Esercizio 1
Indicare se i seguenti schedule sono VSR.
1. r1(x), r2(y), w1(y), r2(x), w2(x)
2. r1(x), r2(y), w1(x), w1(y), r2(x), w2(x)
3. r1(x), r1(y), r2(y), w2(z), w1(z), w3(z), w3(x)
4. r1(x), r4(x), w4(x), r1(y), r4(z), w4(z), w3(y), w3(z), w1(t), w2(z), w2(t)

Soluzione:
1) r1(x), r2(y), w1(y), r2(x) w2(x)
Questo schedule non è VSR, perché i due schedule seriali:
S1: r1(x), w1(y), r2(y), r2(x), w2(x) e
S2: r2(y), r2(x), w2(x), r1(x), w1(y)
non sono view-equivalenti con lo schedule dato. Hanno entrambi una differente relazione LEGGE - DA

2) r1(x), r2(y), w1(x), w1(y), r2(x) w2,(x)
Questo schedule non è VSR perché gli schedule seriali
S1: r1(x), w1(x), w1(y), r2(y), r2(x), w2(x)
S2: r2(y), r2(x), w2(x), r1(x), w1(x),w1(y)

hanno entrambi una differente relazione LEGGE – DA

3) r1(x), r1(y), r2(y), w2(z), w1(z), w3(z), w3(x)
Questo schedule è VSR e view-equivalente allo schedule seriale
S: r2(y), w2(z), r1(x), r1(y), w1(z), w3(z), w3(x)

sono caratterizzati dalle stesse scritture finali e dalle stesse relazioni LEGGI –DA.

4) r1(x), r4(x), w4(x), r1(y), r4(z), w4(z), w3(y), w3(z), w1(t), w2(z), w2(t)

Consideriamo le operazioni relative a ogni risorsa separatamente:
T: w1, w2
X: r1, r4, w4
Y: r1, w3
Z: r4, w4, w3, w2
Visto che il grafo è aciclico lo schedule è CSR, quindi anche VSR.

Lo schedule seriale equivalente è:
S: r1(x), r1(y), w1(t), r4(x), w4(x), r4(z), w4(z), w3(y), w3(z), w2(z), w2(t)

Esercizio 2
Indicare se i seguenti schedule possono produrre anomalie; i simboli ci e ai indicano l'esito (commit o abort) della transazione i.

1. r1(x), w1(x), r2(x), w2(y), a1, c2

2. r1(x), w1(x), r2(y), w2(y), a1, c2
3. r1(x), r2(x), r2(y), w2(y), r1(z), a1, c2
4. r1(x), r2(x), w2(x), w1(x), c1, c2
5. r1(x), r2(x), w2(x), r1(y) , c1, c2
6. r1(x), w1(x), r2(x), w2(x) , c1, c2

Soluzione:
1) r1(x), w1(x), r2(x), w2(y), a1, c2
L'operazione r2(x) legge il valore scritto da w1(x), ma la transazione 1 termina con un abort. Questo è un caso di lettura sporca (Dirty Read) e anche la transazione 2 deve essere abortita.

2) r1(x), w1(x), r2(y), w2(y), a1, c2
Questo schedule non produce anomalie, perché le due transazioni fanno riferimento a oggetti differenti.

3) r1(x), r2(x), r2(y), w2(y), r1(z), a1, c2
Questo schedule non produce anomalie, perché la transazione 1 che termina in abort non effettua operazioni di scrittura.

4) r1(x), r2(x), w2(x), w1(x), c1, c2
Questo schedule ha una perdita di aggiornamento(lost update), in quanto gli effetti della transazione 2 vengono persi.

5) r1(x), r2(x), w2(x), r1(y) , c1, c2
Questo schedule non produce anomalie.

6) r1(x), w1(x), r2(x), w2(x) , c1, c2
Questo schedule non produce anomalie

Esercizio 3

Se i seguenti schedule si presentassero a uno scheduler che usa il locking a due fasi, quali transazioni verrebbero messe in attesa? Una volta posta in attesa una transazione, le sue successive azioni non vanno più considerate.

Soluzione:

1) r1(x), r3(y), w1(y), w4(x), w1(t), w5(x), r2(z), r3(z), w2(z), w5(z), r4(t), r5(t)
Le transazioni 1, 4, 5 e 2 sono in attesa. La transazione 1 deve aspettare per y (allocato da 3), le transazioni 4 e 5 devono aspettare per x (allocato da 1) e la transazione 2 deve aspettare per z (allocato da 3). Consideriamo che tutte le transazioni terminino con successo, qual'è un possibile ordine dei commit?

r1(x), r3(y), w1(y), w4(x), w1(t), w5(x), r2(z), r3(z), w2(z), c3, c2, c1, w5(z), r4(t),c4, r5(t), c5

2) r1(x), r2(x), w2(x), r3(x), r4(z), w1(x), r3(y), r3(x), w1(y), w5(x), w1(z), r5(y), r5(z)
Le transazioni 2, 1 e 5 sono in attesa. Devono aspettare per x (allocata da 1, 2 e 3), 3 termina ma le altre no

3) r1(x), r1(t), r3(z), r4(z), w2(z), r4(x), r3(x), w4(x), w4(y), w3(y), c3, w1(y), c1, w2(t), c2, c4
Le transazioni 2 e 4 vengono messe in attesa. 2 per z (allocata da 3) e 4 per x (allocata da 1 e 3)

4) r1(x), r4(x), w4(x), r1(y), r4(z), w4(z), w3(y), w3(z), w1(t), c1, w2(z), w2(t), c2, c4, c3
Le transazioni 3 e 4 sono in attesa. Devono aspettare per x e y, allocate da 1.

Esercizio 4

Se gli schedule precedenti si presentassero a uno scheduler basato su timestamp, quali transazioni verrebbero abortite? RTM e WTM valgono 0 per ogni risorsa all'inizio. I TS corrispondono al numero della transazione.

Soluzione:

1) r1(x), r3(y), w1(y), w4(x), w1(t), w5(x), r2(z), r3(z), w2(z), w5(z), r4(t), r5(t)

Operazione	Risposta	Nuovo Valore
read(x,1)	Ok	RTM(x)=1
read(y,3)	Ok	RTM(y)=3
write(y,1)	1 abortita	
write(x,4)	Ok	WTM(x)=4
write(x,5)	Ok	WTM(x)=5
read(z,2)	Ok	RTM(z)=2
read(z,3)	Ok	RTM(z)=3
write(z,2)	2 abortita	
write(z,5)	Ok	WTM(z)=5
read(t,4)	Ok	RTM(t)=4
read(t,5)	Ok	RTM(x)=5

2) r1(x), r2(x), w2(x), r3(x), r4(z), w1(x), r3(y), r3(x), w1(y), w5(x), w1(z), r5(y), r5(z)

Operazione	Risposta	Nuovo Valore
read(x,1)	Ok	RTM(x)=1
read(x,2)	Ok	RTM(x)=2
write(x,2)	Ok	WTM(x)=2
read(x,3)	Ok	RTM(x)=3
read(z,4)	Ok	RTM(z)=4

write(x,1)	1 abortita	
read(y,3)	Ok	RTM(y)=3
read(x,3)	Ok	RTM(x)=3
write(x,5)	Ok	WTM(x)=5
read(y,5)	Ok	RTM(y)=5
read(z,5)	Ok	RTM(y)=5

3) r1(x), r1(t), r3(z), r4(z), w2(z), r4(x), r3(x), w4(x), w4(y), w3(y), w1(y, w2(t)

Operazione	Risposta	Nuovo Valore
read(x,1)	Ok	RTM(x)=1
read(t,1)	Ok	RTM(t)=1
read(z,3)	Ok	RTM(z)=3
read(z,4)	Ok	RTM(z)=4
write(z,2)	2 abortita	
read(x,4)	Ok	RTM(x)=4
read(x,3)	3 abortita	
write(x,4)	Ok	WTM(x)=4
write(y,4)	Ok	WTM(y)=4
write(y,1)	1 abortita	

4) r1(x), r4(x), w4(x), r1(y), r4(z), w4(z), w3(y), w3(z), w1(t), w2(z), w2(t)

Operazione	Risposta	Nuovo Valore
read(x,1)	Ok	RTM(x)=1
read(x,4)	Ok	RTM(x)=4
write(x,4)	Ok	WTM(x)=4
read(y,1)	Ok	RTM(y)=1
read(z,4)	Ok	RTM(z)=4
write(4,z)	OK	WTM(z)=4
write (y,3)	Ok	WTM(y)=3
write (z,3)	3 abortita	
write (t, 1)	OK	WTM(t)=1
write(z,2)	2 abortita	
write (t, 2)	OK	WTM(t)=2

Esercizio 5

Si supponga che si verifichi un guasto di dispositivo che coinvolge gli oggetti O2, O3; descrivere la ripresa a freddo.

DUMP, B(T1), B(T2), B(T3), I(T1, O1, A1), I(T2, O2, A3), B(T4), U(T4, O3, B4, A4), U(T2, O2, B2, A2), U(T1, O4, B5, A5), C(T2), CK(T1, T3, T4), B(T5), B(T6), U(T5, O5, B6, A6), A(T3),

CK(T1, T4, T5, T6), B(T7), *A*(T4), U(T7, *O*6, B7, *A*7), *C*(T7), U(T6, *O*3, B8, *A*8), B(T8), U(T8, *O*3, B9, *A*9), *guasto*

Soluzione:

La ripresa a freddo è articolata in tre fasi successive.

1. Il log viene percorso a ritroso fino al primo record DUMP e si ricopia selettivamente la parte deteriorata della base dati.

2. Si ripercorre in avanti il log, applicando relativamente alla parte deteriorata della base di dati sia le azioni sugli oggetti sia le azioni di commit o abort, riportandosi così nella situazione precedente al guasto.

$O_2=A_3$
$O_3=A_4$
$O_2=A_2$
Commit (T_2)
Abort (T_3)
Abort (T_4)
Commit (T_7)
$O_3=A_8$
$O_3=A_9$

3. Si svolge la ripresa a caldo.

1) Per prima cosa bisogna percorrere il log a ritroso fino al più recente record di check-point: *CK*(T1,T4,T5,T6)
Si costruiscono gli insiemi di UNDO e di REDO: UNDO= { T_1, T_4, T_5, T_6 }, REDO={}

2) Il log viene percorso in avanti, aggiornando i due insiemi:
B(T_7) UNDO= { T_1, T_4, T_5, T_6, T_7 } REDO={}
A(T_4) UNDO= { T_1, T_4, T_5, T_6, T_7 } REDO={}
C(T_7) UNDO= { T_1, T_4, T_5, T_6 } REDO={ T_7}
B(T_8) UNDO= { T_1, T_4, T_5, T_6, T_8 } REDO={ T_7}

3) Il log viene ripercorso ancora a ritroso, fino all'operazione I(T1,O1,A1), che è la più vecchia eseguita da una transazione nei due insiemi, eseguendo le seguenti operazioni (operazioni eseguite da transazioni nell'insieme UNDO):

$O_3=B_9$
$O_3=B_8$
$O_5=B_6$
$O_4=B_5$

O_3=B_4
Delete O_1

4) Il log viene ripercorso in avanti per rieseguire le operazioni eseguite da transazioni nell'insieme REDO.

O_6=A_7