ESERCIZI DI BASI DI DATI
Tecnologia delle basi di dati
(draft)

prof. Angelo Montanari
Michela Zamparini
Alberto Guglielmetti

17 maggio 2001

Indice

Riferimenti bibliografici:

- R. Elmasri, S. Navathe, *Fundamentals of Database Systems*, 3^{rd} ed., Addison-Wesley, 2000;
- P. Atzeni, S. Ceri, S. Paraboschi, R. Torlone, *Basi di Dati*, 2^{a} ed., McGraw-Hill, 1999;
- A. Albano, *Costruire sistemi per basi di dati*, Addison-Wesley Italia, 2001.

1 Gestione della concorrenza

Esercizio 1.1 Siano date le seguenti 3 transazioni:

$\mathbf{t_1}$: $r_1(x), r_1(y), w_1(y)$,

$\mathbf{t_2}$: $r_2(x), w_2(x)$,

$\mathbf{t_3}$: $r_3(x), r_3(z), w_3(z)$.

Determinare (se esistono) uno schedule, non seriale, view-serializzabile e uno schedule, non seriale, conflict-serializzabile delle transizioni considerate.

Soluzione. Osservando attentamente le transazioni, notiamo che:

(a)
- l'elemento x è scritto solo da t_2;
- l'elemento y è scritto solo da t_1;
- l'elemento z è scritto solo da t_3.

Quindi l'insieme delle *scritture finali* sarà sempre il seguente: $SF = \{w_1(y), w_2(x), w_3(z)\}$.

(b) L'unico elemento che può comparire come argomento delle coppie di operazioni lettura-scrittura dell'insieme *legge_da* è x. Infatti l'elemento y viene *prima* letto e *poi* aggiornato all'interno di *un'unica* transazione, e analogamente per z.

Tenendo conto di ciò, un possibile approccio per trovare uno schedule view-serializzabile è quello di considerare uno schedule seriale e di modificarlo preservando la relazione legge_da e le scritture finali. Qualora tutte le possibili modifiche non producano il risultato voluto, si considera un nuovo schedule seriale e si ripete.

$\mathbf{s_1} = \mathbf{t_1 t_2 t_3}$: $r_1(x), r_1(y), w_1(y), r_2(x), w_2(x), r_3(x), r_3(z), w_3(z)$

$LD_1 = \{(r_3(x), w_2(x))\}$

$SF_1 = SF = \{w_1(y), w_2(x), w_3(z)\}$

$\mathbf{s_2}$: $r_1(x), r_1(y), r_2(x), w_1(y), w_2(x), r_3(x), r_3(z), w_3(z)$

$LD_2 = \{(r_3(x), w_2(x))\} = LD_1$

$SF_2 = SF = SF_1$

Controlliamo ora se lo schedule $\mathbf{s_2}$ proposto è anche conflict-serializzabile.
L'insieme dei conflitti è:

$C_2 = \{(r_1(x), w_2(x)), (w_2(x), r_3(x))\}$

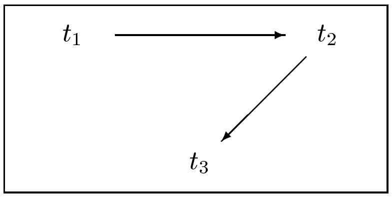


Figura 1: Grafo dei conflitti di $\mathbf{s_2}$

Il grafo dei conflitti è *aciclico* (vedi fig. 1), quindi lo schedule $\mathbf{s_2}$ è conflict-serializzabile.

□

Esercizio 1.2 Siano dati i seguenti schedule:

$\mathbf{s_1}$: $r_2(x), r_1(x), w_1(x), w_2(x)$;

$\mathbf{s_2}$: $r_2(x), r_1(x), w_1(x), r_2(x)$;

$\mathbf{s_3}$: $r_2(y), r_2(z), r_1(x), r_1(z), w_1(z), w_1(x), r_2(x)$;

$\mathbf{s_4}$: $w_1(y), r_2(x), w_1(x), r_2(x), r_3(y), r_0(x), w_0(x), w_2(y)$.

Stabilire se gli schedule dati sono (o meno) view-serializzabili e se sono o meno conflict-serializzabili.

Soluzione. Ricordiamo che $CSR \subsetneq VSR$, perciò $s \in CSR \Rightarrow s \in VSR \wedge s \notin VSR \Rightarrow s \notin CSR$.

$\mathbf{s_1}$: Costruiamo il grafo dei conflitti.

$C_1 = \{(r_2(x), w_1(x)), (r_1(x), w_2(x))\}$

$$\begin{tikzcd} t_1 \arrow[r, shift left] & t_2 \arrow[l, shift left] \end{tikzcd}$$

Figura 2: Grafo dei conflitti di $\mathbf{s_1}$

Il grafo ha un ciclo (vedi fig. 2), quindi $\mathbf{s_1} \notin CSR$.
Controlliamo se $\mathbf{s_1} \in VSR$.
$LD_1 = \varnothing$
$SF_1 = \{w_2(x)\}$
Possiamo derivare dal caso particolare alcuni vincoli che riducono il numero di schedule da analizzare:

- se $t_1 < t_2$ allora $r_2(x)$ legge da $w_1(x)$, e $LD'_1 \neq \varnothing$;
- se $t_2 < t_1$ allora $r_1(x)$ legge da $w_2(x)$, $LD'_1 \neq \varnothing$ e $SF'_1 = \{w_1(x)\}$.

Da ciò, $\mathbf{s_1} \notin VSR$.

$\mathbf{s_2}$: $C_2 = \{(r_2(x), w_1(x)), (w_1(x), r_2(x))\}$, il grafo non è aciclico (vedi fig. 3), quindi $\mathbf{s_2} \notin CSR$.

$$\begin{tikzcd} t_1 \arrow[r, shift left] & t_2 \arrow[l, shift left] \end{tikzcd}$$

Figura 3: Grafo dei conflitti di $\mathbf{s_2}$

$LD_2 = \{(r_2(x), w_1(x))\}$
$SF_2 = \{w_1(x)\}$

- se $t_1 < t_2$ abbiamo due letture da $w_1(x)$, e ricordando che legge_da è un multinsieme, si ha: $LD'_2 = \{(r_2(x), w_1(x)), (r_2(x), w_1(x))\} \neq LD_2$;
- se invece $t_2 < t_1$, legge_da è vuoto.

Non essendoci altri schedule seriali da analizzare, concludiamo che $\mathbf{s_2} \notin VSR$.

$\mathbf{s_3}$: $C_3 = \{(r_2(z), w_1(z)), (w_1(x), r_2(x))\}$, il grafo ha un ciclo (vedi fig. 4), quindi $\mathbf{s_3} \notin CSR$.

$$\begin{tikzcd} t_1 \arrow[r, shift left] & t_2 \arrow[l, shift left] \end{tikzcd}$$

Figura 4: Grafo dei conflitti di $\mathbf{s_3}$

$LD_3 = \{(r_2(x), w_1(x))\}$
$SF_3 = \{w_1(x), w_1(z)\}$
La relazione legge_da impone che t_2 segua t_1, in tal caso, però, anche $r_2(z)$ legge da $w_1(z)$, quindi $\mathbf{s_3} \notin VSR$.

$\mathbf{s_4}$: A volte, fare il controllo di aciclicità durante la costruzione dell'insieme dei conflitti può portare ad un risparmio di tempo:

$C_4 = \{(w_1(y), r_3(y)), (w_1(y), w_2(y)), (r_2(x), w_1(x)), \ldots\}$.

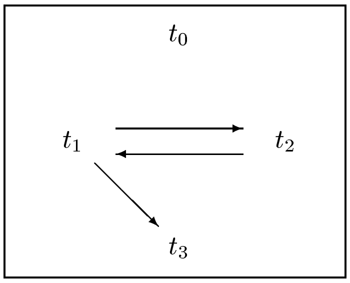


Figura 5: Grafo dei conflitti di $\mathbf{s_4}$

In questo caso i primi tre conflitti generano un ciclo nel grafo di figura 5 e quindi è inutile proseguire: $\mathbf{s_4} \notin CSR$.

$LD_4 = \{(r_2(x), w_1(x)), (r_3(y), w_1(y)), (r_0(x), w_1(x))\}$

$SF_4 = \{w_0(x), w_2(y)\}$

Da LD_4 deriviamo subito questi tre vincoli:

- $t_2 > t_1$;
- $t_3 > t_1$;
- $t_0 > t_1$.

Quindi la transazione t_1 deve essere schedulata per prima.

La transazione t_0 comprende un'operazione di scrittura sull'elemento x, quindi, sempre da LD_4 deduciamo che:

- $t_0 > t_2$.

Gli schedule seriali possibili sono pertanto i seguenti:

- $\mathbf{t_1 t_2 t_0 t_3}$;
- $\mathbf{t_1 t_2 t_3 t_0}$;
- $\mathbf{t_1 t_3 t_2 t_0}$.

Infine notiamo che la transazione t_2 ha *due* operazioni di lettura dell'elemento x, quindi troveremo sempre *due* coppie $(r_2(x), w_1(x))$ nelle relazioni legge_da dei tre schedule seriali candidati. Non essendoci altre possibilità, concludiamo che $\mathbf{s_4} \notin VSR$.

□

Esercizio 1.3 Si considerino i seguenti schedule:

$\mathbf{s_1}$: $r_1(x), r_2(x), w_2(z), w_1(z), r_3(z), w_2(y), r_1(y), w_3(z), w_1(y)$;

$\mathbf{s_2}$: $r_1(z), r_2(x), w_3(x), w_1(z), r_3(z), r_2(z), r_3(y), w_2(z), w_3(y), r_2(y)$.

Stabilire se gli schedule dati sono (o meno) view-serializzabili e se sono (o meno) conflict-serializzabili. In caso di risposta positiva, fornire uno schedule seriale equivalente.

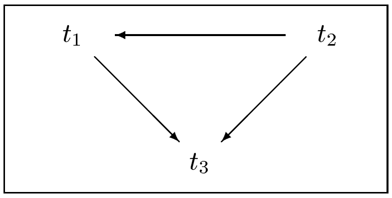


Figura 6: Grafo dei conflitti di $\mathbf{s_1}$

Soluzione.

$\mathbf{s_1}$: Questo schedule è conflict-serializzabile, infatti il grafo dei conflitti è aciclico (vedi fig. 6).

$$C_1 = \{(w_2(z), w_1(z)), (w_2(z), r_3(z)), (w_2(z), w_3(z)), (w_1(z), r_3(z)),\\ (w_1(z), w_3(z)), (w_2(y), r_1(z)), (w_2(y), w_1(y))\}.$$

Ricordando che $CSR \subsetneq VSR$, concludiamo che lo schedule $\mathbf{s_1}$ è anche view-serializzabile. Lo schedule seriale equivalente è $\mathbf{t_2 t_1 t_3}$ e si ottiene facilmente dal grafo dei conflitti.

$\mathbf{s_2}$: Costruiamo l'insieme dei conflitti:

$C_2 = \{(r_1(z), w_2(z)), (r_2(x), w_3(x)), (w_1(z), r_3(z)), (w_1(z), r_2(z)), (w_1(z), w_2(z)), (r_3(z), w_2(z)), \ldots\}$

Ci possiamo fermare perché c'è un ciclo nel grafo (vedi fig. 7), dunque $\mathbf{s_2} \notin CSR$.

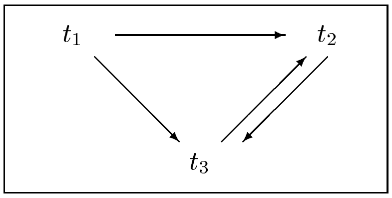


Figura 7: Grafo dei conflitti di $\mathbf{s_2}$

Vediamo ora se $\mathbf{s_2} \in VSR$.

$LD_2 = \{(r_3(z), w_1(z)), (r_2(z), w_1(z)), (r_2(y), w_3(y))\}$

$SF_2 = \{w_3(x), w_3(y), w_2(z)\}$

Dalla relazione legge_da deriviamo i seguenti vincoli:

- $t_1 < t_3$;
- $t_1 < t_2$;
- $t_3 < t_2$.

Esiste solo un possibile schedule seriale: $\mathbf{t_1 t_3 t_2}$, che rispetta anche le scritture finali. Però, essendo $t_3 < t_2$, viene inserita la coppia $(r_2(x), w_3(x))$ non presente in LD_2. Concludiamo che $\mathbf{s_2} \notin VSR$.

□

Esercizio 1.4 Si considerino le seguenti transazioni:

$t_0 : w_0(x)w_0(z)r_0(y)$;

$t_1 : r_1(z)w_1(x)w_1(y)$;

$t_2 : r_2(z)r_2(y)w_2(y)$.

(a) Per $T = \{t_0, t_1, t_2\}$, trovare uno schedule (non seriale) serializzabile rispetto ai conflitti e uno non serializzabile (serializzabilità e non serializzabilità vanno provate).

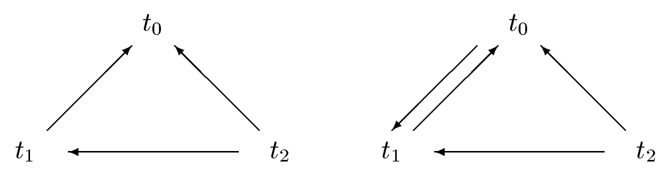
(b) Produrre uno schedule (non seriale) che rispetti il protocollo di locking a due fasi.

Soluzione.

(a) Un possibile approccio per trovare uno schedule conflict serializzabile è quello di modificare uno schedule seriale preservando l'aciclicità del grafo associato.

$\mathbf{s_1}: r_1(z), r_2(z), w_1(x), w_0(x), w_0(z), r_2(y), w_2(y), w_1(y), r_0(y)$

$\mathbf{s_2}: r_1(z), r_2(z), w_0(x), w_1(x), w_0(z), r_2(y), w_2(y), w_1(y), r_0(y)$



(a) Grafo dei conflitti di $\mathbf{s_1}$ (b) Grafo dei conflitti di $\mathbf{s_2}$

Figura 8: Grafi dell'esercizio 1.4

Dal grafo in figura 8(a) possiamo dedurre che $\mathbf{s_1} \in CSR$. Lo schedule seriale equivalente è $\mathbf{t_2 t_1 t_0}$.

Lo schedule $\mathbf{s_2}$ invece non è serializzabile rispetto ai conflitti (fig. 8(b)).

(b) Uno schedule che rispetta il protocollo di locking a 2 fasi è rappresentato in figura 9. Le operazioni segnate con un asterisco provocano la sospensione della transazione che le effettua. Le sequenze di *unlock* di ogni transazione possono essere eseguite in un unico passo.

□

Esercizio 1.5 Si considerino i seguenti schedule:

$\mathbf{s_1}: r_1(x), r_2(x), w_1(x), r_3(x), w_3(x), w_2(y), w_1(y);$

$\mathbf{s_2}: r_2(x), w_2(y), r_1(x), w_1(x), r_3(x), w_3(x), w_1(y);$

$\mathbf{s_3}: r_1(x), r_2(x), w_1(x), r_3(x), w_3(x), w_1(y), w_2(y).$

(a) Per ogni coppia s_i, s_j, con $1 \leq i, j \leq 3$, stabilire se s_i e s_j sono o meno equivalenti rispetto alle viste e se sono o meno equivalenti rispetto ai conflitti.

(b) Stabilire se gli schedule dati sono o meno serializzabili rispetto alle viste e rispetto ai conflitti.

Soluzione.

(a) $\mathbf{s_1}$ e $\mathbf{s_2}$ sono equivalenti rispetto alle viste, le altre due coppie non lo sono:

$LD_1 = \{(r_3(x), w_1(x))\}$ $SF_1 = \{w_3(x), w_1(y)\}$

$LD_2 = \{(r_3(x), w_1(x))\}$ $SF_2 = \{w_3(x), w_1(y)\}$

$LD_3 = \{(r_3(x), w_1(x))\}$ $SF_3 = \{w_3(x), w_2(y)\}$

$\mathbf{s_1}$ e $\mathbf{s_2}$ sono equivalenti anche rispetto ai conflitti, mentre le altre due coppie non lo sono (vedi fig. 10).

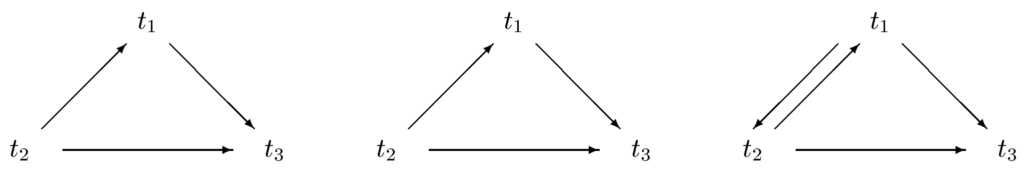
$C_1 = \{(r_1(x), w_3(x)), (r_2(x), w_1(x)), (r_2(x), w_3(x)), (w_1(x), r_3(x)), (w_1(x), w_3(x)), (w_2(y), w_1(y))\}$

$C_2 = \{(r_2(x), w_1(x)), (r_2(x), w_3(x)), (w_2(y), w_1(y)), (r_1(x), w_3(x)), (w_1(x), r_3(x)), (w_1(x), w_3(x))\}$

$C_3 = \{(r_1(x), w_3(x)), (r_2(x), w_1(x)), (r_2(x), w_3(x)), (w_1(x), r_3(x)), (w_1(x), w_3(x)), (w_1(y), w_2(y))\}$

t_0	t_1	t_2
	`bot`	
	$rlock_1(z)$	
	$r_1(z)$	
		`bot`
		$rlock_2(z)^*$
	$wlock_1(x)$	
	$w_1(x)$	
`bot`		
$wlock_0(x)^*$		
	$wlock_1(y)$	
	$w_1(y)$	
	`commit`	
	$unlock_1(z)$	
	$unlock_1(x)$	
	$unlock_1(y)$	
	`eot`	
		$r_2(z)$
$w_0(x)$		
$wlock_0(z)^*$		
		$wlock_2(y)$
		$r_2(y)$
		$w_2(y)$
		`commit`
		$unlock_2(y)$
		$unlock_2(z)$
		`eot`
$w_0(z)$		
$rlock_0(y)$		
$r_0(y)$		
`commit`		
$unlock_0(y)$		
$unlock_0(z)$		
$unlock_0(x)$		
`eot`		

Figura 9: Schedule per l'esercizio 1.4(b)



(a) Grafo dei conflitti di $\mathbf{s_1}$ (b) Grafo dei conflitti di $\mathbf{s_2}$ (c) Grafo dei conflitti di $\mathbf{s_3}$

Figura 10: Grafi dell'esercizio 1.5

(b) Dalle considerazioni fatte al punto (a) possiamo dire che:

- $\mathbf{s_1} \in CSR \Rightarrow \mathbf{s_1} \in VSR$ e lo schedule seriale equivalente è $\mathbf{t_2 t_1 t_3}$;
- $\mathbf{s_2} \in CSR \Rightarrow \mathbf{s_2} \in VSR$ e lo schedule seriale equivalente è $\mathbf{t_2 t_1 t_3}$;
- $\mathbf{s_3} \notin CSR$, quindi si deve controllare se $\mathbf{s_3} \in VSR$.

Ricaviamo da LD_3 e da SF_3 alcuni vincoli:

- $t_1 < t_3$;
- $t_1 < t_2$.

Questi vincoli restringono la scelta dei candidati a:

(i) $\mathbf{t_1 t_2 t_3} : r_1(x), w_1(x), w_1(y), r_2(x), w_2(y), r_3(x), w_3(x)$

$$LD_3' = \{(r_2(x), w_1(x)), (r_3(x), w_1(x))\} \neq LD_3 \quad \text{No.}$$

(ii) $\mathbf{t_1 t_3 t_2} : r_1(x), w_1(x), w_1(y), r_3(x), w_3(x), r_2(x), w_2(y)$

$$LD_3'' = \{(r_3(x), w_1(x)), (r_2(x), w_3(x))\} \neq LD_3 \quad \text{No.}$$

Non essendoci altre possibilità, ne segue che $\mathbf{s_3} \notin VSR$.

$\square$