazioni

Sommario:

1. **Cos'è una transazione**
2. **Proprietà transazioni**
3. **Transazioni flat**
4. **Transazioni e vincoli di integrità**

Transazione = Unità logica di elaborazione

proprietà vengono garantite dal gestore della concorrenza e gestore del ripristino: **ACID Atomicità, consistenza, isolamento, Durabilità (persistenza)**

Atomicità = è detta anche proprietà tutto-o-niente; tutte le operazioni di una transazione devono essere trattate come una singola unità

Consistenza = Una transazione deve agire sulla base di dati in modo corretto

Isolamento = L'esito (lo stato generato) di ogni transazione non deve essere influenzato da esecuzioni concorrenti di altre transazioni

Durabilità (o Persistenza) = I risultati di una transazione terminata con successo devono essere resi permanenti nella base di dati nonostante possibili malfunzionamenti del sistema.

Transazioni Flat

Esistono diversi tipi di transazioni; il tipo più semplice corrisponde alle transazioni flat.

Sono convenienti quando dobbiamo organizzare la nostra logica applicativa con **codice semplice e di breve durata**.

Una transazione flat viene iniziata:

- **implicitamente** all'esecuzione del primo comando SQL; ogni volta che eseguiamo un comando inizia una transazione flat.
- **esplicitamente** con il comando BEGIN [TRANSACTION] o START TRANSACTION

Esiti possibili:

- **Terminazione Corretta**, Una transazione termina correttamente quando, dopo aver eseguito tutte le proprie operazioni, esegue una particolare istruzione SQL, detta **COMMIT [WORK]**
- **Terminazione non Corretta**, Possiamo indicare una terminazione non corretta esplicitamente: la transazione decide di "abortire" eseguendo l'istruzione SQL ROLLBACK [WORK]; Il DBMS deve "disfare" (UNDO) le eventuali modifiche apportate al DB dalla transazione, riportando lo stato della BD a quello prima dell'inizio della transazione.

- **Guasto:** Si verifica un guasto e la transazione non può essere portata a termine.
- **Violazione di un Vincolo:** caso di violazione di vincoli di integrità da parte della transazione, ovvero nel caso in cui uno stato di essa provochi la violazione dell'integrità della BD.

Ritornando sulla proprietà di consistenza: questa proprietà **ammette la possibilità che gli stati intermedi possano violare la consistenza.**

Esistono due diverse modalità di controllo che stabiliscono quando effettuare la valutazione del vincolo e ripristinare la consistenza:

1. Con **NOT DEFERRABLE** tutte le transazioni potranno essere solo immediate.
2. Con **DEFERRABLE** può essere sia immediata che differita, a seconda di quello che dichiariamo dopo.
 - a. **INITIALLY IMMEDIATE:** di base tutte le transazioni sono dichiarate immediate; Il vincolo può essere sovrascritto all'interno della transazione attraverso: SET CONSTRAINTS ALL DEFERRED; In questo caso si parlerà quindi di transazione differita.
 - b. **INITIALLY DEFERRED:** di base tutte le transazioni sono dichiarate differite; Il vincolo può essere sovrascritto all'interno della transazione attraverso: SET CONSTRAINTS ALL IMMEDIATE; In questo caso si parlerà quindi di transazione immediata.

E' prevista una modalità di terminazione implicita, gestita da una proprietà chiamata **AUTOCOMMIT**, che può essere o meno abilitata:

- Se **AUTOCOMMIT = TRUE**, ogni comando costituisce implicitamente una transazione a se stante (= eseguire COMMIT dopo ogni comando SQL);
- Se **AUTOCOMMIT = FALSE**, tutti i comandi eseguiti all'interno di una sessione costituiscono implicitamente una transazione (= eseguire COMMIT dopo l'ultimo comando della sessione).

Concorrenza

1. **Obiettivi gestione della concorrenza**
2. **Anomalie**
3. **Protocolli di locking per la gestione della concorrenza**
4. **Livelli di isolamento**

Un DBMS, dovendo supportare l'esecuzione di diverse transazioni che accedono a dati condivisi, potrebbe eseguire tali transazioni in sequenza, attraverso quindi una **serial execution**.

Un DBMS potrebbe anche eseguire più transazioni in concorrenza, alternando l'esecuzione di operazioni di una transazione con quella di operazioni di altre transazioni, questa è detta **interleaved execution (Esecuzione Concorrente di Transazioni)**.

Tipi di anomalie generate dall'esecuzione concorrente:

- **Lost Update**, perdita di aggiornamento;
- **Dirty Read**, lettura sporca;
- **Unrepeatable Read**, letture inconsistenti;
- **Phantom Row**, aggiornamento fantasma;

Unrepeatable Read: per il concerto degli U2 (finalmente la data è stata fissata!) vedete che il prezzo è di 40 €, ci pensate su 5 minuti, ma il prezzo nel frattempo è salito a 50 € (!?)

Lost Update: due persone, in due agenzie diverse, comprano entrambe l'ultimo biglietto per il concerto degli U2 a Roma (?)

T1	X	T2
R(X)	1	
X=X-1	1	
	1	R(X)
	1	X=X-1
W(X)	0	
Commit	0	
	0	W(X)
	0	Commit

Dirty Read: nel programma dei concerti degli U2 figura una tappa a Bologna il 15/07/02, ma quando provate a comprare un biglietto per quella data vi viene detto che in realtà non è ancora stata fissata (?)

Esempio: X corrisponde al calendario dei concerti degli U2, modificato da T1 aggiungendo la nuova tappa (15/07/02)

T1	X	T2
R(X)	0	
X=X+1	0	
W(X)	1	
	1	R(X)
Rollback	0	
	0	...
	0	Commit

Questa lettura è "sporca"!

- **Esempio:** T2 legge il calendario e trova la nuova data (15/07/02), se poi prova comprare il biglietto più avanti

Unrepeatable Read: per il concerto degli U2 (finalmente la data è stata fissata!) vedete che il prezzo è di 40 €, ci pensate su 5 minuti, ma il prezzo nel frattempo è salito a 50 € (!?)

aggiungendo la nuova tappa (15/07/02)

	T1	X	T2
R(X)	0		
	0	R(X)	
	0	X=X+1	
	1	W(X)	
	1	Commit	
R(X)	1		
Commit	1		

- Esempio: T2 legge il calendario e trova la nuova data (15/07/02), se poi prova comprare il biglietto più avanti nel codice (...), la nuova data non compare più

	T1	X	T2
R(X)	0		
	0	R(X)	
	0	X=X+1	
	1	W(X)	
	1	Commit	
R(X)	1		
Commit	1		

Esempio: X corrisponde al prezzo del concerto degli U2

	T1	T2
R(Biglietti)		
12/06/19	Bologna	Italia
18/07/19	Firenze	Italia
5/08/19	Milano	Italia
		NBiglietti disponibili
		50
		32
		250

Alla seconda lettura T1 «vede» una tupla in più – phantom row

	T1	T2
t1	R(Biglietti)	
t2		Insert (Biglietti, I3)
...		
t3	R(Biglietti)	
...		
	Commit	
		Commit

Una comune tecnica usata dai DBMS per evitare i problemi visti consiste nell'uso di **lock**.

I lock - "blocchi" sono un meccanismo comunemente usato dai sistemi operativi per **disciplinare l'accesso a risorse condivise**.

Per eseguire un'operazione su una risorsa condivisa è **prima necessario "acquisire" un lock sulla risorsa interessata**, ad esempio su una tupla.

La richiesta di lock è **implicita**, e quindi non è visibile a livello SQL;

I lock sono di vario tipo, quelli di base sono:

S-lock (Shared): un lock condiviso è necessario per leggere;

X-lock (eXclusive): un lock esclusivo è necessario per scrivere/modificare.

Livello di isolamento	Lost update	Dirty read	Unrepeatable read	Phantom problem
READ UNCOMMITTED	NO	YES	YES	YES
READ COMMITTED	NO	NO	YES	YES
REPEATABLE READ	NO	NO	NO	YES
SERIALIZABLE	NO	NO	NO	NO

Ripristino

- Obiettivi gestione del ripristino,
- Ripristino basato sui file di log,
- Problemi prestazionali ripristino,
- Ripristino per media failure.

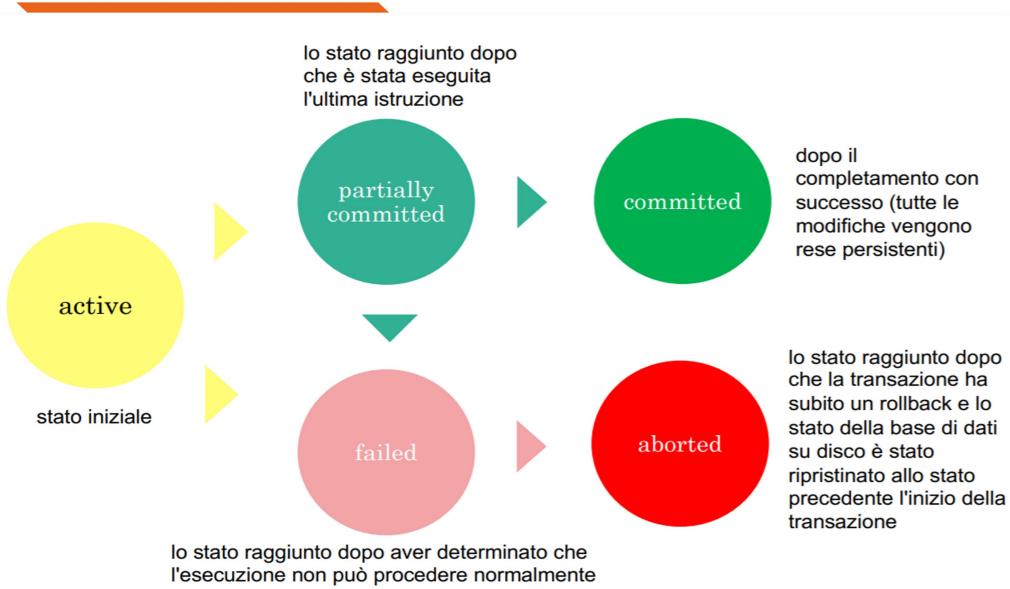
Durability e Atomicity vengono **assicurati** da una particolare componente del Transaction Manager: **il gestore del ripristino**.

I possibili malfunzionamenti possono essere classificati in 3 gruppi:

- Transaction failure:** è il caso in cui una transazione abortisce per sua scelta (ROLLBACK) gli effetti della transazione sul DB devono essere annullati.
- System failure:** il sistema ha un guasto hardware o software che provoca l'interruzione di tutte le transazioni in esecuzione, senza però danneggiare la memoria permanente (dischi) spesso dovuto

a problemi a memoria volatile (memoria principale e cache).

- **Media failure:** il contenuto (persistente) della base di dati viene danneggiato problemi a memoria non volatile (dischi, nastri).



Le attività di ripristino eseguite a valle del verificarsi di un **transaction o di un system failure** vengono genericamente indicate con il termine **ripresa a caldo**; essa si basa su i file di log.

Durante l'esecuzione di una transazione tutte le operazioni di scrittura sono registrate in un file di log, **memorizzato su memoria stabile**.

Affinché il Log possa essere utilizzato per ripristinare lo stato del DB a fronte di malfunzionamenti, è importante che venga applicato il cosiddetto **protocollo WAL (Write-ahead Logging)**.

Prima di scrivere su disco una pagina P modificata, il corrispondente record di Log deve essere già stato scritto nel file di Log.

Vi sono 4 passi per costruire un log che possa garantire un possibile ripristino:

1. Una transazione esegue una modifica (update) su una pagina P,
2. a valle di questo aggiornamento il sistema deve generare l'opportuno record del file di log in memoria centrale,
3. in base al protocollo WAL questo record viene conservato in memoria stabile,
4. a quel punto la pagina P può essere scritta su disco.

Politica No-Steal: Si mantiene la pagina P nel buffer e si attende che T abbia eseguito COMMIT prima di scriverla su disco (copia su disco dal momento del COMMIT, non prima).

Politica Steal: Si scrive P su disco quando più conviene (ovvero quando è necessario per liberare il buffer o per ottimizzare le prestazioni di I/O), eventualmente anche prima della terminazione di T.

Politica Force: Prima di scrivere il record di COMMIT sul Log si forza la scrittura su disco di tutte le pagine modificate da T.

Politica No-Force: Si scrive subito il record di COMMIT sul file di Log.

La combinazione **steal-no force** è la più utilizzata dai DBMS.

L'approccio della Ripresa a caldo (Recovery con log), usato per System Failure e Transaction Failure non

basta per il media failure.

Il sistema deve adottare una strategia per affrontare i failure che riguardano memoria persistente.

Il media failure viene quindi affrontato combinando recovery con log e **dump**.

Un dump (backup) è una copia completa della base di dati, **memorizzata in memoria stabile**.

Domande ricorrenti BD

1. Si illustrino le motivazioni e la struttura dell'architettura ANSI-SPARC per DBMS in particolare per quanto riguarda l'indipendenza dei dati.

L'architettura ANSI-SPARC è usata principalmente per garantire l'indipendenza dei dati dalla loro rappresentazione fisica. Essa si suddivide in 3 livelli:

- Schema esterno: descrizione di una parte della base di dati (utilizzo di viste) in un modello logico
- Schema logico: descrizione della base di dati nel modello logico principale del DBMS
- Schema fisico: rappresentazione dello schema logico per mezzo di strutture fisiche di memorizzazione.

Questa architettura permette indipendenza:

- Fisica: il livello logico ed esterno sono indipendenti da quello fisico, dunque posso modificare le strutture fisiche di memorizzazione senza modificare quelle logiche.
- Logica: il livello esterno è indipendente da quello logico perché le modifiche allo schema logico che lascino inalterato lo schema esterno sono trasparenti ed aggiunte/modifiche di schemi esterni non richiedono modifiche del livello logico.

2. Si definisca il concetto di vista, la relazione tra vista e modello ANSI SPARC e infine si spieghi come le viste possono essere impiegate utilmente nel gestire autorizzazioni sui dati nei database.

Una vista offre la consultazione di tabelle virtuali, ossia tabelle che non hanno una corrispondenza con le tabelle vere e proprie del DB; possono essere create a partire da una singola tabella (viste semplici) o da più tabelle (viste complesse) e possono proiettarne solo alcuni campi.

Le viste possono contenere altre viste, basta che non ci sia mutua dipendenza (ricorsione); inoltre sono indispensabili per esprimere query con operatori aggregati nidificati.

Possiamo collocare le viste in corrispondenza del livello esterno dell'architettura ANSI-SPARC: forniscono una visione parziale di tutto lo schema logico del DB.

Le viste possono essere utilizzate per gestire i permessi sui dati; ad esempio, se non voglio far visualizzare a tutti gli utenti del DB tutti i campi di una data tabella, posso creare una vista dei soli campi di interesse e dare a specifici utenti i permessi per accedere alla vista creata e non a tutta la tabella.

3. Definire cosa si intende per transazione e cosa sono le proprietà ACID.

Una transazione è un'unità logica di elaborazione su una base di dati che comprende una o più operazioni di read/write.

È bene che una transazione soddisfi le proprietà ACID:

- Atomicità: le operazioni di una singola transazione o vengono eseguite tutte o nessuna. In caso di successo viene eseguito il commit della transazione, altrimenti si ripristina lo stato precedente attraverso un rollback.
- Consistenza: ogni transazione porta il DB da uno stato consistente ad un altro stato consistente, preservando i vincoli di integrità. È possibile verificare la consistenza a seguito di ogni singola operazione della transazione o soltanto in corrispondenza del commit finale.
- Isolamento: le transazioni non si influenzano reciprocamente, ossia una transazione non può leggere dati intermedi di altre.

2

Controllo dell'Accesso ai Dati

Bisogna stabilire vari punti per definire come il DBMS possa soddisfare i 5 punti fondamentali che deve garantire:

- a. chi può accedere alla b.d.,
- b. a quali risorse si può accedere,
- c. chi può condividere privilegi e su cosa,
- d. chi può revocare i privilegi e su cosa.

Il controllo dell'accesso si basa sulla specifica di opportune **politiche di sicurezza**, che dipendono dal dominio considerato.

Le **autorizzazioni** sono triple di (**oggetti, soggetti, privilegi**) e sono mantenute nel database su disco all'interno di **cataloghi**.

comando **GRANT**: concede privilegi su una risorsa a uno o più utenti

comando **REVOKE**: toglie a uno o più utenti i privilegi che erano stati loro concessi.

Un utente può revocare solo i privilegi da lui stesso concessi.

Alla creazione di una risorsa, il sistema concede tutti i privilegi su tale risorsa all'utente che ha creato la risorsa.

Il controllo dell'accesso è effettuato mediante il **gestore degli accessi**.

Il gestore dell'accesso:

- durante l'esecuzione di comandi di GRANT e REVOKE, modifica opportunamente il contenuto dei cataloghi (inserendo, aggiornando o modificando tuple)
- durante l'esecuzione di altri comandi SQL, legge il contenuto dei cataloghi per stabilire se il comando può essere eseguito.

Esiste un grafo per ogni privilegio su una certa tabella.

Nel modello di controllo dell'accesso di SQL, i soggetti possono rappresentare ruoli.

RBAC - Role-based Access Control:

Un ruolo rappresenta una funzione che gli utenti ricoprono all'interno della realtà organizzativa in cui operano.

I privilegi possono essere concessi ai singoli utenti ma anche ai ruoli.

SQL permette anche di **definire i ruoli gerarchicamente**.

Una gerarchia sui ruoli definisce un ordinamento parziale \geq tra di essi:

- induce un'ereditarietà dei privilegi tra ruoli nella gerarchia,
- stabilisce una relazione tra gli utenti abilitati a ricoprire i vari ruoli.

Le **viste** sono un importante meccanismo attraverso cui è possibile **fornire forme più sofisticate di controllo dell'accesso**.

4

Elaborazione di Interrogazioni

Vediamo come il sistema esegue una interrogazione, sfruttando le strutture ausiliarie di accesso e la struttura attribuita ai file del livello fisico per determinare l'algoritmo di esecuzione più efficiente.

Passi nell'esecuzione di una interrogazione:

Parsing (eseguito dal parser)

- input: interrogazione SQL
- output: parse tree (o abstract syntax tree)

In questo passo viene controllata la correttezza sintattica della query SQL e ne viene generata una rappresentazione interna (in termini di parse tree).

Traduzione (effettuata dal translator):

- input: parse tree

- output: espressione algebrica canonica

In questo passo viene generata una espressione algebrica corrispondente a:
prodotto cartesiano delle relazioni in clausola FROM seguita da
condizioni di selezione, da clausola WHERE seguita da
condizioni di proiezione, da clausola SELECT

Ottimizzazione Logica:

- input: espressione algebrica canonica
- output: piano di esecuzione logico ottimizzato (logical query plan – LQP)

Piano di esecuzione logico è un'espressione algebrica (in un'algebra estesa) per l'interrogazione, rappresentata come albero; l'espressione algebrica canonica viene rappresentata come piano di esecuzione logico.

Il piano di esecuzione logico iniziale viene poi trasformato in un piano equivalente ma più efficiente da eseguire, usando le proprietà dell'algebra relazionale.

Ottimizzazione Fisica:

- input: piano di esecuzione logica ottimizzato,
- output: piano di esecuzione fisico ottimale (physical query plan - PQP).

Il piano di esecuzione fisico è l'algoritmo di esecuzione dell'interrogazione, rappresentato come albero, sul livello fisico.

Si seleziona il piano di esecuzione fisico più efficiente, che riduce quindi gli accessi a disco.

Si determina in modo preciso come la query sarà eseguita (per esempio si determina che indici si useranno); la scelta avviene considerando tutti i possibili piani fisici che realizzano il piano logico scelto, valutando il costo di ognuno di essi e scegliendo il piano fisico di minor costo.