Transazioni

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

Definizione di una transazione

- Un'unità elementare di lavoro svolta da un programma applicativo, cui si vogliono associare particolari caratteristiche di correttezza, robustezza, isolamento.
- Sistema transazionale: mette a disposizione un meccanismo per la definizione e l'esecuzione di transazioni

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-)

Transazione

- Ogni transazione è incapsulata tra due comandi:
  - begin transaction (bot)
  - end transaction (eot)
- Dentro ad una transazione devono essere eseguiti (solo una volta) almeno uno dei seguenti comandi:
  - commit work (buon fine dell'operazione)
  - rollback work (aborto)

Transazione ben-formata

(proprietà che si manifesta a tempo di esecuzione)

- Una transazione comincia con *begin transaction* e finisce con *end transaction*, nella cui esecuzione solo uno dei due comandi commit work o rollback work viene eseguito, le operazioni sui dati sono eseguite solo quando si fa una di queste due operazioni

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 è assicurata dal sottosistema di ripristino (recovery)

Transazione -Atomicità

- Non deve lasciare il database in uno stato intermedio:
  - un errore prima di commit deve causare l'undo delle operazioni fatte prima
  - un errore dopo il commit deve far fare un REDO del lavoro fatto prima
- Possibili comportamenti:
  - buon fine
  - abort richiesto dall'applicazione (suicidio)
  - aborto richiesto dal sistema (morte)

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

Transazione - Consistenza

- La consistenza richiede l'esecuzione della transazione senza violare i vincoli di integrità definiti sulla base di dati
- La verifica dei vincoli di integrità può essere:
  - immediata: durante la transazione (le operazioni devono essere inabilitate)
  - differita: quando il cliente ha deciso il commit nel qual caso l'intera transazione deve essere cancellata
  - **conseguenza: se lo stato iniziale è corretto lo è anche quello finale**

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

Transazione - Isolamento

- Richiede che ogni transazione venga eseguita indipendentemente dall'esecuzione di tutte le altre transazioni concorrenti
  - eliminare effetto domino: il rollback di una transazione causa il rollback delle altre

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
- indipendente dai guasti di sistema

Transazioni e moduli di sistema

- Atomicità e persistenza sono garantiti dal controllo dell'affidibalità
- L'isolamento è garantito dal controllo di concorrenza
- La consistenza è infine gestita dai compilatori DDL che introducono le opportune verifiche
  - nb: la transazione non corrisponde ad ogni interazione del sistema con l'utente

Transazioni e protocolli

- Le proprietà 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'atomicità, 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 (cioè 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 - 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
- Carico applicativo di DBMS : numero di transazioni per secondo
- Misurato in tps (transation per second):10-100 migliaia di operazioni
  - esempi: banche, aerei...

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	

- La somma depositata da T2 è persa (lost update) e non si ottiene l'effetto voluto sulla base di dati

Transazioni – Concorrenza: esempio

- l'esecuzione concorrente di più transazioni genera un'alternanza di computazioni da parte delle varie transazioni, detta i*nterleaving*
- 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

Anomalie: Perdita di update

Anomaly 1: Update loss

- Consider two identical transactions:
  - $t1 : r(x), x = x + 1, w(x)$
  - $t2 : r(x), x = x + 1, w(x)$
- Assume initially $x=2$; after serial execution $x=4$
- Consider concurrent execution:

Transaction t_1	Transaction t_2
bot	
$r_1(x)$	
$x = x + 1$	
	bot
	$r_2(x)$
	$x = x + 1$
$w_1(x)$	
commit	
	$w_2(x)$
	commit

- One update is lost, final value $x=3$

Anomalie: Letture sporche

Anomaly 2: Dirty read

- Consider the same two transactions, and the following execution (note that the first transaction fails):
  - Transaction t_1 / Transaction t_2

Transaction t_1	Transaction t_2
bot	
$r_1(x)$	
$x = x + 1$	
$w_1(x)$	
	bot
	$r_2(x)$
	$x = x + 1$
abort	
	$w_2(x)$
	commit

- Critical aspect: t_2 reads an intermediate state (dirty read)

Anomalie: Letture inconsistenti

Anomaly 3: Inconsistent read

- t_1 repeats two reads:
  - Transaction t_1 / Transaction t_2

Transaction t_1	Transaction t_2
bot	
$r_1(x)$	
	bot
	$r_2(x)$
	$x = x + 1$
	$w_2(x)$
	commit
$r_1(x)$	
commit	

- t_1 reads different values for x

Anomalie : Update fantasmi

Anomaly 4: Ghost update

- Assume the integrity constraint $x + y + z = 1000$;
  - Transaction t_1 / Transaction t_2

Transaction t_1	Transaction t_2
bot	
$r_1(x)$	
	bot
	$r_2(y)$
$r_1(y)$	
	$y = y - 100$
	$r_2(z)$
	$z = z + 100$
	$w_2(y)$
	$w_2(z)$
	commit
$r_1(z)$	
$s = x + y + z$	
commit	

- In the end, $s = 110$: the integrity constraint is not satisfied
- t_1 sees a ghost update

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

Teoria del controllo della concorrenza

- Transazione: sequenza di operazioni di lettura e scrittura
- Ogni transazione ha un identificatore univoco assegnato dal sistema
- Assumiamo di omettere il begin / end transaction
- una transazione può essere rappresentata come: t1: r1(x)r1(y)w1(x)w1(y)

Schedule:

- Rappresenta la sequenza di operazioni I/O presentate da transazioni concorrenti
- esempio: S :r1(x)r2(z)w1(x)w2(z)
- Il compito del controllo di concorrenza è di accettare alcuni schedule e rifiutarne altri

Scheduler serializzabile

- Uno scheduler seriale non è efficiente anche se elimina alla fonte molte anomalie
- scheduler view-serializzabile:
  - Uno scheduler che produce lo stesso risultato di uno seriale a partire dalle stesse transazioni ma in condizioni di concorrenza

Primitive di lock

- Tutte le operazioni di lettura e scrittura devono essere protette tramite l'esecuzione di tre diverse primitive:
  - r_lock
  - w_lock
  - unlock
- Lo scheduler che riceve una sequenza di richieste di esecuzione di queste primitive da parte delle transazioni ne determina la concorrenza

Transazioni ben formate

- Stato di un oggetto:
  - Libero
  - r-locked (bloccato da un lettore)
  - w-locked (bloccato da uno scrittore)
- Ogni read di un oggetto deve essere preceduto da r_lock e seguito da unlock
- Ogni write di un oggetto deve essere preceduto da w_lock e seguito da unlock

Tabella dei conflitti

	libera	r-locked	w-locked
r-lock	si	ok	no
wlock	si	no	no

Si: blocco della risorsa il programma procede
no: il programma va in attesa che la risorsa venga sbloccata
ok : il contatore dei lettori viene incrementato per ogni lettore e decrementato dopo operazione unlock (una risorsa puo` essere letta da piu` lettori, ma non modificata mentre si legge)

Locking a due fasi

- Una transazione dopo aver rilasciato un lock non può acquisirne degli altri:
  - prima fase: la transazione acquisisce tutti i lock che le servono (fase crescente)
  - seconda fase (calante) i lock acquisiti vengono rilasciati

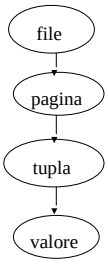
Assunzioni

- Transazioni ben formate
- politica dei conflitti (come da tabella)
- locking a due fasi

->implicano la serializzabilita`

Realizzazione del locking

Lock a livello di:
•file
•pagina fisica
•tupla
•valore di un attributo di una tupla



Granularita ridotta

Maggiore concorrenza

Problema del deadlock

- T1
  - w-lock(d1)
  - w-lock(d2)
  - lettura e scrittura di d1 e d2
  - unlock(d1)
  - unlock(d2)
- T2
  - w-lock(d2)
  - w-lock(d1)
  - lettura e scrittura di d1 e d2
  - unlock(d2)
  - unlock(d1)

Insorgenza del deadlock

- T1 esegue w-lock(d1)
- T2 esegue w-lock(d2)

- T1 attende una risorsa controllata da T2
- T2 attende una risorsa controllata da T1

Tecniche risolutive del deadlock

- Time-out
  - un'attesa eccessiva e` interpretata come deadlock
- Prevenzione
  - evitare alcune condizioni di attesa
- Determinazione
  - ricerca dei cicli sul grafo di attesa

Tecnica più usata: time-out

Transazioni- Gestione del ripristino

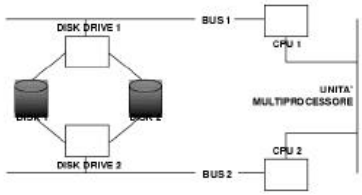
- Tre principali tipi di malfunzionamenti:
  - **malfunzionamenti del disco**: le informazioni residenti su disco vengono perse (rottura della testina, errori durante il trasferimento dei dati)
  - **malfunzionamenti di alimentazione**: le informazioni memorizzate in memoria centrale e nei registri vengono 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

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

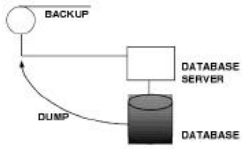
Come garantire la memoria stabile

- Replicazione on-line: mirroring di due dischi



Come garantire la memoria stabile

- Replicazione off-line: unità nastro (backup)



Ripristino: modello astratto

- Una transazione è sempre in uno dei seguenti stati:
  - **active**: lo stato iniziale
  - **partially committed**: lo stato raggiunto dopo che è stata eseguita l'ultima istruzione
  - **failed**: lo stato raggiunto dopo aver determinato che l'esecuzione non può 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

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

Ripristino: esempio

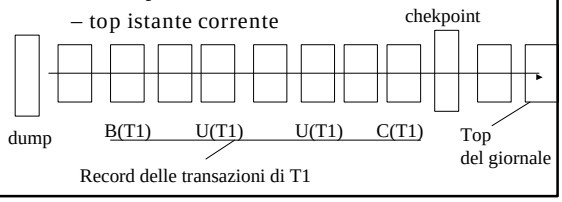
- T transazione che trasferisce Lit. 100000 dal conto A al conto B
- A = Lit. 1000000, B = Lit. 15000000
- dopo la modifica di A e prima della modifica di B, si verifica una caduta di sistema e i contenuti della memoria vengono persi
  - se si riesegue T: stato (inconsistente) in cui A = Lit. 800000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

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

Giornale delle transazioni

- File sequenziale di record che descrivono le azioni svolte dalle transazioni
  - Dump
  - checkpoint
  - top istante corrente



Principale funzione del giornale di transazione

- Registra in memoria stabile le azioni svolte dalla transazione sotto forma di transazioni di stato
  - se UPDATE (U) trasforma O dal valore O1 al valore O2
  - registrazione sul giornale:
    - BEFORE_STATE (U)=O1
    - AFTER_STATE (U) =O2

Uso del giornale

- DOPO: rollback-work oppure guasto prima di commit
  - UNDO T1: O=O1
- DOPO: guasto dopo commit
  - REDO T1: O=O2

Proprietà idempotenza di undo e redo:

dato che queste primitive sono definite tramite una azione di copiatura vale per esse la propieta per la quale l'effettuazione di un numero arbitrario di undo e redo equivale allo svolgimento della stessa azione una sola volta

Undo(Undo(A)) = Undo(A)
Redo(Redo(A)) = Redo(Redo(A))

Tipi di record del giornale

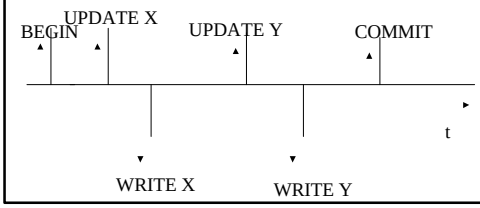
- **Record relativi ai comandi transazionali**:
  - BEGIN, COMMIT, ABORT
- **Record relativi alle operazioni**:
  - INSERT, DELETE,UPDATE
- **Record relativi alle azioni di recovery**:
  - DUMP (copia completa della base di dati, normalmente viene effettuata quando il sistema non e` operativo),
  - CHEKPOINT (svolta periodicamente, obiettivo: controllo delle transazioni attive)

Tipi di record nel giornale

- Record relativi ai comandi transazionali: B(T),C(T),A(T)
- Record relativi alle operazioni: I(T,O,AS), D(T,O,BS), U(T,O,BS,AS)
- Record relativi alle azioni di recovery: DUMP, CKPT(T1,T2,…Tn)
- Campi dei record:
  - T identificatore di transazione
  - O identificatore di oggetto
  - BS, AS: before state, after state

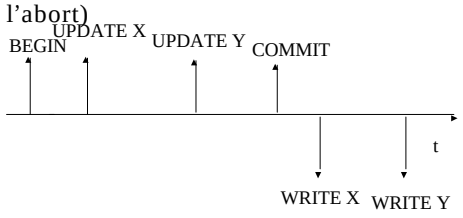
Scritture del log e della base di dati

- Scrittura del database prima del commit (richiede scritture per realizzare l'abort)



Scritture del log e della base di dati

- Scrittura del database dopo il commit (non richiede scritture per realizzare l'abort)



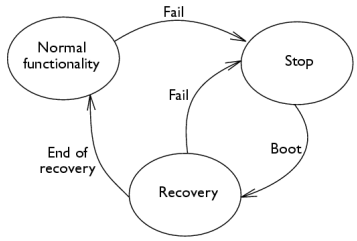
In caso di guasto

- Guasto soft perdita di memoria centrale richiede una RIPRESA A CALDO
- Guasto hard danneggiamento della memoria richiede una RIPRESA A FREDDO

Modello fail stop

- Quando il sistema individua un guasto, sia di sistema che di dispositivo, esso forza un completo stop delle transazioni ed il successivo ripristino del Sistema Operativo (boot).
- Quindi viene attivata una procedura di ripresa a caldo o a freddo a sconda del tipo di guasto

Modello fail-stop



Recovery

- Checkpoint
  - Istante di tempo "consistente" (in cui tutte le transazioni scrivono i loro dati dal buffer al disco)
  - Si registrano le transazioni attive
- Dump
  - Istante di tempo in cui viene effettuata una copia completa della base di dati (tipicamente di notte oppure alla fine della settimana);

Ripresa a caldo

- Si leggono le registrazioni del giornale a partire dal checkpoint
- Si separano le transazioni in:
  - INSIEME UNDO (da disfare)
  - INSIEME REDO (da rifare)
- Si eseguono le azioni di UNDO e REDO

Ripresa a freddo

- Si ripristinano i dati a partire dal backup
- Si eseguono le operazioni registrate sul giornale fino all'istante del guasto
- Si esegue una ripresa a caldo