Esecuzione concorrente di transazioni

ALBERTO BELUSSI

PARTE II
ANNO ACCADEMICO 2010-2011

Tecniche applicate nei DBMS

Le tecniche per il controllo della concorrenza che non richiedono di conoscere l'esito delle transazioni sono:

- Locking a due fasi stretto (2PL stretto)
- Timestamp con scritture bufferizzate (TS buffer)

Timestamp

Il controllo della concorrenza basato su timestamp adotta il seguente meccanismo:

- Ad ogni transazione viene associato un numero (timestamp) che rappresenta l'istante di inizio della transazione
- Si accetta uno schedule solo se esso riflette l'ordinamento delle transazioni basate sui timestamp assegnati. Ciò si traduce nelle seguenti regole:
  - Una transazione non può leggere o scrivere una risorsa scritta da una transazione con TS superiore
  - Una transazione non può scrivere una risorsa letta da una transazione con TS superiore

Realizzazione del controllo basato sui timestamp nei DBMS

Le transazioni per accedere alle risorse invocano due primitive:

- Read(x, ts): dove x è la risorsa da leggere e ts è il timestamp della transazione
- Write(x, ts): dove x è la risorsa da scrivere e ts è il timestamp della transazione

Inoltre, ogni risorsa x ha associati due indicatori:

- RTM(x) – *Read Timestamp Maximum*: indica il TS dell'ultima transazione che ha eseguito una lettura di x.
- WTM(x) – *Write Timestamp Maximum*: indica il TS dell'ultima transazione che ha eseguito una scrittura di x.

Realizzazione del controllo basato sui timestamp nei DBMS

Un modulo, detto SCHEDULER, regola gli accessi alle risorse rispondendo alle primitive Read e Write invocate dalle transazioni secondo la seguente politica:

- Read(x, ts): viene accettata se ts ≥ WTM(x) altrimenti la transazione viene abortita;
  se accettata allora RTM(x) = max(ts, RTM(x))
- Write(x, ts): viene accettata se ts ≥ WTM(x) e ts ≥ RTM(x) altrimenti la transazione viene abortita;
  se accettata allora WTM(x) = ts

Realizzazione del controllo basato sui timestamp nei DBMS

Esempio su risorsa x

		risp.	oper.	WTM(x)	RTM(x)
stessa transazione	read(x,4)	no	t4 abort	5	7
	read(x,6)	ok		5	7
	read(x,9)	ok		5	9
stessa transazione	write(x,8)	no	t8 abort	5	9
	write(x,10)	ok		10	9
	write(x,13)	ok		13	9
stessa transazione	read(x,11)	no	t11 abort	13	9
	read(x,14)	ok		13	14

Realizzazione del controllo basato sui timestamp nei DBMS

Per rendere il controllo basato sui timestamp indipendente dall'esito delle transazioni è necessario

- Bufferizzare in memoria centrale le scritture
- Trasferire in memoria secondaria i buffer relativi ad una transazione al momento del commit di tale transazione
- Porre in attesa dell'esito della transazione le transazioni che richiedono l'accesso in lettura a risorse presenti nei buffer.

Locking a due fasi

- E' il metodo applicato nei sistemi commerciali per la gestione dell'esecuzione concorrente di transazioni.
- Non richiede di conoscere in anticipo l'esito delle transazioni

Tre sono gli aspetti che caratterizzano il locking a due fasi:

- Il **meccanismo** di base per la gestione dei **lock**
- La **politica** di concessione dei lock sulle risorse
- La regola che garantisce la **serializzabilità**

Meccanismo di base

Si basa sull'introduzione di alcune primitive di lock che consentono alle transazioni di bloccare (lock) le risorse sulle quali vogliamo agire con operazioni di lettura e scrittura.

Primitive di lock

- **r_lock$_K$(x)**: richiesta di un lock *condiviso* da parte della transazione $\mathbf{t_K}$ sulla risorsa **x** per eseguire una lettura.
- **w_lock$_K$(x)**: richiesta di un lock *esclusivo* da parte della transazione $\mathbf{t_K}$ sulla risorsa **x** per eseguire una scrittura.
- **unlock$_K$(x)**: richiesta da parte della transazione $\mathbf{t_K}$ di liberare la risorsa **x** da un precedente lock.

Meccanismo di base

Regole per l'uso delle primitive da parte delle transazioni.

- **R1**: ogni lettura deve essere preceduta da un r_lock e seguita da un unlock. Sono ammessi più r_lock contemporanei sulla stessa risorsa (lock condiviso).
- **R2:** ogni scrittura deve essere preceduta da un w_lock e seguita da un unlock. Non sono ammessi più w_lock (oppure w_lock e r_lock) contemporanei sulla stessa risorsa (lock esclusivo).

Se una transazione segue le regole R1 e R2 si dice BEN FORMATA rispetto al locking.

Politica di concessione dei lock

Il gestore dei lock mantiene per ogni risorsa le seguenti informazioni:

risorsa x

- **stato**: $s(x) \in \{\text{libero}, \text{r\_lock}, \text{w\_lock}\}$
- **transazioni in r_lock**: $c(x) = \{t_1, \ldots, t_n\}$
  dove $t_1, \ldots, t_n$ hanno un r_lock su x

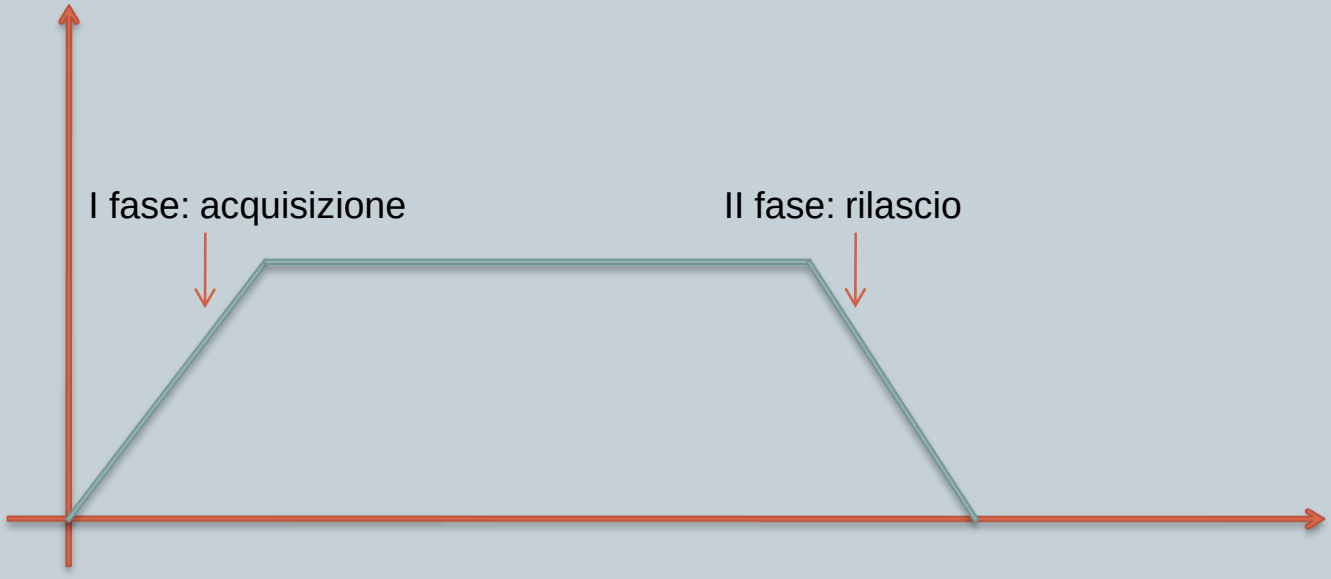
Comportamente del gestore dei lock

Stato **Richiesta**	**LIBERO**		**R_LOCK**		**W_LOCK**	
$r\_lock_K(x)$	Esito OK	Operazioni s(x) = r_lock c(x) ={k}	Esito OK	Operazioni c(x) = c(x)∪{k}	Esito Attesa	Operaz. -
$w\_lock_K(x)$	Esito OK	Operazioni s(x) = w_lock	if |c(x)|=1 and k∈c(x) then		Esito Attesa	Operaz. -
			Esito OK	Operazioni s(x) = w_lock		
			else Attesa	-		
$unlock_K(x)$	Esito Errore	Operazioni -	Esito OK	Operazioni c(x) = c(x) – {k} if c(x)=∅ then s(x)=Libero verifica coda	Esito OK	c(x)=∅ s(x)=Libero verifica coda

Serializzabilità

La regola che garantisce la serializzabilità (da cui prende il nome il metodo 2PL)

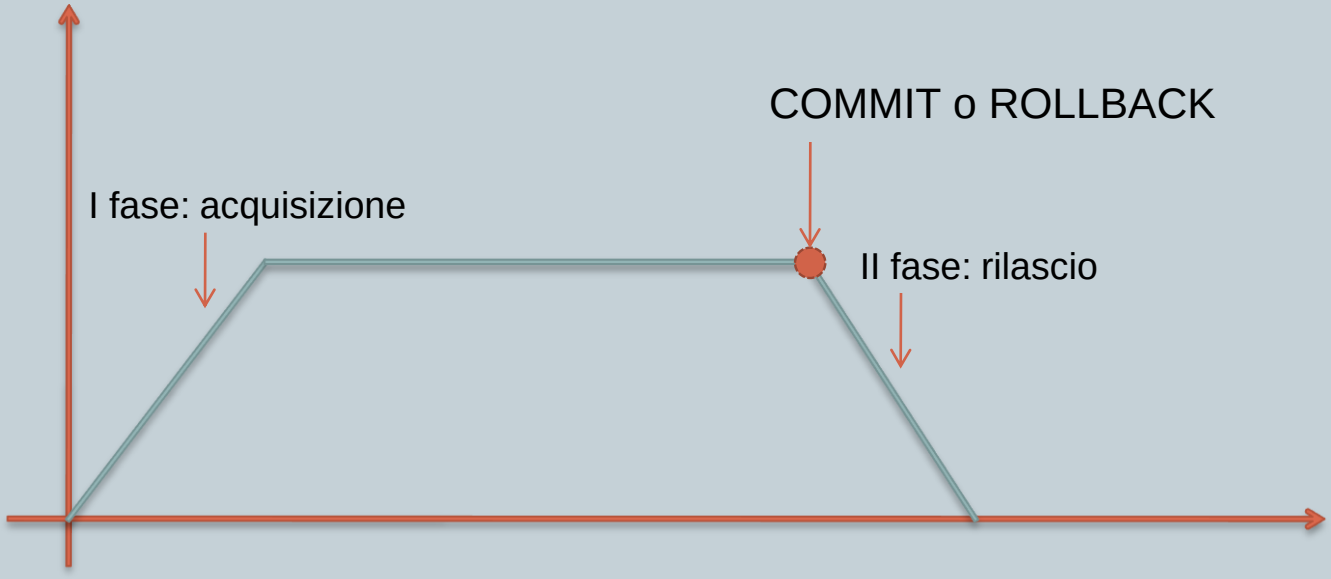
Una transazione dopo aver rilasciato un lock non può acquisirne altri.



Per rimuovere COMMIT-proiezione

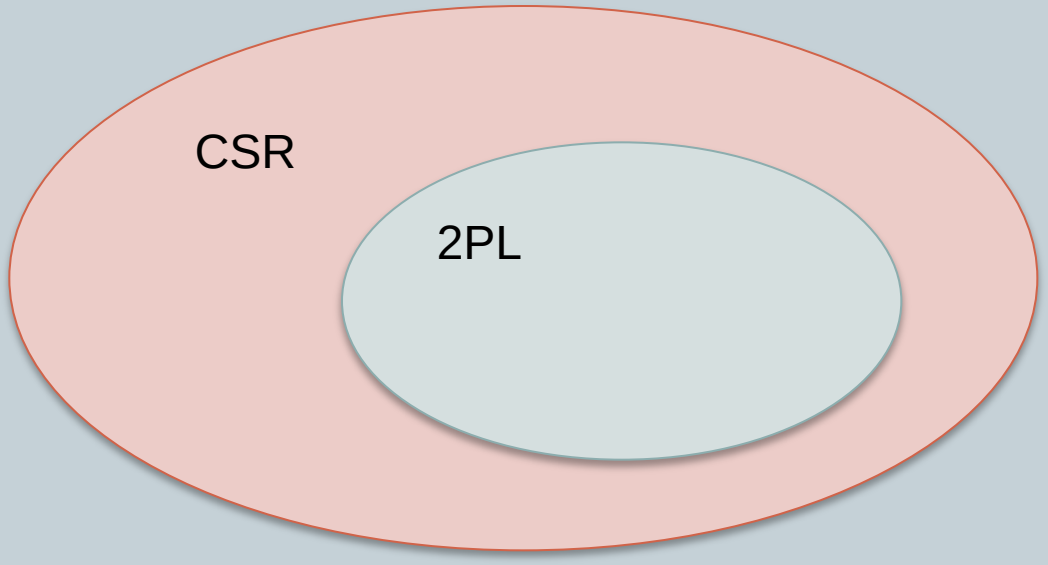
Condizione aggiuntiva (2PL stretto)

Una transazione può rilasciare i lock solo quando ha eseguito correttamente un COMMIT o un ROLLBACK.



2PL a confronto con le tecniche precedenti

Relazione tra 2PL e CSR



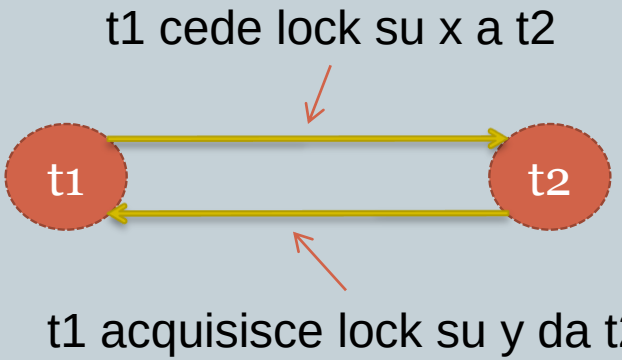
2PL a confronto con le tecniche precedenti

Teorema: $2PL \subset CSR$

1. Se $s \in 2PL \Rightarrow s \in CSR$

   Per assurdo: $s \in 2PL$ e $s \notin CSR$

   Se $s \notin CSR$ allora esiste una coppia (catena) di conflitti ciclici



Esempio: w1(x) r2(x) w2(x) w2(y) r1(y)

t1 rilascia lock su x t1 acquisisce lock su y

Quindi s non è 2PL

2PL a confronto con le tecniche precedenti

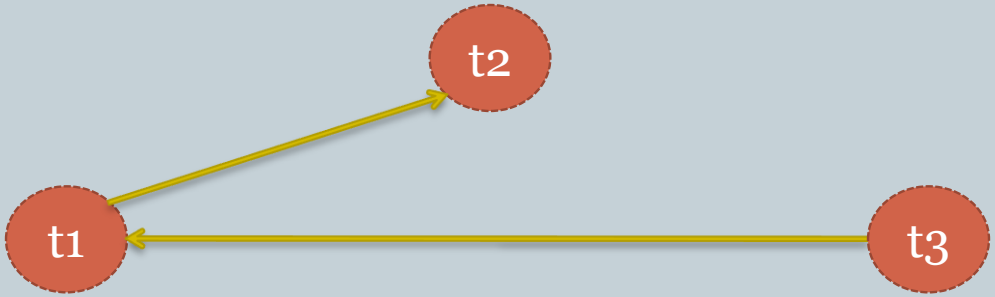
Teorema: 2PL $\subset$ CSR

2. $\exists s \in CSR$ e $s \notin 2PL$

s: r1(x) w1(x) r2(x) w2(x) r3(y) w1(y)

t1 rilascia lock su x t1 acquisisce lock su y

Conflitti(s)={(r1(x),w2(x)),(w1(x),r2(x)),(w1(x),w2(x)),(r3(y),w1(y))}



Esempio perdita di update con 2PL

Esempio su risorsa x=2

t1	M.C.	Database			M.C.	T2
		c(x)	s(x)	Val		
bot			L			
$r\_lock_1(x)$		{1}	RL	2		
$r_1(x)$	2	{1}	RL	2		
x = x +1	3	{1}	RL	2		
	3	{1}	RL	2		bot
	3	{1,2}	RL	2		$r\_lock_2(x)$
	3	{1,2}	RL	2	2	$r_2(x)$
	3	{1,2}	RL	2	3	x=x+1
	3	{1,2}	RL	2	3	$w\_lock_2(x)$
$w\_lock_1(x)$	3	{1,2}	RL	2	3	→ ATTESA
→ ATTESA						

BLOCCO CRITICO

Blocco critico

E' una situazione di blocco che si verifica nell'esecuzione di transazioni concorrenti quando:

- Due transazioni hanno bloccato delle risorse: t1 ha bloccato r1 e t2 ha bloccato r2
- Inoltre t1 è in attesa su r2 e
- T2 è in attesa su r1

Quanto spesso si verifica?

Se il numero medio di tuple per tabella è n e la granularità del lockè la tupla, la probabilità che si verifichi un lock tra due transazioni è pari a:

$$P(\text{deadlock di lunghezza } 2) = 1/n$$

Tecniche per risolvere il blocco critico

- Timeout: una transazione in attesa su una risorsa trascorso il timeout viene abortita
- Prevenzione
  - Una transazione blocca tutte le risorse a cui deve accedere in lettura o scrittura in una sola volta (o blocca tutto o non blocca nulla)
  - Ogni transazione ti acquisisce un timestamp TSi all'inizio della sua esecuzione. Ti può attendere una risorsa bloccata da tj solo se vale una certa condizione sui TS (TSi< TSj) altrimenti viene abortita e fatta ripartire con lo stesso TS.
- Rilevamento: si esegue un'analisi periodica della tabella di LOCK per rilevare la presenza di un blocco critico (corrisponde ad un ciclo nel grafo delle relazioni di attesa). Questo sistema è quello più frequentemente adottato dai sistemi reali.

Gestione della concorrenza in SQL

Poiché risulta molto oneroso per il sistema gestire l'esecuzione concorrente di transazioni con una conseguente diminuzione delle prestazioni del sistema, SQL prevede la possibilità di rinunciare in tutto o in parte al controllo di concorrenza per aumentare le prestazioni del sistema.

Ciò significa che è possibile a livello di singola transazione decidere di tollerare alcune anomalie di esecuzione concorrente.

Gestione della concorrenza in SQL

Livello di isolamento	Perdita di update	Lettura sporca	Lettura inconsistente	Update fantasma	Inserimento fantasma
`serializable`	X	X	X	X	X
`repeatable read`	X	X	X	X	
`read committed`	X	X			
`read uncommitted`	X				

Gestione della concorrenza in SQL

Note:

- Tutti i livelli richiedono il 2PL stretto per le scritture.
- `serializable`: lo richiede anche per le letture.
- `repeatable read`: applica il 2PL stretto per tutti i lock in lettura applicati a livello di tupla e non di tabella.
- `read committed`: richiede lock condivisi per le letture ma non il 2PL stretto.
- `read uncommetted`: non applica lock in lettura.