Basi di Dati
Modulo Tecnologie

Esecuzione concorrente di transazioni (III parte)

Dr. Sara Migliorini

Schedule serializzabili

Tecniche viste per la verifica di serializzabilità:

- View-serializzabilità
  - Calcolo degli insiemi LEGGI_DA e SCRITTURE FINALI
  - Difetti:
    - Ipotesi di commit-proiezione
    - Test VSR ha complessità elevata: richiede di calcolare tutti i possibili schedule seriali
- Conflict-serializzabilità
  - Calcolo del grafo dei conflitti
  - Difetti:
    - Ipotesi di commit-proiezione

Tecniche applicate nei DBMS

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

- 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

- 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

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 (o Gestore della concorrenza) mantiene per ogni risorsa x le seguenti informazioni:

- stato: $s(x) \in$ {libero, r_lock, w_lock}
- transazioni in r_lock: $c(x)$ = {t1,…, tn}
  - dove t1,…, tn hanno un r_lock su x

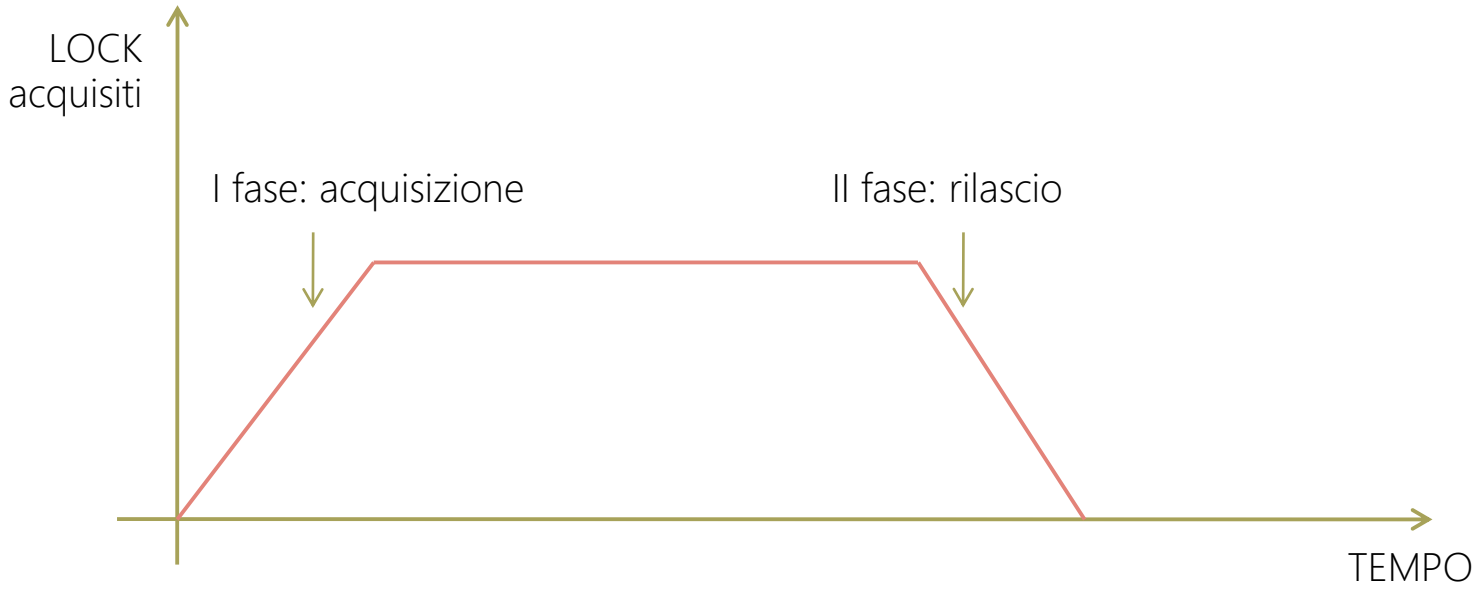
Comportamento del Gestore dei LOCK

<table>
<tr><th>Stato x
Richiesta</th><th colspan="2">LIBERO</th><th colspan="2">R_LOCK</th><th colspan="2">W_LOCK</th></tr>
<tr><td>r_lock$_K(x)$</td><td>Esito
OK</td><td>Operazioni
s(x) = r_lock
c(x) ={k}</td><td>Esito
OK</td><td>Operazioni
c(x) = c(x)∪{k}</td><td>Esito
Attesa</td><td>Operazioni
-</td></tr>
<tr><td rowspan="3">w_lock$_K(x)$</td><td rowspan="3">Esito
OK</td><td rowspan="3">Operazioni
s(x) = w_lock</td><td colspan="2">if |c(x)|=1 and k∈c(x) then</td><td rowspan="3">Esito
Attesa</td><td rowspan="3">Operazioni
-</td></tr>
<tr><td>Esito
OK</td><td>Operazioni
s(x) = w_lock</td></tr>
<tr><td>else
Attesa</td><td>-</td></tr>
<tr><td>unlock$_K(x)$</td><td>Esito
Errore</td><td>Operazioni
-</td><td>Esito
OK</td><td>Operazioni
c(x) = c(x) – {k}
if c(x)=∅ then
s(x)=Libero
verifica coda</td><td>Esito
OK</td><td>c(x)=∅
s(x)=Libero
verifica coda</td></tr>
</table>

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.



Esempio perdita di aggiornamento con 2PL

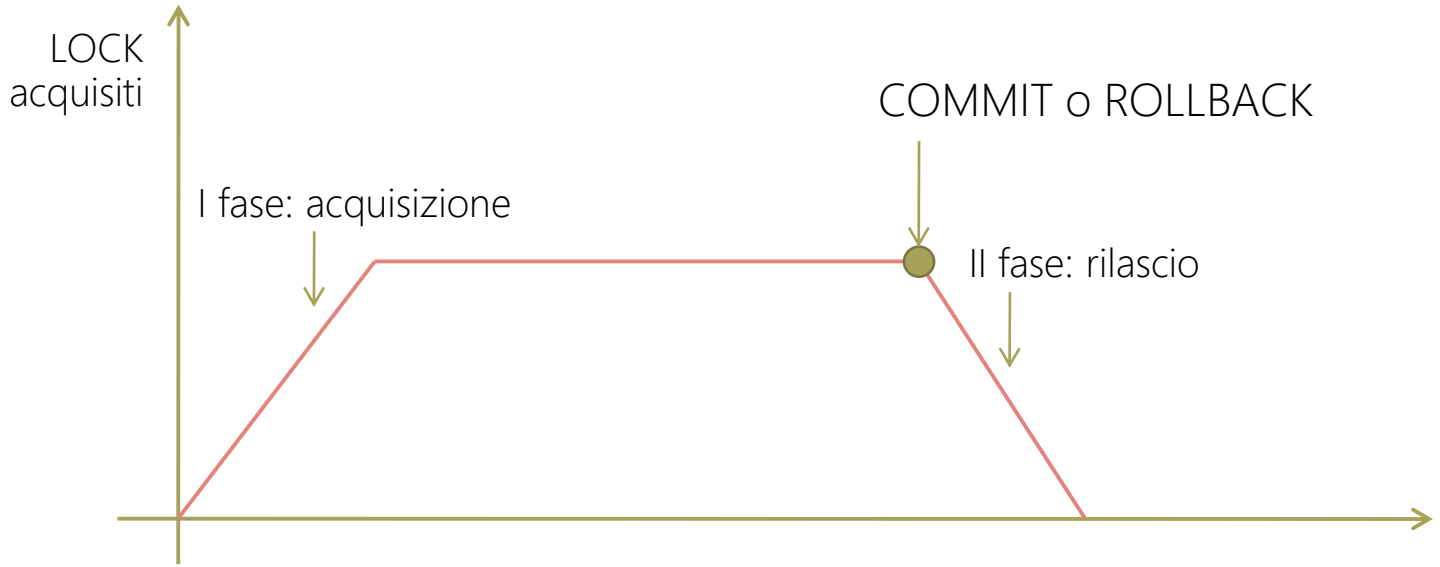
t1	M.C.	Database			M.C.	t2
		c(x)	s(x)	Val		
bot		∅	L	2		
$r\_lock_1(x)$		{1}	RL	2		
$r_1(x)$	2	{1}	RL	2		
x = x +1	3	{1}	RL	2		
	3	{1}	RL	2		bot
	3	{1,2}	RL	2		$r\_lock_2(x)$
	3	{1,2}	RL	2	2	$r_2(x)$
	3	{1,2}	RL	2	3	x=x+1
	3	{1,2}	RL	2	3	$w\_lock_2(x) \rightarrow$ ATTESA
$w\_lock_1(x) \rightarrow$ ATTESA	3	{1,2}	RL	2	3	

BLOCCO CRITICO

Per rimuovere COMMIT-proiezione

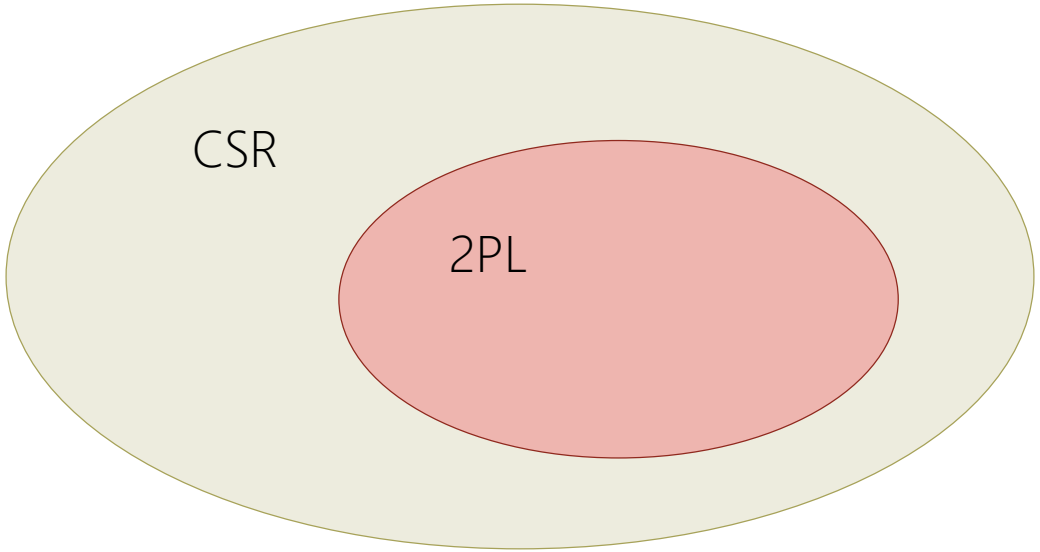
Condizione aggiuntiva (2PL stretto)

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



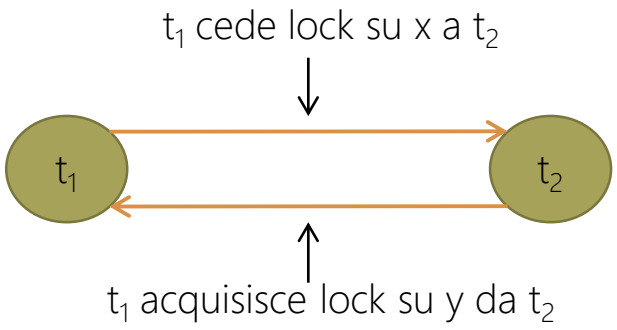
2PL a confronto con le tecniche precedenti

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)

t_1 rilascia lock su x t_1 acquisisce lock su y

Quindi s non è 2PL

2PL a confronto con le tecniche precedenti

Teorema: 2PL $\subset$ CSR

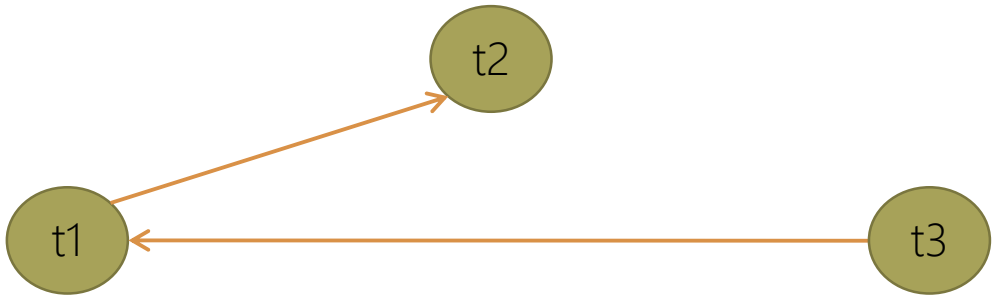
2. $\exists s \in CSR$ e $s \notin 2PL$

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

t_1 rilascia lock su x

t_1 acquisisce lock su y

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



Quindi s non è 2PL
ma è CSR

Blocco critico

- È 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(deadlock di lunghezza 2) = $1/n^2$

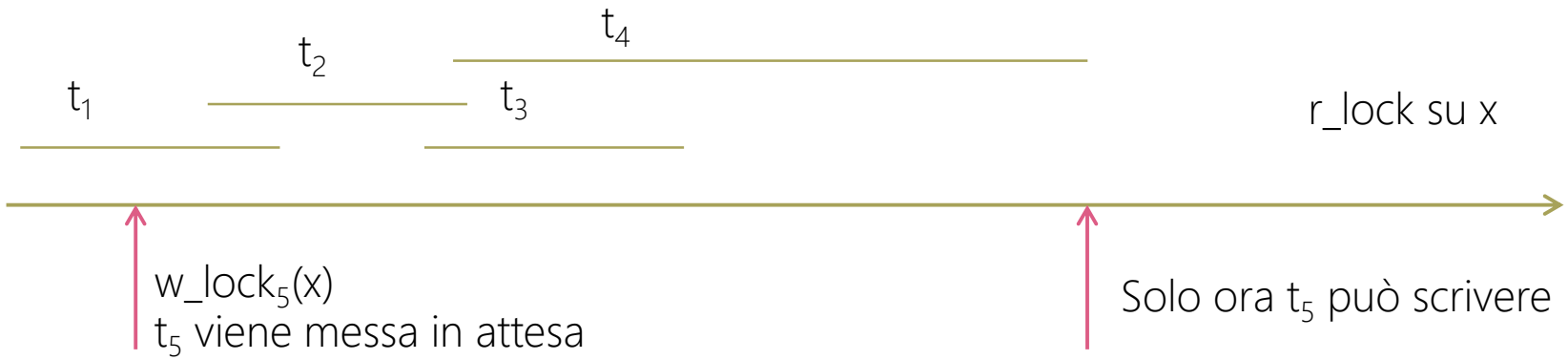
Tecniche per risolvere il blocco critico

- **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 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)

- La scelta di gestire i lock in lettura come lock condivisi produce un ulteriore problema chiamato starvation che tuttavia nel contesto delle DBMS risulta poco probabile.

- 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)

- 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

- 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	✓	✓	✓	✓	✓
repeatable read	✓	✓	✓	✓	
read committed	✓	✓			
read uncommitted	✓				

Gestione della concorrenza in SQL

Nota:

- 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 uncommetted: non applica lock in lettura.