Timestamp

Introduction

Il **timestamp** identifica una transazione e gli è assegnato dallo scheduler quando la transazione ha inizio. Questo può essere:

- il valore di un contatore (più è alto il valore più la transazione è “giovane“)
- l’ora di inizio della transazione

> **Osservazione**
>
> Se il timestamp della transazione T_1 è minore del timestamp della transazione T_2, la transazione T_1 è iniziata prima della transazione T_2.
>
> Quindi se la transizioni non fossero eseguire in modo concorrente ma seriale, T_1 verrebbe eseguita prima di T_2

Serializzabilità

Uno schedule è serializzabile se è equivalente allo schedule seriale in cui le transazioni compaiono ordinate in base al loro timestamp

Quindi uno schedule è serializzabile se per ciascun item acceduto da più di una transazione, l’ordine con cui le transazioni accedono all’item è quelli imposto dal timestamp

Esempi

> **Esempio 1**
>
> Date T_1, T_2 e i loro timestamp $TS(T_1) = 110$ e $TS(T_2) = 100$

T_1
read(X)
X:=X+10
write(X)

T_2
read(X)
X:=X+5
write(X)

Quindi lo schedule è serializzabile se è equivalente allo schedule seriale T_2T_1

T_1	T_2
	read(X)
	X:=X+5
	write(X)
read(X)	
X:=X+10	
write(X)	

Consideriamo il seguente schema

T_1	T_2
	read(X)
read(X)	
	X:=X+5
X:=X +10	
write(X)	
	write(X)

Lo schedule non è serializzabile in quanto T_1 legge X prima che T_2 l'abbia scritto

Esempio 2

Date T_1, T_2 e i loro timestamp $TS(T_1) = 110$ e $TS(T_2) = 100$

T_1
read(Y)
X:=Y+10
write(X)

T_2
read(Y)
X:=Y+5
write(X)

Quindi lo schedule è serializzabile se è equivalente allo schedule seriale $T_2 T_1$

T_1	T_2
	read(Y)
	X:=Y+5
	write(X)
read(Y)	
X:=Y+10	
write(X)	

Consideriamo il seguente schema

T_1	T_2
	read(Y)
read(Y)	
	X:=Y+5
X:=Y+10	
write(X)	
	write(X)

Lo schedule non è serializzabile in quanto il valore di X viene scritto da T_2 al posto che da T_1

read timestamp, write timestamp

A ciascun item X vengono associati due timestamp:

- **read timestamp** di X ($read\_TS(X)$) → il valore più grande fra tutti i timestamp delle transazioni che hanno letto con successo X
- **write timestamp** di X ($read\_WS(X)$) → il valore più grande fra tutti i timestamp delle transazioni che hanno scritto con successo X

Esempi

Tornando agli esempi di prima

Esempio 1

T_1	T_2	read_TS(X)	write_TS(X)
	read(X)	100	
read(X)		110	
	X:=X+5		
X:=X+10			
write(X)			110
	write(X)	read_TS(X)>TS(T_2) roll-back di T_2	

Esempio 2

T_1	T_2	write_TS(X)	read_TS(Y)
	read(Y)		100
read(Y)			110
	X:=Y+5		
X:=Y+10			
write(X)		110	
	write(X)	write_TS(X)>TS(T_2) write(X) non viene eseguita	

Controllo della serializzabilità

Ogni volta che una transazione T cerca di eseguire un $read(X)$ o un $write(X)$, occorre confrontare il timestamp $TS(T)$ di T con il read timestamp e il write timestamp di X per assicurarsi che l'ordine basato sui timestamp non è violato

Algoritmo

T cerca di eseguire una $write(X)$:

1. $read\_TS(X) > TS(T) \rightarrow T$ viene rolled back
2. $write\_TS(X) > TS(T) \rightarrow$ l'operazione di scrittura non viene effettuata
3. se nessuna delle condizioni precedenti è soddisfatta allora
   - $write(X)$ è eseguita
   - $write\_TS(X) := TS(T)$

T cerca di eseguire una $read(X)$:

1. $write\_TS(X) > TS(T) \rightarrow T$ viene rolled back
2. $write\_TS(X) \leq TS(T) \rightarrow$ allora
   - $read(X)$ è eseguita
   - se $read\_TS(X) < TS(T)$ allora $read\_TS(X) := TS(T)$

Esempio

Esempio 1

Il controllo della concorrenza è basato su timestamp, e le transazioni hanno i seguenti timestamp: $TS(T_1) = 110$, $TS(T_2) = 100$, $TS(T_3) = 105$

T1	T2	T3
	READ(X)	
		READ(Y)
	X = X - 20	
READ(X)		
X = X + 10		
		Y = Y - 5
WRITE(X)		
		WRITE(Y)
	WRITE(X)	
READ(Y)		
Y = Y + 20		
WRITE(Y)		

Assumiamo che all'inizio tutti i valori siano azzerati

Passo	Transazione	Operazione	RTS(X)	RTS(Y)	WTS(X)	WTS(Y)	X	Y
1	T2 TS(T2)=100	READ(X)	100	0	0	0	x0	y0
2	T3 TS(T3)=105	READ(Y)	100	105	0	0	x0	y0
3	T2 TS(T2)=100	X=X-20	100	105	0	0	x0	y0
4	T1 TS(T1)=110	READ(X)	110	105	0	0	x0	y0
5	T1 TS(T1)=110	X=X+10	110	105	0	0	x0	y0
6	T3 TS(T3)=105	Y=Y-5	110	105	0	0	x0	y0
7	T1 TS(T1)=110	WRITE(X)	110	105	110	0	x0+10	y0
8	T3 TS(T3)=105	WRITE(Y)	110	105	110	105	x0+10	y0-5
9	T2 TS(T2)=100	WRITE(X) ROLL BACK	110	105	110	105	x0+10	y0-5
10	T1 TS(T1)=110	READ(Y)	110	110	110	105	x0+10	y0-5
11	T1 TS(T1)=110	Y=Y+20	110	110	110	105	x0+10	y0-5
12	T1 TS(T1)=110	WRITE(Y)	110	110	110	110	x0+10	(y0-5)+20

Al passo 9 la transazione T_2 viene abortita. Dovrebbe eseguire la scrittura sull'item X, ma il suo timestamp è minore del timestamp della transazione più giovane (con timestamp più alto) che ha letto l'item X ($RTS(X) = 100 > TS(T_2) = 100$).

Ciò significa che una transazione che ha iniziato le proprie operazioni dopo che T_2 ha già letto il velore dell'item X, mentre secondo l'ordine di esecuzione corretto avrebbe dovuto leggere il valore di X già modificato da T_2. Da qui la necessità di eseguire il rollback della transazione T_2

⁝≡ Esempio 2

Il controllo della concorrenza è basato su timestamp, e le transazioni hanno i seguenti timestamp: $TS(T_1) = 120$, $TS(T_2) = 110$, $TS(T_3) = 100$

T1	T2	T3
		READ(Y)
	READ(X)	
READ(Y)		
Y = Y - 50		
		Y = Y - 20
	X = X + 50	
READ(X)		
X = X + Y		
	WRITE(X)	
WRITE(Y)		
		WRITE(Y)
WRITE(X)		

Assumiamo che all'inizio tutti i valori siano azzerati

Passo	Transazione	Operazione	RTS(X)	RTS(Y)	WTS(X)	WTS(Y)	X	Y
1	T3 TS(T3)=100	READ(Y)	0	100	0	0	x0	y0
2	T2 TS(T2)=110	READ(X)	110	100	0	0	x0	y0
3	T1 TS(T1)=120	READ(Y)	110	120	0	0	x0	y0
4	T1 TS(T1)=120	Y=Y-50	110	120	0	0	x0	y0
5	T3 TS(T3)=100	Y=Y-20	110	120	0	0	x0	y0
6	T2 TS(T2)=110	X=X+50	110	120	0	0	x0	y0
7	T1 TS(T1)=120	READ(X)	120	120	0	0	x0	y0
8	T1 TS(T1)=120	X = X +Y	120	120	0	0	x0	y0
9	T2 TS(T2)=110	WRITE(X) ROLL BACK	120	120	0	0	x0	y0
10	T1 TS(T1)=120	WRITE(Y)	120	120	0	120	x0	y0-50
11	T3 TS(T3)=100	WRITE(Y) ROLL BACK	120	120	0	120	x0	y0-50
12	T1 TS(T1)=120	WRITE(X)	120	120	120	120	x0+ y0-50	y0-50

Al passo 9 la transazione T_2 viene abortita. Dovrebbe eseguire la scrittura sull'item X, ma il suo timestamp è minore del timestamp della transazione più

giovane (con timestamp più alto) che ha letto l'item X ($RTS(X) = 120 > TS(T_2) = 110$).

Ciò significa che una transazione che ha iniziato le proprie operazioni dopo che T_2 ha già letto il velore dell'item X, mentre secondo l'ordine di esecuzione corretto avrebbe dovuto leggere il valore di X già modificato da T_2. Da qui la necessità di eseguire il rollback della transazione T_2

Al passo 11 la transazione T_3 viene abortita per lo stesso motivo. Dovrebbe eseguire la scrittura dell'item Y, ma il suo timestamp è minore del timestamp della transazione più giovane (con timestamp più alto) che ha letto X ($RTS(X) = 120 > TS(T_3) = 100$)

Osservazioni

In entrambi gli esempi precedenti, lo schedule delle transazioni superstiti è equivalente allo schedule seriale delle transazioni eseguite in ordine di arrivo

Quello che provoca il rollback di una transazione Tr che vuole leggere è trovare che una più giovane Tw ha già scritto i dati (nello schedule seriale Tr avrebbe letto la versione precedente alla modifica di Tw che ma non ha eseguito la lettura "in tempo")

Quello che provoca il rollback di una transazione Tw che vuole scrivere è trovare che una più giovane Tr ha già letto i dati (nello schedule seriale avrebbe letto quelli di Tw che però non ha eseguito la scrittuta "in tempo")

Hint

Se Tw vuole srivere, non c'è stata una transazione più giovane Tr che ha già letto i dati, ma una transazione più giovane Tw' li ha già scritti, non è necessario il rollback

Domanda: come mai possiamo saltare l'operazione di scrittura di una transazione T senza violare la proprietà di atomicità (la transazione viene eseguita per intero oppure ogni suo effetto viene annullato)?

La proprietà di atomicità serve a garantire la coerenza dei dati nella base di dati. Dal punto di vista degli effetti della transazione sulla base di dati e di questa coerenza, le transazioni che potrebbero trovare una situazione incoerente sono quelle che

avrebbero dovuto leggere
proprio i dati scritti da T, e invece hanno trovato quelli di una transazione più giovane T' ma nessuna transazione T'' con $TS(T') > TS(T'') > TS(T)$, (arrivata dopo T ma prima di T', che avrebbe dovuto quindi leggere il valore prodotto da T) può aver letto X altrimenti T sarebbe stata rolled back al passo precedente dell'algoritmo di controllo della scrittura e se poi dovesse arrivare, T'' sarebbe rolled back in base al primo passo dell'algoritmo di controllo della lettura (a causa della scrittura di T'') ma se T'' non arriva abbiamo risparmiato un roll back!