

# Esecuzione concorrente di transazioni



**ALBERTO BELUSSI**

**PARTE II**

**ANNO ACCADEMICO 2018-2019**

# Tecniche applicate nei DBMS

2

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

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

# Locking a due fasi

3

- E' il metodo applicato nei sistemi reali 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

4

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

## Primitive di lock

- **$r\_lock_K(x)$** : richiesta di un lock *condiviso* da parte della transazione  $t_K$  sulla risorsa  $x$  per eseguire una lettura.
- **$w\_lock_K(x)$** : richiesta di un lock *esclusivo* da parte della transazione  $t_K$  sulla risorsa  $x$  per eseguire una scrittura.
- **$unlock_K(x)$** : richiesta da parte della transazione  $t_K$  di liberare la risorsa  $x$  da un precedente lock.

# Meccanismo di base

5

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

6

Il Gestore dei LOCK (o Gestore della concorrenza) 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:  $c(x) = \{t_1, \dots, t_n\}$   
dove  $t_1, \dots, t_n$  hanno un r\_lock su  $x$

# Comportamento del Gestore dei LOCK

7

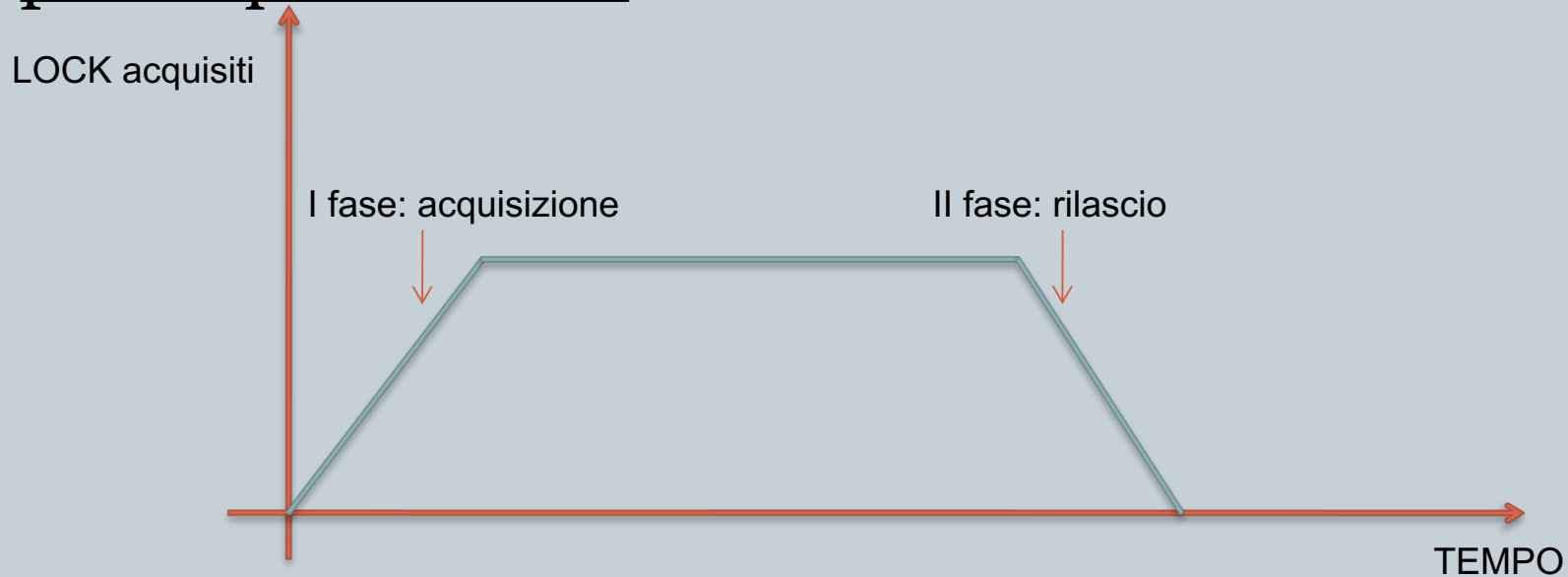
Stato di x	LIBERO		R_LOCK		W_LOCK	
Richiesta						
$r\_lock_k(x)$	Esito OK	Operazioni $s(x) = r\_lock$ $c(x) = \{k\}$	Esito OK	Operazioni $c(x) = c(x) \cup \{k\}$	Esito Attesa	Operaz. -
$w\_lock_k(x)$	Esito OK	Operazioni $s(x) = w\_lock$	if $ c(x) =1$ and $k \in c(x)$ then		Esito Attesa	Operaz. -
			Esito OK	Operazioni $s(x) = w\_lock$		
			else Attesa	-		
$unlock_k(x)$	Esito Errore	Operazioni -	Esito OK	Operazioni $c(x) = c(x) - \{k\}$ if $c(x) = \emptyset$ then $s(x) = \text{Libero}$ verifica coda	Esito OK	$c(x) = \emptyset$ $s(x) = \text{Libero}$ verifica coda

# Serializzabilità

8

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.



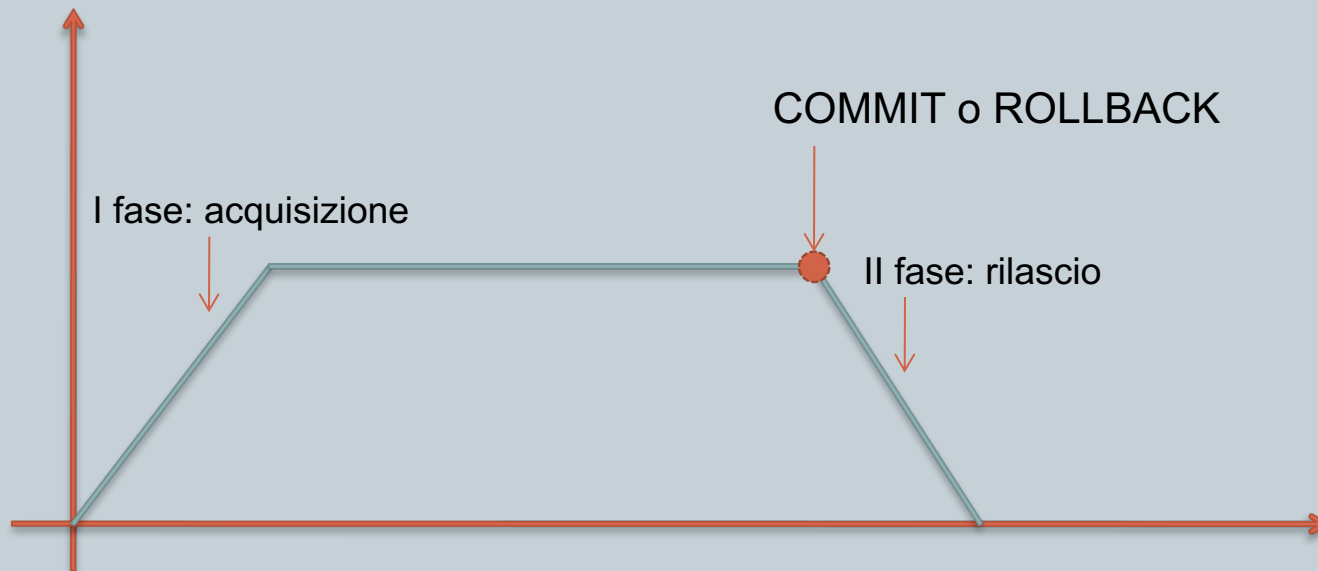


# Per rimuovere COMMIT-proiezione

9

## Condizione aggiuntiva (2PL stretto)

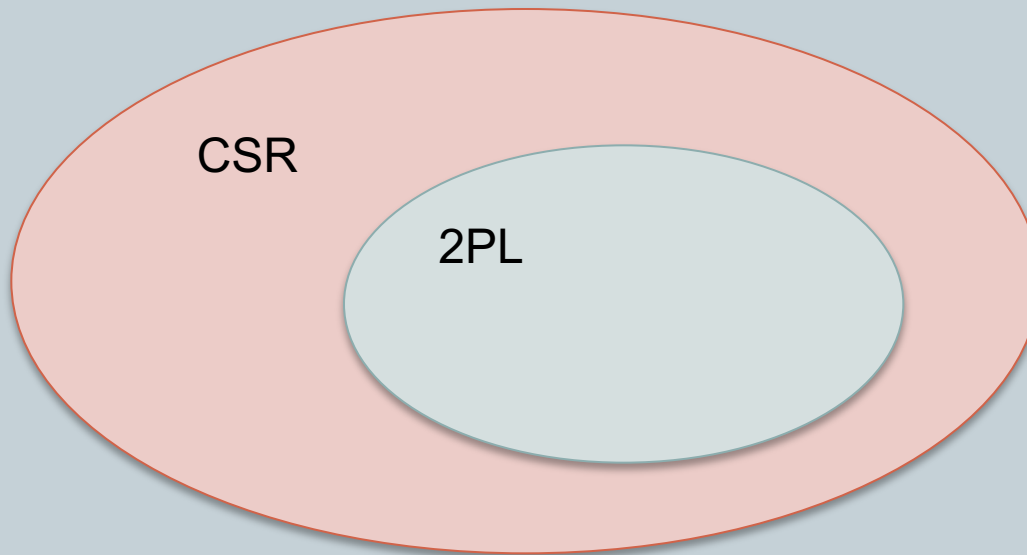
Una transazione può rilasciare i LOCK solo quando ha eseguito correttamente un COMMIT o un ROLLBACK.



# 2PL a confronto con le tecniche precedenti

10

## Relazione tra 2PL e CSR



# 2PL a confronto con le tecniche precedenti

11

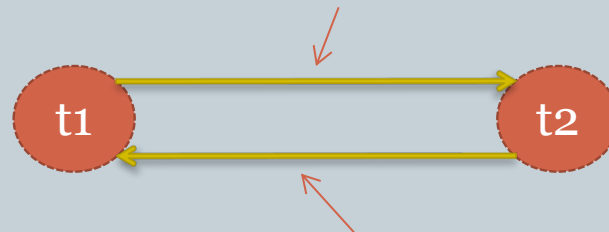
**Teorema:  $2PL \subset CSR$**

**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

t1 cede lock su x a t2



t1 acquisisce lock su y da t2

**Esempio:  $w_1(x) \ r_2(x) \ w_2(x) \ w_2(y) \ r_1(y)$**

t1 rilascia lock su x      t1 acquisisce lock su y

Quindi s non è  
2PL

# 2PL a confronto con le tecniche precedenti

12

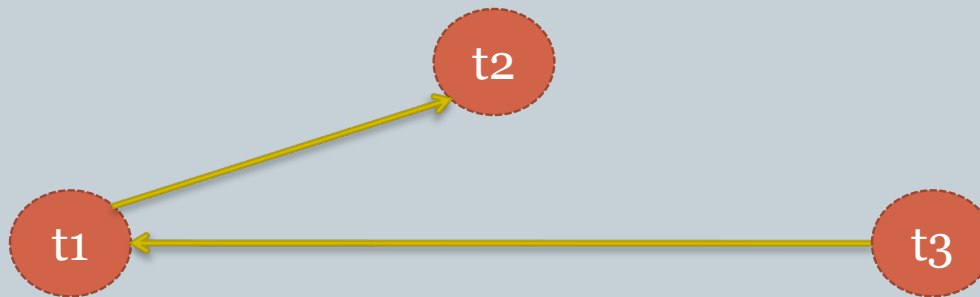
Teorema:  $2PL \subset CSR$

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

13

## Esempio su risorsa $x=2$

<b>t1</b>	<b>M.C.</b>	<b>Database</b>	<b>M.C.</b>	<b>T2</b>
		c(x) s(x) Val		
bot		$\emptyset$ L 2		
r_lock <sub>1</sub> (x)		{1} RL 2		
r <sub>1</sub> (x)	2	{1} RL 2		
x = x + 1	3	{1} RL 2		
	3	{1} RL 2		bot
	3	{1,2} RL 2		r_lock <sub>2</sub> (x)
	3	{1,2} RL 2	2	r <sub>2</sub> (x)
	3	{1,2} RL 2	3	x=x+1
	3	{1,2} RL 2	3	w_lock <sub>2</sub> (x)
w_lock <sub>1</sub> (x)	3	{1,2} RL 2	3	→ ATTESA
→ ATTESA				

**BLOCCO CRITICO**

# Blocco critico

14

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 sulla risorsa r2 e
- t2 è in attesa sulla risorsa r1

Quanto spesso si verifica?

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

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

# Tecniche per risolvere il blocco critico

15

- **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  $t_i$  acquisisce un timestamp  $TS_i$  all'inizio della sua esecuzione. La transazione  $t_i$  può attendere una risorsa bloccata da  $t_j$  solo se vale una certa condizione sui TS ( $TS_i < TS_j$ ) 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). Questa tecnica è quella più frequentemente adottata dai sistemi reali.

# Blocco di una transazione (starvation)

16

- La scelta di gestire i lock in lettura come lock condivisi produce un ulteriore problema che tuttavia nel contesto delle DBMS risulta poco probabile.
- Infatti se una risorsa  $x$  viene costantemente bloccata da una sequenza di transazioni che acquisiscono  $r\_lock$  su  $x$ , un'eventuale transazione in attesa di scrivere su  $x$  viene bloccata per un lungo periodo fino alla fine della sequenza di letture.





# Blocco di una transazione (starvation)

17

- Anche se poco probabile tale evenienza può essere scongiurata con tecniche simili a quanto visto per il blocco critico
- In particolare è possibile analizzare la tabella delle relazioni di attesa e verificare da quanto tempo le transazioni stanno attendendo la risorsa e di conseguenza sospendere temporaneamente la concessione di lock in lettura condivisi per consentire la scrittura da parte della transazione in attesa.

# Gestione della concorrenza in SQL

18

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

19

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

20

## Note:

- Tutti i livelli richiedono il 2PL stretto per le scritture.
- `serializable`: lo richiede anche per le letture e applica il lock di predicato per evitare l'inserimento fantasma.
- `repeatable read`: applica il 2PL stretto per tutti i lock in lettura applicati a livello di tupla e non di tabella. Consente inserimenti e quindi non evita l'anomalia di inserimento fantasma.
- `read committed`: richiede lock condivisi per le letture ma non il 2PL stretto.
- `read uncommitted`: non applica lock in lettura.