BD 2 - Transazioni

Luca Cosmo

Università Ca' Foscari Venezia

Università
Ca'Foscari
Venezia

Introduzione

Quando programmiamo applicazioni che si interfacciano con un database, è normale raggruppare insieme una sequenza di operazioni su di esso per implementare una determinata funzionalità.

Questa pratica necessaria può creare problemi quando:

- ci sono molte operazioni concorrenti sulla base di dati, per esempio perchè più persone stanno contemporaneamente prenotando un posto a sedere sullo stesso volo
- certe operazioni falliscono e non possono essere completate, per esempio perchè troppe persone sono attive nello stesso momento

Entrambi questi scenari possono danneggiare l'integrità della base di dati.

Concorrenza

Un utente *u* vuole riservare un posto sul proprio volo:

```
SELECT seatNo
FROM Flights
WHERE fltNo = 123 AND fltDate = DATE '2020-03-07'
                  AND seatStatus = 'available'
```

Una volta scoperto che il posto 22A è libero, *u* procede alla prenotazione:

```
UPDATE Flights
SET seatStatus = 'occupied'
WHERE fltNo = 123 AND fltDate = DATE '2020-03-07'
                  AND seatNo = '22A'
```

Concorrenza = Problemi!

Ma anche l'utente v vuole prenotare un posto sullo stesso volo e proprio nello stesso momento!

u esegue la SELECT 22A è libero	
	v esegue la SELECT 22A è libero
u esegue l'UPDATE 22A diventa occupato	
	v esegue l'UPDATE 22A diventa occupato di nuovo!

Risultato: u e v hanno lo stesso posto assegnato! Chi resta a terra?!

Fallimenti

Visto che la compagnia aerea ha lasciato il posto 22A a u, il biglietto di v va rimborsato. Si prelevano 800 euro dal conto della compagnia:

```
UPDATE Accounts
SET balance = balance - 800
WHERE acctNo = 123;
```

e si accreditano poi sul conto di v:

```
UPDATE Accounts
SET balance = balance + 800
WHERE acctNo = 456;
```

Cosa succede nel caso di un crash dopo la prima operazione?

Transazioni

Una transazione è una sequenza di operazioni sul database che soddisfa le seguenti proprietà:

1. serializzabilità: l'esecuzione concorrente di più transazioni è equivalente ad una loro esecuzione seriale in un qualche ordine
2. atomicità: se la transazione termina prematuramente, tutti i suoi effetti parziali sono annullati
3. persistenza: le modifiche effettuate da una transazione terminata con successo sono permanenti

In certi libri di testo una presentazione leggermente diversa: ACID, un popolare acronimo per Atomicity, Consistency, Isolation e Durability.

Serializzabilità

Questa esecuzione non è più ammissibile, perché viola la condizione di serializzabilità delle transazioni:

u esegue la SELECT 22A è libero	
	v esegue la SELECT 22A è libero
u esegue l'UPDATE 22A diventa occupato	
	v esegue l'UPDATE 22A diventa occupato di nuovo

Non è possibile che due utenti trovino il posto 22A libero se le due transazioni sono eseguite una dopo l'altra!

Serializzabilità

Questa esecuzione è invece ammissibile rispetto alla condizione di serializzabilità, anche se ha favorito v rispetto ad u:

u esegue la SELECT 22A è occupato	v esegue la SELECT 22A è libero v esegue l'UPDATE 22A diventa occupato

Non importa chi è penalizzato fra u e v, l'importante è che uno dei due sia costretto a trovarsi un altro posto a sedere!

Atomicità

Supponiamo che avvenga un crash dopo aver prelevato 800 euro dal conto della compagnia:

```
UPDATE Accounts
SET balance = balance - 800
WHERE acctNo = 123;
```

La seguente operazione nella stessa transazione non verrà più eseguita:

```
UPDATE Accounts
SET balance = balance + 800
WHERE acctNo = 456;
```

Alla fine del crash anche la prima operazione sarà pertanto annullata.

Programmare Transazioni

Normalmente ogni operazione SQL è gestita come una transazione indipendente. E' possibile però usare i seguenti comandi:

1. `START TRANSACTION` indica l'inizio di una nuova transazione
2. `COMMIT` indica la terminazione corretta di una transazione: tutto ciò che è stato fatto durante la transazione deve essere reso persistente
3. `ROLLBACK` indica la terminazione anomala di una transazione: tutto ciò che è stato fatto durante la transazione deve essere annullato

Tipicamente API come JDBC offrono metodi che permettono di gestire le transazioni, che si appoggiano a questi comandi SQL.

Transazioni e Vincoli

Data l'atomicità delle transazioni, è possibile differire il controllo di alcuni vincoli di integrità alla fine di una transazione.

Ciascun vincolo può appartenere ad una fra tre categorie:

- `NOT DEFERRABLE`: viene sempre controllato dopo ogni operazione
- `DEFERRABLE INITIALLY IMMEDIATE`: viene controllato dopo ogni operazione della transazione, ma è possibile rilassarlo per farlo controllare solo prima del commit
- `DEFERRABLE INITIALLY DEFERRED`: viene controllato solo prima del commit, ma è possibile rafforzarlo per farlo controllare dopo ogni operazione della transazione

I vincoli differibili possono essere configurati tramite `SET CONSTRAINTS`

Implementazione di Transazioni

Una transazione è una sequenza di operazioni serializzabile. Quanto ci costa ciò in termini di performance?

- soluzione banale: ciascuna transazione prende un lock globale sul database, che viene rilasciato solo dopo commit o rollback
- ottimizzazioni: insieme di lock locali, che predicano solo su porzioni del database, e gestione rilassata delle transazioni read only, che non possono compromettere l'integrità della base di dati
- nota: i DBMS moderni usano soluzioni senza lock come MVCC (multiversion concurrency control)
- i DBMS principali forniscono ulteriore controllo al programmatore definendo diversi livelli di isolamento fra transazioni

Rilassare il livello di isolamento ideale (SERIALIZABLE) può portare ad un incremento delle prestazioni, ma è in generale rischioso.

Transazioni Read Only

Una transazione può essere dichiarata read only tramite la sintassi:

```
SET TRANSACTION READ ONLY;
```

L'effetto di tale dichiarazione è il seguente:

- la transazione può solo leggere dati (SELECT), ma non scriverli
- tutte le query nella transazione possono vedere solo le scritture committate prima dell'inizio della transazione
- più transazioni read only che operano sugli stessi dati possono essere eseguite concorrentemente senza rischi per l'integrità dei dati!

Una transazione read only fornisce una lettura consistente dello stato del database prima dell'inizio della transazione.

Transazioni Read Uncommitted

Il livello di isolamento READ UNCOMMITTED consente ad una transazione di leggere dirty data, cioè dati scritti da altre transazioni che non hanno ancora fatto commit:

- quando ciò si verifica, si parla di dirty read
- il rischio di una dirty read è che la transazione che ha scritto i dirty data potrebbe abortire: in tal caso, i dirty data dovrebbero essere rimossi e non dovrebbero influenzare le altre transazioni

SQL limita l'uso di READ UNCOMMITTED alle sole transazioni read only, a meno che lo sviluppatore non decida di rilassare questo vincolo.

Dirty Reads: Esempio

Consideriamo la seguente definizione di una transazione per effettuare un bonifico fra due conti correnti:

1. Accredita immediatamente l'importo nel conto 2
2. Controlla se il conto 1 conteneva abbastanza denaro:
   - (s) In caso positivo, addebita l'importo nel conto 1 (commit)
   - (f) In caso negativo, annulla l'accredito nel conto 2 (rollback)

Consideriamo due transazioni eseguite concorrentemente:

1. T_1 trasferisce 150 dal conto A_1 al conto A_2
2. T_2 trasferisce 250 dal conto A_2 al conto A_3

Dirty Reads: Esempio

T_1 trasferisce 150 da A_1 ad A_2
T_2 trasferisce 250 da A_2 ad A_3

A_1	A_2	A_3
100	200	300
T_2 esegue il passo 1: accredita 250 ad A_3		
100	200	550
T_1 esegue il passo 1: accredita 150 ad A_2		
100	350	550
T_2 esegue il passo $2s$: addebita 250 ad A_2		
100	100	550
T_1 esegue il passo $2f$: anulla accredito di 150 ad A_2		
100	-50	550

Transazioni Read Committed

Il livello di isolamento READ COMMITTED impedisce il fenomeno delle dirty read, fornendo un maggiore isolamento:

- quando una transazione vuole effettuare una scrittura, acquisisce un lock che viene rilasciato solo dopo la sua terminazione
- si può verificare il fenomeno di unrepeatable read: due letture degli stessi dati in momenti diversi possono portare a risultati diversi a causa dell'intervento di un'altra transazione
- si può verificare il fenomeno di lost update, cioè la perdita di una modifica da parte di una transazione causata da un aggiornamento operato da un'altra transazione

Unrepeatable Read: Esempio

Esempio: calcolo della media dei voti di tutti gli studenti in presenza di una cancellazione concorrente

T_1	T_2
Inizio della transazione	
Somma i voti di tutti gli studenti	Inizio della transazione
(Aspetta causa lock)	Cancella uno studente, lock su tabella acquisito
(Aspetta causa lock)	Commit, lock rilasciato
Calcola il numero degli studenti	
Calcola la media via divisione	

La media calcolata è sbagliata!

Lost Update: Esempio

Esempio: vendita concorrente di articoli da un sito di e-commerce

T_1	T_2
Inizio della transazione	
Leggi il numero di articoli disponibili nel magazzino: 12	Inizio della transazione
	Leggi il numero di articoli disponibili nel magazzino: 12
Vendi 3 articoli: il numero di articoli è 9, lock su tabella acquisito	(Aspetta causa lock)
Commit, lock rilasciato	Vendi 4 articoli: il numero di articoli rimanenti è 8

Il numero di articoli rimanenti è sbagliato!

Transazioni Repeatable Read

Il livello di isolamento REPEATABLE READ impedisce il fenomeno delle dirty read, delle unrepeatable read e dei lost update:

- quando una transazione vuole effettuare una lettura oppure una scrittura, acquisisce un lock che viene rilasciato solo dopo la sua terminazione
- per motivi di efficienza, i lock vengono implementati a livello di righe
- si può verificare il fenomeno dei fantasmi: un'altra transazione può aggiungere dati ad una tabella prima che la transazione in corso sia stata completata, andando ad influenzarne il risultato

Prevenzione di Unrepeatable Read

Esempio: calcolo della media dei voti di tutti gli studenti in presenza di una cancellazione concorrente

T_1	T_2
Inizio della transazione	
Somma i voti di tutti gli studenti, lock su righe acquisito	Inizio della transazione
Calcola il numero degli studenti	Prova a cancellare uno studente, ma c'è un lock
Calcola la media via divisione	(Aspetta causa lock)
Commit, lock rilasciato	(Aspetta causa lock)
	Cancella uno studente

La media calcolata stavolta è corretta!

Prevenzione di Lost Update

Esempio: vendita concorrente di articoli da un sito di e-commerce

T_1	T_2
Inizio della transazione	
Leggi il numero di articoli disponibili nel magazzino: 12, lock su righe acquisito	Inizio della transazione
Vendi 3 articoli: il numero di articoli è 9	(Aspetta causa lock)
Commit, lock rilasciato	Leggi il numero di articoli disponibili nel magazzino: 9
	Vendi 4 articoli: il numero di articoli rimanenti è 5

Ora il numero di articoli rimanenti è corretto!

Fantasmi: Esempio

Esempio: calcolo della media dei voti di tutti gli studenti in presenza di un inserimento concorrente

T_1	T_2
Inizio della transazione	
Somma i voti di tutti gli studenti, lock su righe acquisito	Inizio della transazione
Prova a calcolare il numero degli studenti, ma c'è un lock	Inserisci uno studente, lock sulla nuova riga acquisito
(Aspetta causa lock)	Commit, lock rilasciato
Calcola il numero degli studenti	
Calcola la media via divisione	

La media calcolata è di nuovo sbagliata!

Livelli di Isolamento: Riassunto

Ricapitolando, ci sono quattro possibili livelli di isolamento, che offrono diversi livelli di performance e diverse garanzie di integrità

Isolamento	**DR**	**UR**	**LU**	**F**
READ UNCOMMITTED	sì	sì	sì	sì
READ COMMITTED	no	sì	sì	sì
REPEATABLE READ	no	no	no	sì
SERIALIZABLE	no	no	no	no

La scelta finale è delegata al programmatore! Attenzione a non sacrificare la correttezza solo per avere performance migliori...

Interazioni fra Livelli di Isolamento

Il livello di isolamento di una transazione riguarda esclusivamente ciò che può vedere quella transazione, non le altre!

- se T è SERIALIZABLE, la sua esecuzione deve essere equivalente al caso in cui tutte le altre transazioni sono state eseguite interamente prima o interamente dopo T
- se un'altra transazione gira con un diverso livello di isolamento, essa potrebbe vedere immediatamente i dati scritti da T (anche prima della sua terminazione)
- per esempio, se tale transazione è READ UNCOMMITTED, essa può vedere dirty data scritti da T

Livelli di Isolamento: Esempio 1

A che livello di isolamento andrebbe eseguita la seguente transazione?

Example

Per ogni cliente della banca, contare le operazioni effettuate negli ultimi 500 giorni, ed aggiungere il nome del cliente ad un elenco se le operazioni sono più di 300. L'elenco servirà a scopi di marketing.

Livelli di Isolamento: Esempio 1

A che livello di isolamento andrebbe eseguita la seguente transazione?

Example

Per ogni cliente della banca, contare le operazioni effettuate negli ultimi 500 giorni, ed aggiungere il nome del cliente ad un elenco se le operazioni sono più di 300. L'elenco servirà a scopi di marketing.

Visto che la transazione viene usata solo a fini di marketing, è plausibile che qualche sporadico errore nei risultati non vada ad inficiare il quadro generale $\Rightarrow$ READ UNCOMMITTED per maggiore efficienza

Livelli di Isolamento: Esempio 2

A che livello di isolamento andrebbe eseguita la seguente transazione?

Example

Effettuare un trasferimento fondi, sottraendo un ammontare da un conto per poi aggiungerlo ad un altro. Se la sottrazione ha portato ad un valore negativo, l'operazione va annullata.

```
x = get_amount(acc1)
y = get_amount(acc2)
set_amount(x-amt,acc1)
set_amount(y+amt,acc2)
if (x-amt < 0)
  rollback()
```

Livelli di Isolamento: Esempio 2

A che livello di isolamento andrebbe eseguita la seguente transazione?

Example

Effettuare un trasferimento fondi, sottraendo un ammontare da un conto per poi aggiungerlo ad un altro. Se la sottrazione ha portato ad un valore negativo, l'operazione va annullata.

```
x = get_amount(acc1)
y = get_amount(acc2)
set_amount(x-amt,acc1)
set_amount(y+amt,acc2)
if (x-amt < 0)
  rollback()
```

Bisogna acquisire un blocco in fase di lettura già alla prima riga, altrimenti potremmo trasferire gli stessi soldi più volte $\Rightarrow$ REPEATABLE READ

Livelli di Isolamento: Esempio 3

A che livello di isolamento andrebbe eseguita la seguente transazione?

Example

Gestire un prelievo allo sportello come segue: il cassiere legge il saldo corrente del cliente e gli chiede conferma; se il saldo disponibile supera la cifra richiesta dal cliente, viene poi effettuato il pagamento.

```
x = get_amount(acc)
```

```
x = get_amount(acc)
if (x >= amt)
  set_amount(x-amt,acc)
```

Livelli di Isolamento: Esempio 3

A che livello di isolamento andrebbe eseguita la seguente transazione?

Example

Gestire un prelievo allo sportello come segue: il cassiere legge il saldo corrente del cliente e gli chiede conferma; se il saldo disponibile supera la cifra richiesta dal cliente, viene poi effettuato il pagamento.

```
x = get_amount(acc)
```

```
x = get_amount(acc)
if (x >= amt)
  set_amount(x-amt,acc)
```

La durata delle transazioni dovrebbe essere corta per motivi di efficienza, quindi facciamo due transazioni: una a livello READ COMMITTED con solo la lettura preliminare ed una a livello REPEATABLE READ per il pagamento

Transazioni in Postgres

PostgreSQL considera ciascuna istruzione SQL come una transazione: se non viene eseguito BEGIN, ciascuna istruzione ha un BEGIN implicito e (se ha successo) un corrispondente COMMIT.

Il livello di isolamento di default in Postgres è READ COMMITTED, ma i livelli di isolamento garantiscono proprietà più forti rispetto allo standard.

Isolamento	**DR**	**UR**	**LU**	**F**
READ UNCOMMITTED	**no**	sì	sì	sì
READ COMMITTED	no	sì	sì	sì
REPEATABLE READ	no	no	no	**no**
SERIALIZABLE	no	no	no	no

Nota: sebbene la tabella non evidenzi differenze, SERIALIZABLE offre garanzie migliori rispetto a REPEATABLE READ

Transazioni in Postgres

SERIALIZABLE garantisce l'assenza di anomalie di serializzazione, che invece sono possibili a livello REPEATABLE READ.

ID	color
1	black
2	white
3	black
4	white
5	black

T_1

```
UPDATE Table SET color = 'black'
WHERE color = 'white'
```

T_2

```
UPDATE Table SET color = 'white'
WHERE color = 'black'
```

SERIALIZABLE garantisce che la tabella contenga lo stesso colore in tutte le righe alla fine dell'esecuzione di T_1 e T_2, mentre REPEATABLE READ potrebbe portare ad invertire i colori delle varie righe!

Checkpoint

Concetti Chiave

- Transazioni: serializzabilità, atomicità e persistenza
- I diversi livelli di isolamento e la loro implementazione
- Le problematiche principali dei vari livelli di isolamento

Materiale Didattico

Database Systems: Sezione 6.6. Fondamenti di Basi di Dati: Sezione 8.4