

Il linguaggio SQL: transazioni

Sistemi Informativi T

Versione elettronica: [04.8.SQL.transazioni.pdf](#)

Esecuzione seriale di transazioni

- Un DBMS, dovendo supportare l'esecuzione di diverse transazioni che accedono a dati condivisi, potrebbe eseguire tali transazioni in sequenza (“**serial execution**”)
- Ad esempio, due transazioni T1 e T2 potrebbero essere eseguite in questo modo, in cui si evidenzia la successione temporale (“**schedule**”) delle operazioni elementari sul DB:

T1	T2
R(X)	
W(X)	
Commit	
	R(Y)
	W(Y)
	Commit

Esecuzione concorrente di transazioni

- In realtà, un DBMS eseguire più transazioni in concorrenza, alternando l'esecuzione di operazioni di una transazione con quella di operazioni di altre transazioni (“**interleaved execution**”)
- Eseguire più transazioni concorrentemente è necessario per garantire **buone prestazioni**:
 - Si sfrutta il fatto che, mentre una transazione è in attesa del completamento di una operazione di I/O, un'altra può utilizzare la CPU, il che porta ad aumentare il “**throughput**” (n. transazioni elaborate nell'unità di tempo) del sistema
 - Se si ha una transazione “breve” e una “lunga”, **l'esecuzione concorrente porta a ridurre il tempo medio di risposta del sistema**

T1	T2
R(X)	
	R(Y)
	W(Y)
	Commit
W(X)	
Commit	

Riduzione del tempo di risposta

- T1 è “lunga”, T2 è “breve”; per semplicità ogni riga della tabella è un’unità di tempo

time	T1	T2
1	R(X1)	
2	W(X1)	
...		
999	R(X500)	
1000	W(X500)	
1001	Commit	
1002		R(Y)
1003		W(Y)
1004		Commit

T2 richiede a time = 2 di iniziare

time	T1	T2
1	R(X1)	
2		R(Y)
3		W(Y)
4		Commit
5	W(X1)	
...		
1002	R(X500)	
1003	W(X500)	
1004	Commit	

Tempo medio di risposta =
 $(1001 + (1004-1))/2 = 1002$

Tempo medio di risposta =
 $(1004 + 3)/2 = 503.5$

Isolation: gestire la concorrenza

- Il Transaction Manager garantisce che transazioni che eseguono in concorrenza non interferiscano tra loro. Se ciò non avviene, si possono avere 4 tipi base di problemi, esemplificati dai seguenti scenari:
 - Lost Update:** due persone, in due agenzie diverse, comprano entrambe l'ultimo biglietto per il concerto degli U2 a Roma (!?)
 - Dirty Read:** nel programma dei concerti degli U2 figura una tappa a Bologna il 15/07/10, ma quando provate a comprare un biglietto per quella data vi viene detto che in realtà non è ancora stata fissata (!?)
 - 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 € (!?)
 - Phantom Row:** volete comprare i biglietti di tutte e due le tappe degli U2 in Italia, ma quando comprate i biglietti scoprite che le tappe sono diventate 3 (!?)

Lost Update

- Il seguente schedule mostra un caso tipico di lost update, in cui per comodità si evidenziano anche le operazioni che modificano il valore del dato X e si mostra come varia il valore di X nel DB

Questo update viene perso!

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

- Il problema nasce perché T2 legge il valore di X prima che T1 (che lo ha già letto) lo modifichi (“entrambe vedono l’ultimo biglietto”)

Dirty Read

- In questo caso il problema è che una transazione legge un dato “che non c’è”:

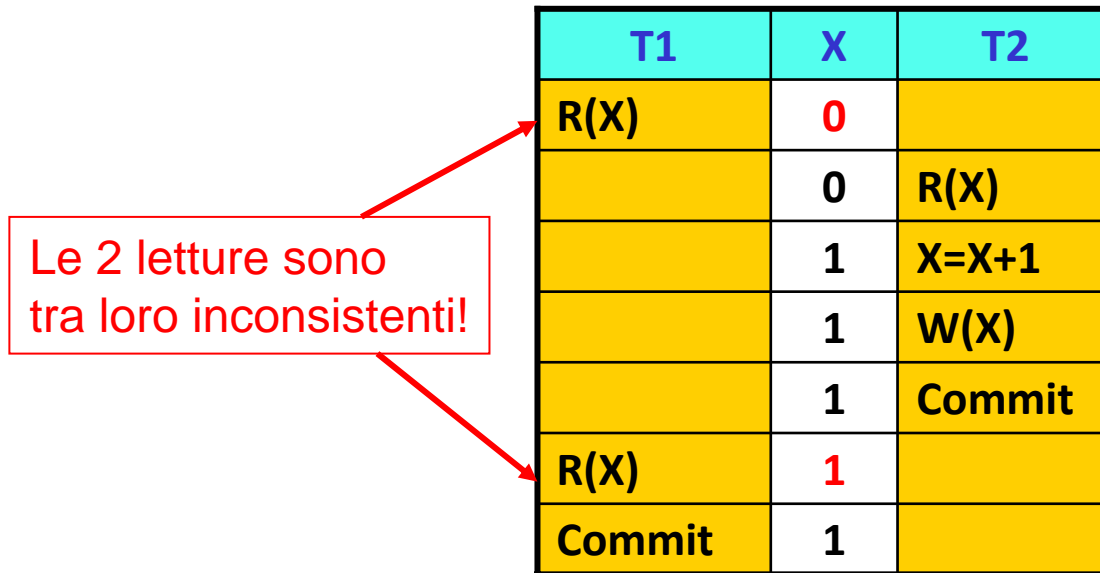
T1	X	T2
R(X)	0	
X=X+1	0	
W(X)	1	
	1	R(X)
Rollback	0	
	0	...
	0	Commit

Questa lettura
è “sporca”!

- Quanto svolto da T2 si basa su un valore di X “intermedio”, e quindi non stabile (“la data definitiva non è il 15/07/10”)
- Le conseguenze sono imprevedibili (dipende cosa fa T2) e si presenterebbero anche se T1 non abortisse

Unrepeatable Read

- Ora il problema è che una transazione legge due volte un dato e trova valori diversi (“il prezzo nel frattempo è aumentato”):



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

Le 2 letture sono tra loro inconsistenti!

- Anche in questo caso si possono avere gravi conseguenze
- Lo stesso problema si presenta per **transazioni di “analisi”**
 - Ad esempio T1 somma l'importo di 2 conti correnti mentre T2 esegue un trasferimento di fondi dall'uno all'altro (T1 potrebbe quindi riportare un totale errato)

Phantom Row

- Questo caso si può presentare quando vengono inserite o cancellate tuple che un'altra transazione dovrebbe logicamente considerare
 - Nell'esempio la tupla t4 è un "phantom", in quanto T1 "non la vede"

T1:

```
UPDATE Prog
SET     Sede = 'Firenze'
WHERE   Sede = 'Bologna'
```

T2:

```
INSERT INTO Prog
VALUES ('P03', 'Bologna')
```

Prog

CodProg	Citta	
P01	Milano	t1
P01	Bologna	t2
P02	Bologna	t3
P03	Bologna	t4

T1 "non vede"
questa tupla!

T1	T2
R(t2)	
R(t3)	
...	
W(t2)	
W(t3)	
	W(t4)
...	
Commit	
	Commit

Livelli di isolamento in SQL

- Scegliere di operare a un livello di isolamento in cui si possono presentare dei problemi ha il vantaggio di aumentare il grado di concorrenza raggiungibile, e quindi di migliorare le prestazioni
- Lo standard SQL definisce 4 livelli di isolamento (si riportano anche i nomi usati da DB2):

Isolation Level	DB2 terminology	Phantom	Unrepeatable Read	Dirty Read	Lost Update
Serializable	Repeatable Read (RR)	NO	NO	NO	NO
Repeatable Read	Read Stability (RS)	YES	NO	NO	NO
Read Committed	Cursor Stability (CS)	YES	YES	NO	NO
Uncommitted Read	Uncommitted Read (UR)	YES	YES	YES	NO

- In DB2 il livello di default è **CS**; per cambiarlo (prima di connettersi al DB) si usa l'istruzione SQL:

CHANGE ISOLATION TO [RR | RS | CS | UR]

Riassumiamo:

- Una **transazione** è un'unità logica di elaborazione che, nel caso generale, si compone di molte operazioni fisiche elementari che agiscono sul DB
- Le proprietà di cui deve godere una transazione si riassumono nell'acronimo **ACID** (**A**tomicity, **C**onsistency, **I**solation, **D**urability)
- **Isolation** richiede che venga correttamente gestita l'esecuzione concorrente delle transazioni
- **Consistency** è garantita dal DBMS verificando che le transazioni rispettino i vincoli definiti a livello di schema del DB