PROVA 1

1)
a) Última instrução do T1

b)
lógica: t1 sempre começa e t3 e t4 sempre serão depois do t2

t1 t2 t3 t5 t4
t1 t2 t3 t4 t5
t1 t2 t4 t5 t3
t1 t2 t4 t3 t5
*tem mais 4, são 8 no total

pela lógica, os outros 4 são:
t1 t5 t2 t3 t4
t1 t5 t2 t4 t3
t1 t2 t5 t3 t4
t1 t2 t5 t4 t3

2)
t3 r(c)
t3 w(b)
t2 w(d)
t2 r(b)
t1 r(f)
t1 r(d)
t3 abort
t2 abort
t1 abort

3)
t1 r(a)
t2 r(b)
t3 r(c)
t4 r(d)
t5 r(e)
t2 abort
t4 abort
t5 r(a)
t1 w(b)
t3 w(d)
t5 commit
t1 r(c)
t3 w(e)
t1 commit

t2 r(b)
t3 commit
t4 r(d)
t2 w(c)
t4 w(e)
t2 commit
t4 commit

4)
t1 r(a)
t2 r(b)
t3 r(c)
t4 r(d)
t5 r(e)
t5 r(a)
t5 commit
t4 w(e=4)
t4 commit
t3 w(d=3)
t3 w(e=10)
t3 commit
t2 w(c=2)
---
t2 commit
t1 w(b=1)
t1 r(c)
t1 commit

a)
t1 start
t2 start
t3 start
t4 start
t5 start
t5 commit
t4 (e, 0, 4)
t4 commit
t3 (d, 0, 3)
checkpoint(t1, t2, t3)
t3 (e, 4, 10)
t3 commit
t2 (c, 0, 2)
---
t2 commit
t1 (b, 0, 1)
t1 commit

b)
undo
c = 0
redo
e = 10

5)
seriable
repeatable read
read committed
read uncommitted

6)
Protocolo estrito. Ele segura os bloqueios exclusivos até o momento do commit/abort. Na versão original o desbloqueio pode acontecer a qualquer momento desde que não se faça nenhum bloqueio depois.

7)
O starvation ocorre quando uma transação costuma ser preterida no momento de acessar um item. Uma forma de evitar é assegurar que transações mais velhas tenham prioridade no acesso aos itens.

PROVA 2

1)
arestas do grafo =
1->2
2->4
3->1
2->5
3->5
1->5
4->5

a) t3 t1 t2 t4 t5

b) mudando de read(c) para read(f)

tira aresta t3->t5
add aresta t2->t3

(geraria ciclo entre t1 t2 t3)

2) A transação t3 commitou um valor inválido pois ele é originado por uma transação que abortou

t1 r(b)
t2 r(a)
t3 r(a)
t1 w(c)
t2 w(b)
t3 r(b)
t3 commit
t2 abort

3)
t1 r(a)
t2 r(b)
t3 r(c)
t4 r(d)
t5 r(e)
t4 w(f)
t5 abort
t4 commit
t5 r(e)
t3 w(d)
t5 r(f)
t3 w(g)
t5 abort
t3 commit
t5 r(e)
t2 w(c)
t5 r(f)
t2 commit
t5 w(c)
t1 r(b)
t5 commit
t1 r(g)
t1 commit

4)
1 r(a)
2 r(b)
3 r(c)
4 r(d)
5 r(e)
4 w(f=4)
4 commit
5 r(f)
3 w(d=3)
3 w(g=9)

3 commit
2 w(c=2)
---
t2 commit
t5 w(c=5)
t1 ??
t5 commit
t1 ??
t1 commit

a)
t1 start
t2 start
t3 start
t4 start
t5 start
t4 f 4
t4 commit
t3 d 3
t3 g 9
t3 commit
t2 c 2
---
t2 commit
t5 c 5
t1 b 1
t5 commit
t1 commit

b)
redo(t3, t4)
f = 4
d = 3
g = 9

5) O problema ocorre se uma transação fizer a leitura de um mesmo item mais do que uma vez. Entre as leituras, outra transação pode ter modificado o valor do item, o que faz com que as duas leituras acessem valores diferentes. Essa anomalia é chamada de leitura não repetida.

6) Uma schedule não recuperável é um cujas mudanças não podem ser desfeitas porque ela já foi confirmada. Se as mudanças levam um banco a um estado inconsistente, ele permanecerá assim. Schedules recuperáveis são imprescindíveis porque não se pode abrir mão da consistência.

O problema do rollback em cascata leva a perda de performance mas não da consistência, por isso é interessante ser livre de rollbacks em cascata, apesar de não ser algo imprescindível.

7) O gerenciador de acesso concorrente mantém um grafo de espera que indica quais transações estão aguardando pela liberação de recursos. Ciclos no grafo indicam uma espera circular, onde nenhuma das transações pode prosseguir, o que configura um deadlock. Para eliminar o deadlock deve abortar alguma transação daquele ciclo.

PROVA 3

1)

T1-T3 (T3-T1)
T1-T2
T2-T5
T2-T4
T4-T5
T3-T5
T1-T5

a) trocar o write(a) de t1 com read(a) de t3
fica:
1->2
1->3 (aparece)
1->5
3->5
4->5
2->4
2->5

a) t1 t2 t3 t4 t5
t1 t3 t2 t4 t5

2)
t1 r(a)
t2 w(a)
t2 commit
t1 w(a)
t1 commit
t3 w(a)
t3 commit
equivalente ao serial t1 t2 t3

regras para equivalência em visão:

- a última escrita em a é feita pela mesma transação (t3)
- a leitura de a em t1 recebe a mesma informação

3)
t1 r(a)
t3 r(c)
t3 w(b)
t3 abort
t1 w(c)
t1 r(d)
t1 commit
t2 w(a)
t3 r(c)
t3 w(b)
t3 abort
t2 w(c)
t2 commit
t3 r(c)
t4 r(a)
…
mais algumas coisas
…
t4 w(alguma coisa)
t4 commit
t3 w(a)
t3 commit

4)
t1 r(a)
t3 r(c)
t3 w(b=5)
t3 commit
t1 w(c=3)
t1 r(d)
t1 commit
t2 w(a=1)
t2 w(c=4)
---

t2 commit
t4 r(a)
t4 w(c=6)
t4 commit

a) t3 t1 t2 t4

b)
t1 start
t2 start
t3 start
t4 start
t3 b 0 5
t3 commit
t1 c 0 3
checkpoint(t1 t2 t4)
t1 commit
t2 a 0 1
t2 c 3 4
---
t2 commit
t4 c 4 6
t4 commit

c)
Undo (t2,t4)
c= 3
a = 0
redo(t1)
t1 = não faz nada;

5)
1 r(a)
2 w(a)
2 commit
1 r(a)
1 commit

No schedule acima, as duas leituras feitas em t1 podem receber valores diferentes. Essa anomalia é chamada de leitura não repetida.

PROVA 4

1)

a)
1 wa=10 2 rf 1 wb=30 3 wc=5 1 wd=20 1 com 2 wd=10 2 com 3 wb=20 --- 3 wf=30 3 com
undo(t3) b=20 c=0 redo(t1 t2) a=10 …

b) **DEU BUG {**
t1 w(a=10)
t2 r(**b**)
t1 w(b=30)
t3 w(c=5)
t1 w(d=20)
t1 commit
t2 w(d=10)
t2 commit
t3 w(b=20)
t3 w(f=30)
t3 commit

o ciclo no grafo abaixo mostra que não é equivalente em conflito a um schedule serial
grafo com arestas 1->2 1->3 2->3 e criou 2->1

mas é equivalente em visão ao schedule serial t1 t2 t3
nos dois schedules:

- o última transação que grava em B é a mesma (t3)
- a leitura de B em T2 recebe o mesmo valor

T1-T2
T1-T3
T2-T3
(T2-T1) <- AQUI DA RUIM
} FIM DO DEU BUG

Por que deu ruim?: “Nesse schedule concorrente read(b) está lendo o valor anterior de b que não foi modificado nem por t1 nem por t3 e no schedule serial o t2 estaria lendo um read(b) modificado por t1, não satisfaz a regra: erro da mesma origem.”

2)
t1 wa)
t2 r(a)
t2 w(b)
t3 r(b)
t3 w(c)
t4 r(c)
t1 abort
t2 abort

t3 abort
t4 abort

“Tem que aparecer três aborts em cascata, o abort em uma causou o abort em outra que causou em outra que causou em outra.”

3)
t1 r(a)
t3 r(c)
t3 w(b)
t3 abort
.... mais muita coisa

4)
a)
t1 r(a)
t2 w(a)
t3 r(c)
t4 r(a)
t1 w(c)
t2 w(c)
t3 w(b)
t4 w(d)
t1 r(d)
t2 commit
t3 w(a)
t4 commit
t1 commit
t3 commit

b) No schedule abaixo a falha em t2 levaria a reversão da transação t4 já que essa transação leu um valor de a gerado por t2.
t1 r(a)
t2 w(b)
t3 r(c)
t4 r(a)
t1 w(c)
t2 w(c)
t3 w(b)
t4 w(d)
t1 r(d)
t2 abort
t4 abort
(T1 tbm pode ser abortado ao final já que o T4 abortou)

5) Com um timeout muito curto é possível que a transação seja abortada sem que ela tenha entrado em um sistema de deadlock, o que acarreta em um custo de reversão desnecessário.
Com um timeout muito longo, é possível que demore para que uma transação em um sistema de deadlock seja revertida, o que leva a um tempo de espera demasiado.