ISTITUTO TECNICO INDUSTRIALE
G. M. ANGIOY
SASSARI

PROGETTAZIONE DI DATABASE

Le Transazioni

Lezione 15

Versione del: 29/04/2015
Revisione numero: 3

Prof. Andrea Zoccheddu
Dipartimento di Informatica

Immagine di copertina da: http://www.iamsterdam.com/en-GB/living/education/Dutch-Education-System

DIPARTIMENTO
INFORMATICA E TELECOMUNICAZIONI

LE TRANSAZIONI

GENERALITÀ DELLE TRANSAZIONI

LINK

Fonte: http://it.wikipedia.org/wiki/Transazione_%28basi_di_dati%29

Definizione

In informatica, una transazione è una sequenza di operazioni, che può concludersi con un successo o un insuccesso; in caso di successo, il risultato delle operazioni deve essere permanente (= duraturo = persistente), mentre in caso di insuccesso si deve tornare allo stato precedente all'inizio della transazione.

Le transazioni sono normalmente implementate da DBMS o da gestori di transazioni (application server o ambienti direttamente installati sulla macchina host dove risiede il database (es. CICS)).

CARATTERISTICHE DELLE TRANSAZIONI

Nei linguaggi di accesso ai DBMS, la gestione delle transazioni fa parte del DML (Data Manipulation Language o linguaggio di manipolazione dei dati). Infatti, le modifiche allo schema del database o alle autorizzazioni non sono facilmente gestibili con transazioni.

Un utilizzo tipico delle transazioni è il seguente:

1. Prima di eseguire una transazione, si esegue un'istruzione di "inizio transazione".
2. Si eseguono le operazioni di interrogazione e modifica dei dati.
3. Se si riscontra qualche anomalia, si esegue un'istruzione detta di "rollback", per abortire la transazione.
4. Se si sono eseguite tutte le operazioni senza riscontrare anomalie, si esegue un'istruzione detta di "commit", per confermare la transazione.

Inizio Transazione

Il BEGIN TRANS è un comando che serve per delimitare fisicamente l'inizio dei comandi interni alla transazione. Alcuni sistemi non prevedono un'istruzione di inizio transazione, perché quando ci si collega al DBMS, si inizia automaticamente una transazione, e quando si esegue un commit o un rollback, si inizia automaticamente un'altra transazione.

Rollback

Il rollback deve disfare integralmente la transazione in corso; qualsiasi modifica effettuata tra l'inizio della transazione e il comando di rollback deve essere annullata e non deve portare ad alcuna conseguenza o effetto.

- se il DBMS rileva una sollevazione di eccezione per violazione dei vincoli (chiave primaria, integrità referenziale, obbligatorietà, unicità di valore, ecc);
- se il DBMS riscontra internamente qualche anomalia, esegue automaticamente un rollback;
- se il DBMS stesso termina bruscamente, per intervento esterno, o per un bug, o per spegnimento improvviso del computer, il DBMS, quando viene riattivato, esegue automaticamente il rollback delle transazioni che erano in corso al momento del crash;
- se ci si scollega dal DBMS senza eseguire un commit, alcuni DBMS eseguono automaticamente un commit, altri un rollback.

Per implementare un transazione, tipicamente si usa un'apposita area d'appoggio del disco fisso in cui vengono copiati i dati originali appena prima di essere modificati. Quando viene eseguito un commit, i dati originali copiati vengono eliminati. Quando viene eseguito un rollback, si ricopiano indietro i dati originali copiati. Pertanto, un commit è più efficiente di un rollback.

Una possibile causa del fallimento di una transazione è l'insufficienza di spazio d'appoggio in memoria per copiare i dati originali.

Commit

Il commit deve salvare permanentemente la transazione conclusa con successo; qualsiasi modifica effettuata tra l'inizio della transazione e il comando di commit deve essere confermata e i suoi effetti diventano duraturi.

GESTORE DELLE TRANSAZIONI

LINK

Fonte: http://it.wikipedia.org/wiki/Gestore_di_transazioni

In informatica, un `Gestore di transazioni`, o Transaction Manager, è un modulo del DBMS che garantisce che le transazioni godano delle proprietà ACID.

Il gestore di transazioni svolge il proprio compito coordinando gli altri gestori risorse, ossia il Lock Manager, lo Scheduler, il Recovery Manager, il Buffer Manager e il Log Manager.

I principali aspetti di cui si occupa il gestore di transazioni sono:

- Controllo della concorrenza.
- Protezione dai guasti.

Il Gestore di transazioni accetta comunemente 4 differenti tipi di richieste:

- **BEGIN(TRANSACTION)**: richiesta di inizio di una transazione. Questa giunge in seguito ad una richiesta SQL.
- **COMMIT**: è la segnalazione al transaction manager che la transazione richiesta è andata a buon fine. Il commit richiede inoltre che gli effetti prodotti sulla base di dati vengano resi permanenti.
- **ROLLBACK/ABORT**: è la segnalazione al transaction manager che si sono verificati uno o più problemi con possibilità di stato inconsistente della base di dati.
- **READ/WRITE**: richiesta di accesso ai dati.

PROPRIETÀ DELLE TRANSAZIONI

LINK

Fonte: http://it.wikipedia.org/wiki/ACID

Una transazione, per essere tale, deve godere delle cosiddette proprietà ACID, particolarmente significative nei sistemi in cui possono essere eseguite più transazioni contemporaneamente.

Nell'ambito dei database, ACID deriva dall'acronimo inglese Atomicity, Consistency, Isolation, e Durability (Atomicità, Consistenza, Isolamento e Durabilità) ed indica le proprietà logiche che devono avere le transazioni. Perché le transazioni operino in modo corretto sui dati è necessario che i meccanismi che le implementano soddisfino queste quattro proprietà:

- **atomicità**: la transazione è indivisibile nella sua esecuzione e la sua esecuzione deve essere o totale o nulla, non sono ammesse esecuzioni parziali;
- **consistenza**: quando inizia una transazione il database si trova in uno stato consistente e quando la transazione termina il database deve essere in un altro stato consistente, ovvero non deve violare eventuali vincoli di integrità, quindi non devono verificarsi contraddizioni (inconsistenza) tra i dati archiviati nel DB;
- **isolamento**: ogni transazione deve essere eseguita in modo isolato e indipendente dalle altre transazioni, l'eventuale fallimento di una transazione non deve interferire con le altre transazioni in esecuzione;
- **durabilità**: detta anche persistenza, si riferisce al fatto che una volta che una transazione abbia richiesto un commit work, i cambiamenti apportati non dovranno essere più persi. Per evitare che nel lasso di tempo fra il momento in cui la base di dati si impegna a scrivere le modifiche e quello in cui li scrive effettivamente si verifichino perdite di dati dovuti a malfunzionamenti, vengono tenuti dei registri di log dove sono annotate tutte le operazioni sul DB.

I comandi DML (per es. query, inserimenti, aggiornamenti, cancellazioni) ed altre azioni (procedure) vengono raggruppate in una transazione che deve essere eseguita atomicamente, isolatamente dalle altre e comportando eventualmente una modifica permanente del database.

Nei sistemi di DBMS le transazioni vengono processate dal **transaction manager** che sovrintende al **transaction processing**. Al suo interno i seguenti moduli sono:

- Concurrency Control Manager o WorkSpace Privato che garantisce l'atomicità e isolamento
- Logging / Recovery Manager che garantisce la durabilità e consistenza.

Sebbene non sia possibile in generale garantire che tutte le transazioni siano sempre portate a termine con successo, è però possibile assicurarne le proprietà acide. Le operazioni fondamentali, a tale proposito, sono le operazioni di commit e rollback (o abort):

- un'operazione **commit** segnala il completamento con esito positivo di una transazione. La transazione è stata eseguita senza errori e tutti i cambiamenti apportati dalla transazione sono stati resi permanenti.
- un'operazione **rollback** segnala il fallimento di una transazione. La base di dati potrebbe non essere in uno stato consistente e tutti i cambiamenti apportati dalla transazione devono essere disfatti.

ANOMALIE DELLE TRANSAZIONI CONCORRENTI (VIOLAZIONE DI ISOLAMENTO)

LINK

Fonte: http://users.dimi.uniud.it/~angelo.montanari/SQL-Transazioni.pdf

Una gestione non controllata della concorrenza può causare il verificarsi di diversi tipi di anomalie.

Perdita d'aggiornamento:
Accade quando due transazioni concorrenti sono eseguite in modo tale che il valore di qualche dato viene modificato in modo non corretto. Un esempio di sequenza concorrente con perdita d'aggiornamento è il seguente:

Warning

Esempio: si supponga di avere due transazioni concorrenti indicate con T1 e T2; entrambe elaborano dati e in particolare accedono entrambe alla locazione X e si prefiggono di incrementarla di +1:

1. T1 legge il valore della locazione X (vale 1)
2. T2 legge il valore della locazione X (vale 1)
3. T1 calcola X+1 e aggiorna il valore della locazione X (vale 2)
4. T2 calcola X+1 e aggiorna il valore della locazione X (vale 2)

Questa situazione rappresenta un caso in cui non è garantito l'isolamento delle transazioni e che ha condotto ad un errore: alla fine di T1 e T2 la locazione X dovrebbe valere 3 ed invece vale 2.

Dipendenza da transazioni non committed (lettura sporca):
Accade quando una transazione legge dati scritti da una transazione concorrente attiva. Se quest'ultima transazione fallisce, i dati letti dalla prima non sono piú validi. Un esempio di tale lettura sporca è il seguente:

Warning

Esempio: si supponga di avere due transazioni concorrenti indicate con T1 e T2; entrambe elaborano dati e in particolare accedono entrambe alla tupla r1:

1. T1 legge X e lo incrementa di +1 (adesso vale 2)
2. T2 accede a X in lettura (userà il valore 2 per altre azioni)
3. T1 termina con un rollback (r1 ritorna allo stato originario prima dell'azione **1.**)
4. T2 ha avuto accesso ad un dato che non è mai esistito

Questa situazione rappresenta un caso in cui non è garantito l'isolamento delle transazioni e che può condurre ad un errore.

Analisi inconsistente (aggiornamento fantasma e lettura inconsistente):
Accade quando una transazione legge dati scritti da una transazione concorrente attiva. Se quest'ultima transazione fallisce, i dati letti dalla prima non sono piú validi. Un esempio di tale lettura sporca è il seguente:

Warning

Esempio: si supponga di avere due transazioni concorrenti indicate con T1 e T2; entrambe elaborano dati e in particolare accedono entrambe alla tupla r1:

1. T1 legge X e lo incrementa di +1 (adesso vale 2)
2. T2 accede a X in lettura (userà il valore 2 per altre azioni)
3. T1 termina con un rollback (r1 ritorna allo stato originario prima dell'azione **1.**)
4. T2 ha avuto accesso ad un dato che non è mai esistito

Questa situazione rappresenta un caso in cui non è garantito l'isolamento delle transazioni e che può condurre ad un errore.

Si consideri una transazione che calcola il valore di una funzione aggregata. Durante il calcolo un'altra transazione modifica alcuni dei record coinvolti nel calcolo, cosicché il valore finale calcolato dalla funzione non è corretto. Tale anomalia prende il nome di **aggiornamento fantasma**.

Warning

Esempio: Si consideri, ad esempio, una transazione che calcoli la somma degli studenti di tre classi; le classi sono indicate con lettere X, Y e W con valore iniziale X=30, Y=25, W=20. La somma dovrebbe essere 75, ma mentre T1 effettua la somma, T2 sposta 2 studenti da X a W:

1. T1 prepara la somma S=0
2. T1 legge X e lo somma a S (S=30)
3. T2 sposta 2 studenti da X a W (adesso X=28, W=22)
4. T1 legge Y e lo somma a S (S=55)
5. T1 legge W e lo somma a S (S=77)

Questa situazione rappresenta un caso in cui non è garantito l'isolamento delle transazioni.
T1 ha ottenuto la somma di 77 alunni mentre invece la somma corretta doveva dare 75.

Una lettura inconsistente (o non ripetibile) può avvenire quando una transazione legge due volte lo stesso dato X in momenti successivi e, tra le due letture, un'altra transazione (che va a buon fine) modifica X, cosicché la prima transazione legge due valori diversi per lo stesso dato X.

Warning

Esempio: Nuovamente le classi sono indicate con lettere X, Y e W con valore iniziale X=30, Y=25, W=20. La transazione T1 però deve sommare tre volte sempre il valore di X; la somma dovrebbe essere 90, ma mentre T1 effettua la somma, T2 sposta 5 studenti da X a W:

1. T1 prepara la somma S=0
2. T1 legge X e lo somma a S (S=30)
3. T2 sposta 2 studenti da X a W (adesso X=25, W=25)
4. T1 legge X e lo somma a S (S=55)
5. T1 legge X e lo somma a S (S=80)

Questa situazione rappresenta un caso in cui non è garantito l'isolamento delle transazioni.
T1 ha ottenuto la somma di 80 alunni mentre invece la somma corretta doveva dare 90.

Si noti che in un sistema in cui le letture inconsistenti sono impedite anche il problema dell'aggiornamento fantasma è evitato.

Le anomalie fin qui illustrate riguardano tutti situazioni in cui le transazioni manipolano dati già presenti nella base di dati. Nella pratica, tuttavia, alcune anomalie possono presentarsi a causa di interrogazioni che sono eseguite in concorrenza a operazioni d'inserimento (INSERT).

BLOCCO IN ACCESSO

LINK

Fonte: http://users.dimi.uniud.it/~angelo.montanari/SQL-Transazioni.pdf

Per evitare anomalie come quelle illustrate in precedenza è possibile usare un blocco in accesso ai dati, che tuttavia può portare ad altri inconvenienti.
Il blocco in accesso ai dati significa che quando una transazione intende leggere/scrivere sui dati, il dato viene bloccato e altre eventuali transazioni che lo richiedano devono attendere che la transazione la rilasci.

Esempio

Esempio: poniamo che le transazioni T1 e T2 debbano entrambe accedere alla locazione X per incrementarla di +1. Esse hanno una sequenza simile alla seguente:

T1
Legge X
Calcola X+1
Scrive X aggiornato

T2
Legge X
Calcola X+1
Scrive X aggiornato

ogni transazione dovrebbe bloccare X e l'altra deve attendere che sia rilasciata prima di accedervi.

Senza il blocco si rischia una anomalia come già visto in precedenza:

Warning

Esempio: ecco l'anomalia senza usare il blocco:

1. T1 legge X (vale 0)
2. T2 legge X (vale 0)
3. T1 aggiorna X (vale 1)
4. T2 aggiorna X (vale 1)

X valeva zero e dopo due transazioni invece di valere 2 vale solo 1.

La sequenza si può vedere anche col seguente grafico:

Warning

Esempio: ecco l'anomalia senza usare il blocco:

	T1		T2				
1	Legge X						
2			Legge X				
3	Calcola X+1						
4	Scrive X aggiornato						
5			Calcola X+1				
6			Scrive X aggiornato				

X valeva zero e dopo due transazioni invece di valere 2 vale solo 1.

Con il blocco si evita la anomalia:

Esempio

Esempio: ecco la sequenza col blocco

1. T1 blocca X
2. T1 legge X (vale 0)
3. T2 richiede X ma non può averla (va in attesa WAITING)
4. T1 aggiorna X (vale 1)
5. T1 libera X e consente alla prima transazione in attesa di accedere alla locazione
6. T2 blocca X
7. T2 legge X (vale 1)
8. T2 aggiorna X (vale 1)
9. T2 libera X

X valeva zero e dopo due transazioni invece di valere 2 vale solo 1.

La sequenza si può vedere anche col seguente grafico:

Warning

Esempio: ecco la sequenza col blocco

	T1		T2
1	Richiede X		
2		2	Richiede X va in attesa
3	Blocca X		
4	Legge X		
5	Calcola X+1		
6	Scrive X aggiornato		
7	Libera X		
8		8	Blocca X
9		9	Legge X
10		10	Calcola X+1
11		11	Scrive X aggiornato
12		12	Libera X

X valeva zero e dopo due transazioni invece di valere 2 vale solo 1.

Tuttavia la soluzione del blocco rischia altri problemi.

Con il blocco si rischia lo stallo:

Esempio

Esempio: consideriamo il caso di due transazioni, entrambe facenti uso di due risorse che chiamiamo X e Y

1. T1 richiede e ottiene e blocca X
2. T1 agisce su X
3. T2 richiede e ottiene e blocca Y
4. T2 agisce su Y
5. T1 richiede Y ma non può averla (va in attesa WAITING)
6. T2 richiede X ma non può averla (va in attesa WAITING)

STALLO (DEADLOCK) le due transazioni sono in stallo poiché attendono all'infinito la risorsa richiesta.

GESTIONE DELL'ISOLAMENTO

LINK

Fonte: http://users.dimi.uniud.it/~angelo.montanari/SQL-Transazioni.pdf

Esula da questa dispensa la risoluzione di accessi concorrenti di risorse richieste da transazioni; tuttavia in esistono modalità che assicurano la mancanza di anomalie e in generale consentono la concorrenza senza stalli e senza attese indefinite (STARVATION).

Lo standard SQL individua quattro livelli d'isolamento, che definiscono diversi "gradi d'interferenza" tra transazioni sulla base di quali, tra le seguenti tre anomalie, devono essere proibite: letture sporche, letture inconsistenti e fantasmi.

SQL

I quattro livelli d'isolamento e i corrispondenti requisiti sono riassunti nella seguente tabella:

Livello di Isolamento	Letture sporche	Letture inconsistenti	Fantasmi
READ UNCOMMITTED	⚠ Possibili	⚠ Possibili	⚠ Possibili
READ COMMITTED	✓ Vietate	⚠ Possibili	⚠ Possibili
REPEATABLE READ	✓ Vietate	✓ Vietate	⚠ Possibili
SERIALIZABLE	✓ Vietate	✓ Vietate	✓ Vietate

Nella tabella si vede come la gestione più sicura è quella denominata `SERIALIZABLE` che tuttavia è la più restrittiva per quanto concerne la concorrenza, situazione che può rallentare notevolmente il sistema.

La scelta un po' meno sicura è detta `REPEATABLE READ` che rischia di ammettere Fantasmi, ma consente un certo margine di parallelismo.

La scelta detta `READ COMMITTED` rischia di ammettere sia Fantasmi sia Letture inconsistenti, ma consente un discreto margine di parallelismo.

La scelta detta `READ UNCOMMITTED` ammette sia Fantasmi sia Letture inconsistenti ed anche Letture sporche, e consente il più ampio margine di parallelismo.

L'idea di ammettere vari livelli SQL è che in certe applicazioni è accettabile rinunciare alla serializzabilità e preferire un aumento del grado di concorrenza del sistema. In tal caso, tipicamente i DBMS consentono all'utente di specificare i lock in modo esplicito nei casi in cui il sistema non è in grado di garantire una corretta esecuzione concorrente delle transazioni.

SOMMARIO

GENERALITÀ DELLE TRANSAZIONI 2
Caratteristiche delle transazioni 2
Inizio Transazione 2
Rollback 2
Commit 2
Gestore delle Transazioni 3
Proprietà delle transazioni 3
Anomalie delle transazioni concorrenti (violazione di Isolamento) 4
Perdita d'aggiornamento: 4
Dipendenza da transazioni non committed (lettura sporca): 4
Analisi inconsistente (aggiornamento fantasma e lettura inconsistente): 4
Blocco in accesso 5
GESTIONE DELL'ISOLAMENTO 7