17.4.1

a) <start T>; <T, A, 10, 11>; <T, B, 20, 21>; <commit T>

I undo/redo-logging er kravet at loggrecordene må være på disk før endringene er på databasen. Mis

LX : Loggrecord skrives til disk (dvs. <T, X, v, w>)

X : Ny verdi av X skrives til disk

C : Commitrecord skrives til disk

er alle lovlige rekkefølger disse:

LA, A, LB, B, C

LA, A, LB, C, B

LA, LB, A, B, C

LA, LB, A, C, B

LA, LB, C, A, B

LA, LB, B, A, C

LA, LB, B, C, A

b) <start T>; <T, A, 10, 21>; <T, B, 20, 21>; <T, C, 30, 31>; <commit T>

Tilsvarende her (la CC stå for commit til disk)

LA, A, LB, B, LC, C, CC

LA, A, LB, B, LC, CC, C

⋮ (antar ikke skrive alle)

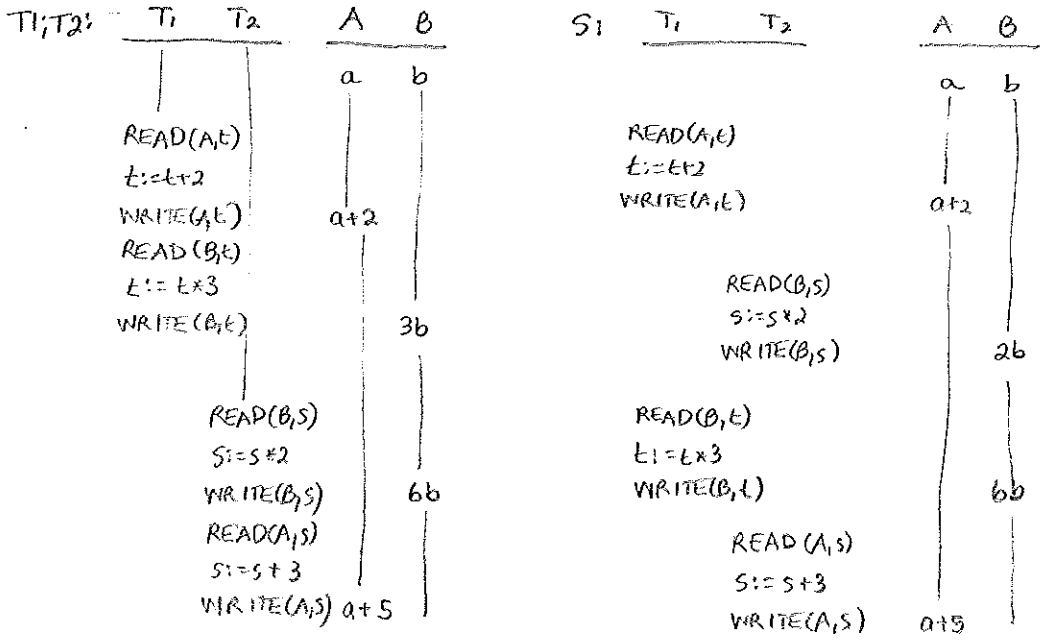
LA, LB, LC, CC, C, B, A

18.2.1

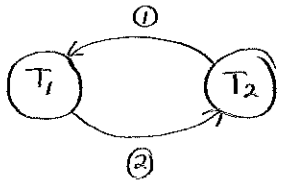
a) Eksempel på en sejalisebar plan:

$S = r_1(A); w_1(A); r_2(B); w_2(B); r_1(B); w_1(B); r_2(A); w_2(A)$

Bevis: Den er ekvivalent med $T_1 ; T_2$:



Merk at S ikke er konfliktserialisebar:



① $S = \ldots w_2(B); r_1(B) \ldots$

② $S = \ldots w_1(A) \ldots r_2(A) \ldots$

Grafen har en sykel.

18.2.1

a) (forts.)

Eksempel på en ikke-serialisebar plan:

$R = r_1(A); w_1(A); r_1(B); r_2(B); w_2(B); w_1(B); r_2(A); w_2(A)$

Bevis:

R:

T_1	T_2	A	B
		a	b
READ(A,t)			
t:=t+2			
WRITE(A,t)		a+2	
READ(B,t)			b
t:=t*3			
	READ(B,s)		b
	s:=s*2		
	WRITE(B,s)		2b
WRITE(B,t)			3b
	READ(A,s)		
	s:=s+3		
	WRITE(A,s)	a+5	

Det endelige resultatet av R er ikke lik $T_1; T_2$ (se forrige side), og heller ikke lik $T_2; T_1$:

$T_2; T_1$

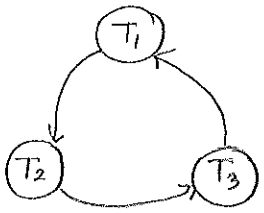
T_1	T_2	A	B
		a	b
	READ(B,s)		
	s:=s*2		
	WRITE(B,s)		2b
	READ(A,s)		
	s:=s+3		
	WRITE(A,s)	a+3	
READ(A,t)			
t:=t+2			
WRITE(A,t)		a+5	
READ(B,t)			
t:=t*3			
WRITE(B,t)			6b

d) Se under a) hvor vi har vist $T_1; T_2$ og $T_2; T_1$ på en vilkårlig initialtilstand (a,b).

18.2.5

a)

$w_3(A); r_1(A); w_1(B); r_2(B); w_2(C); r_3(C)$ konflikter



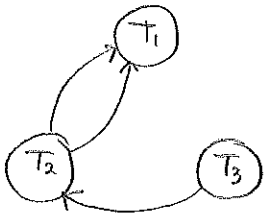
Sykel, så planen er ikke konfliktserialiserbar

Er den likevel ekvivalent med en av $T_1; T_2; T_3 \mid T_1; T_3; T_2 \mid T_2; T_3; T_1 \mid \ldots$ uansett hva som skjer med dataene?

Hvis T_1 beregner ny verdi i B på grunnlag av hva den leser i A, er det ikke ugyldig om T_3 har skrevet ny verdi i A før eller etter T_1 leser den. Tilsvarende argumentasjon gjelder for T_2 og T_3.
Så, nei: Denne planen er likevel ikke ekvivalent med noen seriell plan.

18.2.5

d) $r_1(A); r_2(A); r_3(B); w_1(A); r_2(c); r_2(B); w_2(B); w_1(C)$



Grafen er sykelfri, så planen er konfliktserialiserbar
(Planen er ekvivalent med $T_3; T_2; T_1$)

Siden T_3 bare leser verdier, har den ingen innvirkning på databasetilstanden. Så faktisk vil $T_2; T_1; T_3$ ha samme effekt på databasen, ergo er den ekvivalent, men den er ikke konfliktekvivalent med planen over (Men applikasjonen som initierte T_3, vil generelt få en annen avlesning fra planen $T_2; T_1; T_3$ enn den som den opprinnelige planen gav!)

18,3.2

a) T_1: $l_1(A); r_1(A); l_1(B); w_1(B); u_1(A); u_1(B)$

T_2: $l_2(B); r_2(B); l_2(C); w_2(C); u_2(B); u_2(C)$

T_3: $l_3(A); w_3(A); l_3(C); r_3(C); u_3(A); u_3(C)$

T_1	T_2	T_3
		$l_3(A)$
		$w_3(A)$
$l_1(A)$ – vede		
	$l_2(B)$	
	$r_2(B)$	
	$l_2(C)$	
	$w_2(C)$	
	$u_2(B)$	
	$u_2(C)$	
		$l_3(C)$
		$r_3(C)$
		$u_3(A)$
$l_1(A)$ – imulget		
		$u_3(C)$
$r_1(A)$		
$l_1(B)$		
$w_1(B)$		
$u_1(A)$		
$u_1(B)$		

18.3.2

d)

T_1: $L_1(A); r_1(A); w_1(A); L_1(C); w_1(C); u_1(A); u_1(C)$

T_2: $L_2(A); r_2(A); L_2(C); r_2(C); L_2(B); r_2(B); w_2(B); u_2(A); u_2(C); u_2(B)$

T_3: $L_3(B); r_3(B); u_3(B)$

T_1	T_2	T_3
$L_1(A)$		
$r_1(A)$		
	$L_2(A)$ - wait	
		$L_3(B)$
		$r_3(B)$
		$u_3(B)$
$w_1(A)$		
$L_1(C)$		
$w_1(C)$		
$u_1(A)$		
$u_1(C)$		
	$L_2(A)$ - hldell	
	$r_2(A)$	
	⋮	
	$u_2(B)$	

Oppgave 2

d1	0	0	1
d2	0	1	0
d4	1	0	0
d3	0	1	1
d5	1	0	1
d6	1	1	0
d7	1	1	1

a) $d1 = d3 \oplus d5 \oplus d7$ =
$$\begin{array}{r} 00001111 \\ \oplus\ 00000100 \\ \oplus\ 11000101 \\ \hline = 11001110 \end{array}$$

$d2 = d3 \oplus d6 \oplus d7$ =
$$\begin{array}{r} 00001111 \\ \oplus\ 10011011 \\ \oplus\ 11000101 \\ \hline = 01010001 \end{array}$$

$d4 = d5 \oplus d6 \oplus d7$ =
$$\begin{array}{r} 00000100 \\ \oplus\ 10011011 \\ \oplus\ 11000101 \\ \hline = 01011010 \end{array}$$

b) Siden d2 og d4 er konstruert på grunnlag av d6, må – i tillegg til endringen på d6 – disse diskene endres. Nytt innhold:

ny d2 = gammel d2 $\oplus$ gammel d6 $\oplus$ ny d6 =
$$\begin{array}{r} 01010001 \\ \oplus\ 10011011 \\ \oplus\ 10000100 \\ \hline = 01001110 \end{array}$$

ny d4 = gammel d4 $\oplus$ gammel d6 $\oplus$ ny d6 =
$$\begin{array}{r} 01011010 \\ \oplus\ 10011011 \\ \oplus\ 10000100 \\ \hline = 01000101 \end{array}$$

I praksis kan man først beregne

gammel d6 $\oplus$ ny d6

og legge denne til både d2 og d4.

Oppgave 2 (forts)

c) d5 kan rekonstrueres på ett av følgende vis:

$d4 \oplus d6 \oplus d7$

$d1 \oplus d3 \oplus d7$

d) d6 kan rekonstrueres fra $d4 \oplus d5 \oplus d7$ eller fra $d2 \oplus d3 \oplus d7$.
Men siden d7 også er kresjet, går ikke dette.

d7 kan rekonstrueres fra $d4 \oplus d5 \oplus d6$, $d2 \oplus d3 \oplus d6$ eller $d1 \oplus d3 \oplus d5$.
Siden d6 er kresjet, bruker vi den siste: $d1 \oplus d3 \oplus d5$.

Når d7 er rekonstruert, kan vi rekonstruere også d6, fra
f. eks. $d4 \oplus d5 \oplus d7$.