

TRANSAZIONI

Una transazione è una successione di operazioni che si può concludere con successo o con insuccesso.

Nel caso di successo, i risultati delle operazioni effettuate devono essere resi definitivi; invece, nel caso d'insuccesso, non deve rimanere traccia alcuna delle operazioni.

Una transazione deve godere delle seguenti proprietà (dette *proprietà acide*):

Atomicità: La transazione è un'unità indivisibile d'esecuzione, cioè essa deve essere eseguita del tutto oppure non eseguita affatto.

Consistenza: La transazione deve far passare i dati (la base di dati) su cui opera da uno stato consistente ad un altro. Questo significa che, al termine della transazione, qualunque sia il suo esito, tutti i dati devono ancora soddisfare il complesso di vincoli stabiliti dal progettista.

Isolamento: Due o più transazioni devono svolgere il proprio compito indipendentemente le une dalle altre, senza alcuna interferenza. In particolare, questa proprietà impone che l'esecuzione concorrente di più transazioni conduca la base di dati allo stesso stato in cui si sarebbe giunti se le transazioni fossero state eseguite singolarmente.

Durabilità (o persistenza): I risultati prodotti da una transazione andata a buon fine non devono essere persi per alcun motivo.

TRANSAZIONI

Una transazione deve essere atomica, nel senso che, nel caso in cui essa non vada a buon fine, le risorse utilizzate devono essere ripristinate come se la transazione non sia mai avvenuta.

Se la transazione va a buon fine, si eseguono le operazioni di **commit**, che rendono permanenti le modifiche apportate, altrimenti si eseguono le operazioni di **abort**.

Primitive	Description
BEGIN_TRANSACTION	Mark the start of a transaction
END_TRANSACTION	Terminate the transaction and try to commit
ABORT_TRANSACTION	Kill the transaction and restore the old values
READ	Read data from a file, a table, or otherwise
WRITE	Write data to a file, a table, or otherwise

Primitive per le transazioni

BEGIN_TRANSACTION reserve WP → JFK; reserve JFK → Nairobi; reserve Nairobi → Malindi; END_TRANSACTION (a)	BEGIN_TRANSACTION reserve WP → JFK; reserve JFK → Nairobi; Nairobi → Malindi full ABORT_TRANSACTION (b)
--	--

(a) Transaction to reserve three flights commits

(b) Transaction aborts when third flight is unavailable

Una transazione può prevedere una serie di transazioni annidate. In tale situazione, nel caso in cui la transazione globale non vada a buon fine, devono essere rese nulle le operazioni di tutte le transazioni annidate (anche se queste ultime hanno avuto successo).

TRANSAZIONI

L'atomicità può essere garantita attraverso due metodi:

- ☞ Creazione di un workspace privato (*Private workspace*);
- ☞ Introduzione di un file di log (*Writeahed log*).

Il metodo dello spazio di lavoro privato consiste nell'eseguire una copia della risorsa sulla quale si opera (file, database o altro) e nell'eseguire la transazione sulla copia. Quest'ultima sarà resa permanente solo all'atto della commit, attraverso la sostituzione della risorsa originale con la sua copia modificata.

I problemi legati all'uso della risorsa da parte di più utenti si risolvono con dei semafori.

Le transazioni annidate operano sulla copia della copia della risorsa. Se la transazione annidata esegue la commit, le sue modifiche operano solo sulla prima copia della risorsa. La modifica permanente della risorsa può avvenire solo tramite operazione di commit da parte della transazione primaria.

TRANSAZIONI

Il metodo del file di log si basa sull'aggiornamento di un file, attraverso il quale si tiene traccia di tutte le modifiche apportate ad una risorsa durante una transazione.

Ogni volta che viene apportata una modifica, viene registrato un record contenente le seguenti informazioni:

- ☞ codice della transazione,
- ☞ codice del campo modificato,
- ☞ vecchio valore del campo,
- ☞ nuovo valore del campo.

Se la transazione non va a buon fine, è possibile ripristinare il contenuto della risorsa leggendo all'incontrario il file di log.

Il file di LOG è unico per tutto il sistema.

Quest'ultimo approccio si usa anche nella gestione del dead-lock (all'atto dell'uccisione di uno dei processi che contribuisce allo stallo) o quando in generale si opera il ripristino della copia di una risorsa (CHECK-POINT/RESTART).

TRANSAZIONI

Confronto tra le due tecniche

Il metodo dello spazio di lavoro privato è più facile dal punto di vista della gestione, ma può comportare grossa occupazione di memoria.

Per questo motivo, la copia della risorsa non viene effettuata né all'atto della "begin transaction" né all'atto dell'operazione di "read", ma viene effettuata solo quando si presenta un'operazione di "write". Inoltre, all'atto della modifica, si copia in memoria solo la parte del file o data-base alla quale si riferisce l'operazione. In tal modo, si riduce il problema del grosso costo, in termini di memoria, richiesto da questo metodo.

Il metodo del file di log è più laborioso nella gestione, ma non comporta grossa occupazione di memoria.

TRANSAZIONI

Commit a due fasi

Cosa accade se una transazione, che è un processo indipendente, lancia altri processi a loro volta transattivi?

L'approccio è sostanzialmente analogo anche quando coinvolge transazioni che operano in un sistema distribuito.

Il processo chiamante (coordinatore) può eseguire la commit solo quando tutti gli altri processi chiamati hanno eseguito, a loro volta, la commit.

In questo caso, è indispensabile l'uso del metodo writeahead log.

Il processo coordinatore scrive sul file di log che è pronto per eseguire la commit e richiede agli altri processi di notificare la possibilità di eseguire la loro operazione di commit. Ciascuno degli altri processi può, quindi, scrivere sul file di log che è pronto per la commit e segnala al processo coordinatore il benessere per la commit. Invece, se uno dei processi non risponde, il processo coordinatore non può effettuare la sua commit ed abortisce la transazione.

La tecnica vista va sotto il nome di “commit a due fasi”.

TRANSAZIONI

Concorrenza delle transazioni

Nel caso in cui due applicazioni transattive operino sulla stessa risorsa, si possono avere problemi di corsa tipici della “concorrenza”.

Il controllo della concorrenza si effettua tramite una delle seguenti tecniche:

☞ *locking,*

☞ *timestamping.*

TRANSAZIONI

Locking.

Questa tecnica consiste nell'associare a ciascuna risorsa condivisa una variabile accessibile in modo atomico da tutte le transazioni.

Tale variabile indica se la risorsa è libera oppure in uso ed esprime il tipo d'accesso consentito.

Esistono due tipi di lock: condiviso (o in lettura) ed esclusivo (o in scrittura).

Quando una transazione vuole accedere ad un dato con lo scopo di leggerlo, richiede su di esso un *lock condiviso*. Se altre transazioni vogliono leggere quello stesso dato, possono anch'esse richiedere un lock condiviso senza che il sistema transazionale lo neghi.

Nel caso in cui una transazione voglia scrivere un dato, deve richiedere un *lock esclusivo*. Se tale richiesta viene effettuata per un dato su cui altre transazioni hanno fissato un lock condiviso, essa verrà negata dal sistema transazionale.

Uno dei protocolli di locking più utilizzati (per evitare la monopolizzazione delle risorse) è il locking a due fasi. Tale protocollo prevede che tutte le richieste di locking precedano la prima operazione di unlock (rimozione di un lock). Questo consente di dividere la transazione in una fase d'espansione (in cui vengono stabiliti i lock sulle risorse) ed una fase di contrazione (in cui i lock vengono rimossi).

TRANSAZIONI

Timestamping.

Questa tecnica consiste nell'associare a ciascuna transazione un valore numerico in genere con significato temporale detto timestamp (ricavato dal clock di sistema).

Ad ogni risorsa cui le transazioni accedono, sono associati due indicatori $WTM(x)$ e $RTM(x)$ che sono rispettivamente il timestamp della transazione che ha eseguito l'ultima scrittura ed il timestamp più grande tra quelli delle transazioni che hanno letto x .

Le primitive d'accesso ai dati sono del tipo $read(x,ts)$ oppure $write(x,ts)$.

Il sistema transazionale si comporta nel modo seguente:

- ↳ nel caso di una $read(x,ts)$, se $ts < WTM(x)$ (cioè, la transazione corrente vuole leggere un dato scritto da una transazione più giovane), la transazione viene uccisa; in caso contrario, la richiesta viene accettata ed il valore di $RTM(x)$ viene aggiornato al massimo tra $RTM(x)$ e ts .
- ↳ nel caso di una $write(x,ts)$, se $ts < WTM(x)$ o $ts < RTM(x)$ (cioè, la transazione corrente cerca di scrivere su di un dato già scritto o già letto da una transazione più giovane), la transazione viene uccisa, altrimenti la richiesta viene accettata e $WTM(x)$ viene posto uguale a ts . Questo meccanismo è meno efficace del locking, ma evita il problema del deadlock.

TRANSAZIONI

Nei sistemi transazionali facenti uso di politiche di locking, uno dei problemi ricorrenti è il blocco critico (deadlock). Quando una transazione A detiene una risorsa e ne attende un'altra posseduta da una transazione B, che rilascerà solo avendo a disposizione la risorsa detenuta da A, si determina una situazione che senza intervento esterno non potrà evolvere.

Una modalità per risolvere il problema del deadlock è quella del timeout. Quando una transazione richiede una risorsa che per qualche motivo non può avere, viene fissato un tempo massimo d'attesa scaduto il quale la transazione viene uccisa. Quest'ultima verrà fatta ripartire successivamente.

E' evidente che il sistema non può effettivamente rilevare una situazione di deadlock, ma presume che attese troppo lunghe siano valido sintomo di deadlock. Il valore dell'intervallo di timeout deve essere fissato con molta cura, poichè un tempo troppo lungo risolverebbe tardi una situazione di deadlock, mentre un tempo troppo corto fa apparire come deadlock delle situazioni che in realtà non lo sono.

Il vantaggio maggiore di questa tecnica è l'estrema semplicità implementativa.