Il meccanismo di lock - lock binario

Index

Introduction

Per **lock** si intende il **privilegio di accesso** ad un **singolo item** realizzato mediante una variabile associata all'item (variabile lucchetto) il cui valore descrive lo stato dell'item rispetto alle operazioni che possono essere effettuate su di esso (ogni item può essere locked o unlocked)

Nella sua forma più semplice, un lock:

- viene **richiesto** da una transazione mediante un'operazione di *locking* tramite la quale se il valore della variabile è unlocked la transazione può accedere all'item e alla variabile viene assegnato il valore locked
- viene **rilasciato** da una transazione mediante un'operazione di *unlocking* che assegna alla variabile il valore unlocked

Quindi il locking agisce come **primitiva di sincronizzazione**, cioè se una transazione richiede un lock su un item su cui un'altra transazione mantiene un lock, la transazione non può procedere finché il lock non viene rilasciato dalla prima transazione
Fra l'esecuzioen di un'operazione di locking su un certo item X e l'esecuzione di un'operazione di unlocking su X la transazione **mantiene un lock su** X

Schedule legale

Uno schedule è detto **legale** se una transazione effettua un **locking ogni volta che deve scrivere/leggere** un item e se ciascuna transazione **rilascia ogni lock** che ha ottenuto

---

Lock binario

Un lock **binario** può assumere solo due valori: **locked** e **unlocked**

Le transazioni fanno uso di due operazioni:

- $lock(X) \rightarrow$ per richiedere l'accesso all'item X
- $unlock(X) \rightarrow$ per rilasciare l'item X consentendone l'accesso ad altre transazioni

L'insieme degli item letti e quello degli item scritti da una transazione coincidono

Il lock binario permette di risolvere il problema del lost update (non dirty data né aggregato non corretto)

Esempio

Risolviamo il primo dei problemi visti, cioè il lost update

T_1
read(X)
X:=X-N
write(X)
read(Y)
Y:=Y+N
write(Y)

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

T_1	T_2
read(X)	
X:=X-N	
	read(X)
	X:=X+M
write(X)	
read(Y)	
	write(X)
Y:=Y+N	
write(Y)	

Riscriviamo le transazioni utilizzando le primitive del lock binario

T_1
lock(X)
read(X)
X:=X-N
write(X)
unlock(X)
lock(Y)
read(Y)
Y:=Y+N
write(Y)
unlock(Y)

T_2
lock(X)
read(X)
X:=X+M
write(X)
unlock(X)

Vediamo ora uno schedule legale di T_1 e T_2 che risolve il problema del lost update

T_1	T_2
lock(X)	
read(X)	
X:=X-N	
write(X)	
unlock(X)	
	lock(X)
	read(X)
	X:=X+M
	write(X)
	unlock(X)
lock(Y)	
read(Y)	
Y:=Y+N	
write(Y)	
unlock(Y)	

Modello per le transazioni

T_1
lock(X)
unlock(X)
lock(Y)
unlock(Y)

Una transazione è una **sequenza di operazioni di lock e unlock**:

- ogni $lock(X)$ implica la **lettura** di X
- ogni $unlock(X)$ implica la **scrittura** di X

T_1
lock(X)
unlock(X) $f_1(X)$
lock(Y)
unlock(Y) $f_2(X,Y)$

In corrispondenza di una scrittura viene associato un nuovo valore coinvolto che viene calcolato da una **funzione** che è associata in modo **univoco** ad ogni coppia lock-unlock e che ha come **argomenti tutti gli item letti** (locked) **dalla transazione prima dell'operazione di unlock** (perché magari i loro valori hanno contribuito all'aggiornamento dell'item corrente)

Equivalenza

Due schedule sono **equivalenti** se le formule che danno i valori finali per ciascun item sono le stesse

> ⚠ **Le formule devono essere uguali per tutti gli item**

> ⓘ **Info**
>
> Vedremo che la proprietà di equivalenza degli schedule dipende dal protocollo di locking usato
> Adottiamo un **modello delle transazioni** per poter astrarre delle specifiche operazioni che si basa su quelle rilevanti per valutare le sequenze degli accessi, cioè in questo caso lock e unlock

---

Schedule serializzabile

Uno schedule è **serializzabile** se è equivalente ad uno schedule seriale (basta trovarne uno)

Esempio #1

Consideriamo le due transazioni

T_1
$lock(X)$
$unlock(X)$ $f_1(X)$
$lock(Y)$
$unlock(Y)$ $f_2(X,Y)$

T_2
$lock(Y)$
$unlock(Y)$ $f_3(Y)$
$lock(X)$
$unlock(X)$ $f_4(X,Y)$

e lo schedule

	T_1	T_2	
legge X_0	*lock(X)*		
scrive $f_1(X_0)$	*unlock(X)*		
		lock(Y)	legge Y_0
		unlock(Y)	scrive $f_3(Y_0)$
legge $f_3(Y_0)$	*lock(Y)*		
scrive $f_2(X_0, f_3(Y_0))$	*unlock(Y)*		
		lock(X)	legge $f_1(X_0)$
		unlock(X)	scrive $f_4(f_1(X_0), Y_0)$

Considerando X_0 il valore iniziale di X e Y_0 il valore iniziale di Y allora $f_4(f_1(X_0), Y_0)$ è il valore finale di X

I possibili schedule seriali

Consideriamo lo schedule seriale T_1, T_2

	T_1	T_2	
legge X_0	*lock(X)*		
scrive $f_1(X_0)$	*unlock(X)*		
legge Y_0	*lock(Y)*		
scrive $f_2(X_0, Y_0)$	*unlock(Y)*		
		lock(Y)	legge $f_2(X_0, Y_0)$
		unlock(Y)	scrive $f_3(f_2(X_0, Y_0))$
		lock(X)	legge $f_1(X_0)$
		unlock(X)	scrive $f_4(f_1(X_0), f_2(X_0, Y_0))$

Il valore finale di X è $f_4(f_1(X_0), f_2(X_0, Y_0))$

Consideriamo lo schedule seriale T_2, T_1

	T_1	T_2	
		lock(Y)	legge Y_0
		unlock(Y)	scrive $f_3(Y_0)$
		lock(X)	legge X_0
		unlock(X)	scrive $f_4(X_0, Y_0)$
legge $f_4(X_0, Y_0)$	*lock(X)*		
scrive $f_1(f_4(X_0, Y_0))$	*unlock(X)*		
legge $f_3(Y_0)$	*lock(Y)*		
scrive $f_2(f_4(X_0, Y_0), f_3(Y_0))$	*unlock(Y)*		

Il valore finale di X è $f_1(f_4(X_0, Y_0))$

Pertanto lo schedule **non è serializzabile** in quanto produce per X un valore finale ($f_4(f_1(X_0), Y_0)$) diverso sia da quello prodotto dallo schedule seriale T_1, T_2, sia da quello prodotto dallo schedule seriale T_2, T_1
Vale lo stesso anche per Y

> **Osservazione**
>
> Basta che le formule siano diverse anche per un solo item per concludere che gli schedule non sono equivalenti. Quindi per verificare che uno schedule non è serializzabile, possiamo fermarci appena troviamo un item le cui formule finali sono diverse da quelle di ogni schedule seriale
>
> Per verificare che uno schedule è serializzabile occorre verificare che le formule finali di tutti gli item coincidono con quelle di uno (stesso) schedule seriale

Esempio #2

Consideriamo le due transazioni

T_1
lock(X)
unlock(X) $f_1(X)$
lock(Y)
unlock(Y) $f_2(X,Y)$

T_2
lock(X)
unlock(X) $f_3(X)$
lock(Y)
unlock(Y) $f_4(X,Y)$

e lo schedule

	T_1	T_2	
legge X_0	$lock(X)$		
scrive $f_1(X_0)$	$unlock(X)$		
		$lock(X)$	legge $f_1(X_0)$
		$unlock(X)$	scrive $f_3(f_1(X_0))$
legge Y_0	$lock(Y)$		
scrive $f_2(X_0, Y_0)$	$unlock(Y)$		
		$lock(Y)$	legge $f_2(X_0, Y_0)$
		$unlock(Y)$	scrive $f_4(f_1(X_0), f_2(X_0, Y_0))$

Consideriamo lo schedule seriale T_1, T_2

	T_1	T_2	
legge X_0	$lock(X)$		
scrive $f_1(X_0)$	$unlock(X)$		
legge Y_0	$lock(Y)$		
scrive $f_2(X_0, Y_0)$	$unlock(Y)$		
		$lock(X)$	legge $f_1(X_0)$
		$unlock(X)$	scrive $f_3(f_1(X_0))$
		$lock(Y)$	legge $f_2(X_0, Y_0)$
		$unlock(Y)$	scrive $f_4(f_1(X_0), f_2(X_0, Y_0))$

In questo caso lo schedule è serializzabile in quando produce sia per X che per Y gli stessi valori finali prodotti dallo schedule seriale T_1, T_2

Testare la serializzabilità

Per testare la serializzabilità di uno schedule utilizzo il seguente algoritmo

Dato uno schedule S

Passo 1

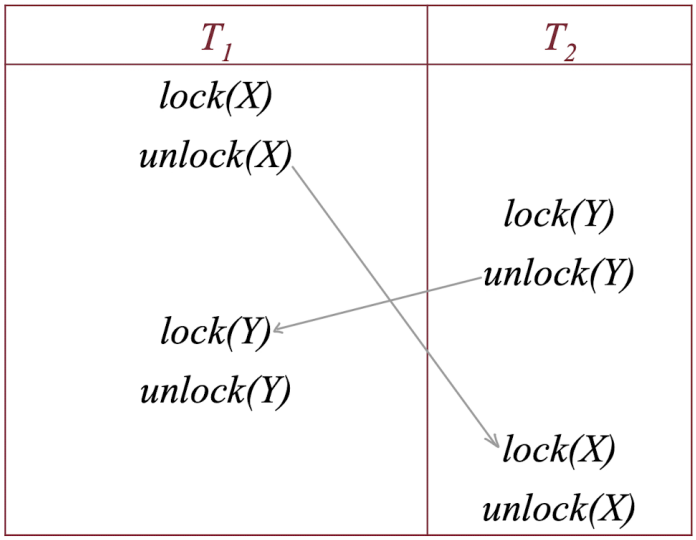
Crea un grafo diretto G (*grafo di serializzazione*) in cui:

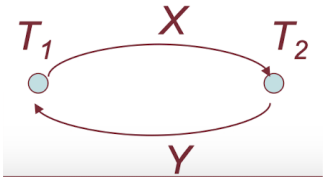
- nodi → transazioni
- archi → $T_i \longrightarrow T_j$ (con etichetta X) se in S si ha che T_i esegue un $unlock(X)$ e T_j esegue il successivo $lock(X)$

> **⚠ Warning**
>
> Non un successivo ma **il** successivo, cioè T_j è la prima transazione che effettua il lock di X dopo che T_i ha effettuato l'unlock, anche se le due operazioni sono di seguito

> **Hint**
>
> Per avere un ciclo bisogna avere archi nella stessa direzione

Esempio





Questo rappresenta il più piccolo gruppo ciclico di due transazioni

T_1	T_2
lock(X)	
unlock(X)	
	lock(X)
	unlock(X)
lock(Y)	
unlock(Y)	
	lock(Y)
	unlock(Y)

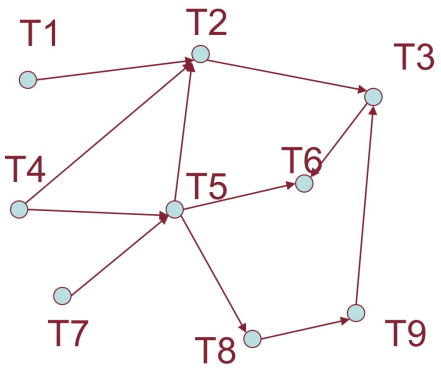
Questo rappresenta il più piccolo gruppo aciclico di due transazioni

Passo 2

Se G ha un ciclo allora S non è serializzabile, altrimenti applicando a G l'**ordinamento topologico** si ottiene uno schedule seriale S' equivalente ad S
Per ottenere l'ordinamento topologico è necessario ricorsivamente un nodo che non ha archi entranti, insieme ai suoi archi uscenti

Esempio

In questo esempio i possibili punti di partenza sono T_1, T_4, T_7 quindi ho almeno 3 possibili schedule seriali



T4 T7 T5 T1 T8 T9 T2 T3 T6

Teorema (correttezza dell'algoritmo del grafo di serializzazione)

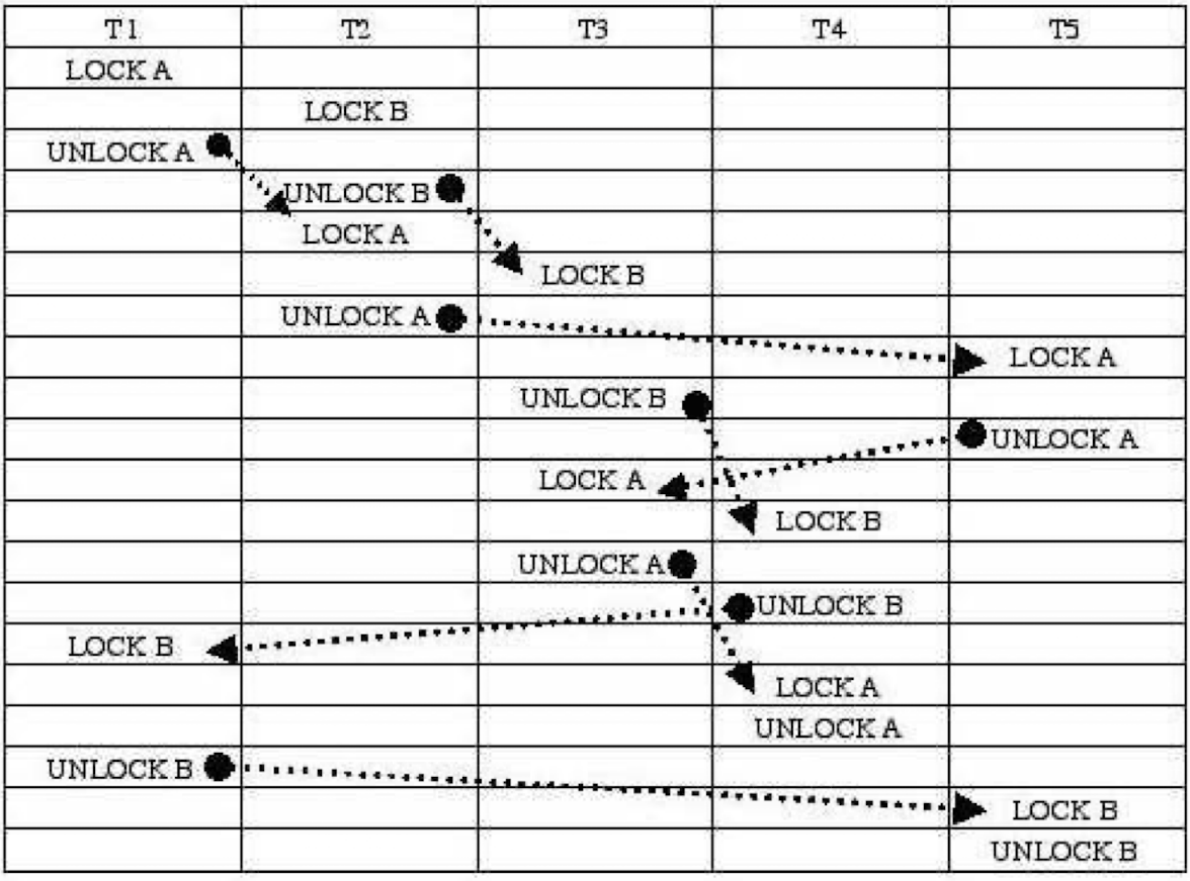
Uno schedule S è serializzabile se e solo se il suo grafo di serializzazione è **aciclico**

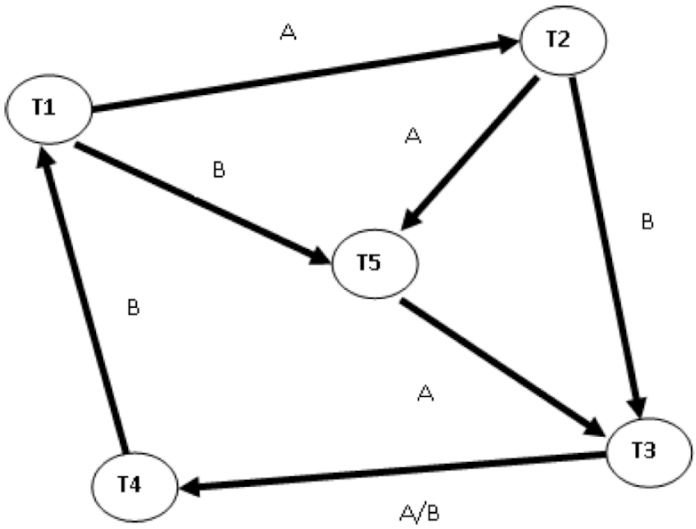
Esempio

Prendiamo questo schedule di 5 transazioni

T1	T2	T3	T4	T5
LOCK A				
	LOCK B			
UNLOCK A				
	UNLOCK B			
	LOCK A			
		LOCK B		
	UNLOCK A			
				LOCK A
		UNLOCK B		
				UNLOCK A
		LOCK A		
			LOCK B	
		UNLOCK A		
			UNLOCK B	
LOCK B				
			LOCK A	
			UNLOCK A	
UNLOCK B				
				LOCK B
				UNLOCK B

Applichiamo l'algoritmo e segniamo sulla tabella le relazioni tra le transazioni che produrranno archi nel grafo





Il grafo presenta il ciclo $T_1 - T_2 - T_3 - T_4$, possiamo quindi concludere che lo schedule non è serializzabile

> ⚠ **Warning**
>
> $T_1 - T_2 - T_5$ e $T_2 - T_5 - T_3$ non sono cicli in quanto i senti delle frecce non descrivono cicli, mentre $T_1 - T_5 - T_3 - T_4 - T_1$ lo è

Protocollo di locking a due fasi

Una transazione obbedisce al protocollo di **locking a due fasi** se prima effettua tutte le operazioni di lock (*fase di locking*) e poi tutte le operazioni di unlock (*fase di unlocking*)

> ⚠ **Warning**
>
> Da non confondere con il lock a due valori. Il fatto di essere a due fasi è una caratteristica in più ma ci sono protocolli a due fasi e tre valori

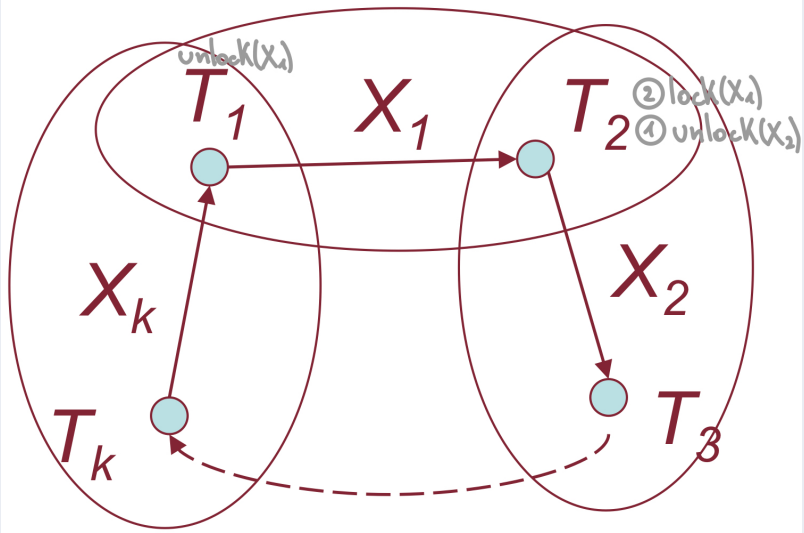
Risolve il problema dell'aggregato non corretto (non dirty data)

Teorema sul lock a due fasi

Sia T un insieme di transazioni. Se ogni transazione in T è a due fasi allora ogni schedule di T è serializzabile

Dimostrazioni

Supponiamo per assurdo che ogni transazione in S è a due fasi ma nel grafo di serializzazione c'è un ciclo



Avendo uno schedule nel corso della sua esecuzione ha eseguito prima una $unlock(X_2)$ e poi una $lock(X_1)$ e dunque lo schema non è a due fasi
CONTRADDIZIONE

■

Bisogna però ricordare che questo teorema non implica il contrario. Possono quindi esistere schedule non a due fasi ma serializzabili

T_1	T_2
lock(X)	
unlock(X)	
	lock(X)
	unlock(X)
lock(Y)	
unlock(Y)	
	lock(Y)
	unlock(Y)

Hint

Tutti i protocolli di lock a due fasi (a prescindere dal numero di valori di lock) risolvono il problema dell’aggregato non corretto