21103 - Sistemas de Gestão de Bases de Dados
2015-2016
e-fólio B

Resolução e Critérios de Correção

PARA A RESOLUÇÃO DO E-FÓLIO, ACONSELHA-SE QUE LEIA ATENTAMENTE O SEGUINTE:

1) O e-fólio é constituído por 3 perguntas. A cotação global é de 3 valores.

2) O e-fólio deve ser entregue num único ficheiro PDF, não zipado, com fundo branco, com perguntas numeradas e sem necessidade de rodar o texto para o ler. Penalização de 1 a 3 valores.

3) Não são aceites e-fólios manuscritos, i.e. tem penalização de 100%.

4) O nome do ficheiro deve seguir a normal "eFolioB" + <nº estudante> + <nome estudante com o máximo de 3 palavras>

5) Durante a realização do e-fólio, os estudantes devem concentrar-se na resolução do seu trabalho individual, não sendo permitida a colocação de perguntas ao professor ou entre colegas.

6) A interpretação das perguntas também faz parte da sua resolução, se encontrar alguma ambiguidade deve indicar claramente como foi resolvida.

7**)** A legibilidade, a objectividade e a clareza nas respostas serão valorizadas, pelo que, a falta destas qualidades serão penalizadas.

Vetor Cotações

1	2	3	pergunta
10	10	10	décimas

Critérios de correção gerais: todas as respostas devem ser justificadas, incluir imagens e exemplos com vista a clarificar os argumentos expostos.

1) Relativo ao Cap. 14- Transações
Defina serialização por conflitos e por vistas. O escalonamento R1(x), R2(x), R3(y), W2(x), R1(y), W1(y), R3(x), W3(z) é serializável a vistas ou a conflitos. Porquê?

Resposta:

1.a) Defina serialização por conflitos e por vistas.

Serialização por conflitos: um escalonamento é serializável por conflitos se o seu grafo de precedência não contiver ciclos.

Na construção do grafo de precedências as arestas são montadas a partir das observações das transações que participa, da escala sendo duas transação T_i e T_j haverá uma aresta: $T_i \rightarrow T_j$ se forem observadas as seguintes condições:

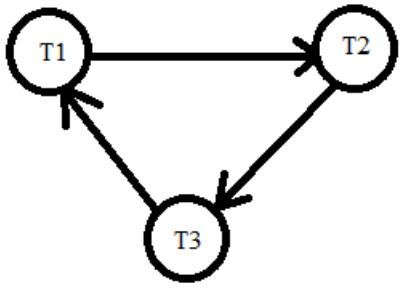
- T_i executa *write*(q) antes de T_j executar *read*(q)
- T_i executa *read*(q) antes de T_j executar *write*(q)
- T_i executa *write*(q) antes de T_j executar *write*(q)

Serialização por vistas: Diz-se que "Ti lê x de Tj", se Wj(x) for a última operação de escrita antes de Ri(x). Dois escalamentos são equivalentes a vistas, se a relação "lê de" em ambas for a mesma [Feliz Gouveia 2014].
Dois escalonamentos S1 e S2 são equivalentes se:
(1) Se S1: $r_i(A)$ lê o valor inicial, então S2: $r_i(A)$ também lê o mesmo valor inicial
(2) Se S1: $w_j(A) \Rightarrow r_i(A)$ então em S2: $w_j(A) \Rightarrow r_i(A)$; têm a mesma vista
(3) Se S1: T_i termina $w_i(A)$, então S2: T_i também termina com $w_i(A)$; os valores finais são os mesmos

1.b) O escalonamento é serializável a vistas ou a conflitos? Porquê?
Divisão por recurso/item:
Item x: R1, R2, W2, R3
Item y: R3, R1, W1
Item z: W3

As precedências são as seguintes:
R1, R2, W2, R3: R1->W2, W2->R3
R3, R1, W1 : R3->W1

Graficamente obtemos um grafo cíclico, pelo que não é serializável a conflitos.



Para analisar se o escalonamento é serializável por vistas, devemos criar um ou mais escalonamentos em série, e de seguida verificar se cumpre a regra "Ti lê x de Tj".

O escalonamento concorrente é o dado no problema. Criámos dois escalonamentos em série:

- Série-1 (S1): verificamos que R1(y) e R3(y) no escalonamento concorrente são iniciados com o mesmo valor, o que não acontece em S1 visto que existe um W1(y).

- Série-2 (S2): no escalonamento S2 verificamos que R2(x) e R1(x) são iniciados com o mesmo valor, o que não acontece com S2 visto que existe um W2(x).

T1	**T2**	**T3**
R(x)		
	R(x)	
		R(y)
	W(x)	
R(y)		
W(y)		
		R(x)
		W(z)
	concorrente	

T1	**T2**	**T3**
R(x)		
R(y)		
W(y)		
	R(x)	
	W(x)	
		R(y)
		R(x)
		W(z)
	série 1	

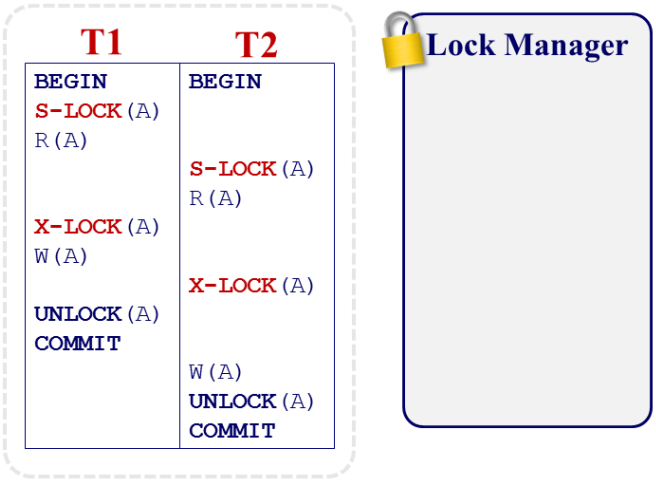
T1	**T2**	**T3**
		R(y)
		R(x)
		W(z)
	R(x)	
	W(x)	
R(x)		
R(y)		
W(y)		
	série 2	

Conclusão: o escalonamento não é serializável por conflito e não é serializável por vistas.

Critério de correção:
- (0,5) parte 1.a) definições do escalonamento seriável a conflitos e a vistas
- (0,5) parte 1.b) estudo detalhado do problema

2) Relativo ao Cap. 15- Concorrência

Defina o protocolo 2-PL. Considere o protocolo 2-PL e explique detalhadamente a execução das seguintes transações.



Resposta:

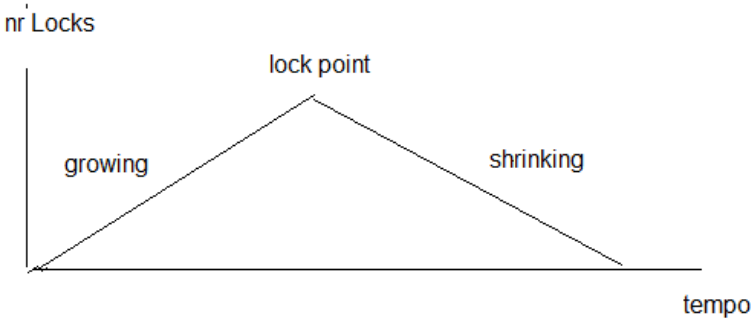
2.a) Defina o protocolo 2-PL.

Existem dois tipos de "locks" para as operações de Read e Write, os partilhados (Shared-lock ou S-lock) e exclusivos (eXclusive-lock ou X-lock).

O protocolo "two-phase locking" (2PL) assegura o controlo da concorrência e garante sequenciamentos conflito - serializáveis. O 2PL é composto, como a própria designação indica, por duas fases:

Fase 1: Fase de Crescimento/Expansão (*growing*)
a transação pode obter *locks* e não pode libertar *locks*

Fase 2: Fase de Encolhimento/Contenção (*shrinking*)
a transação pode libertar *locks* e não pode obter *locks*

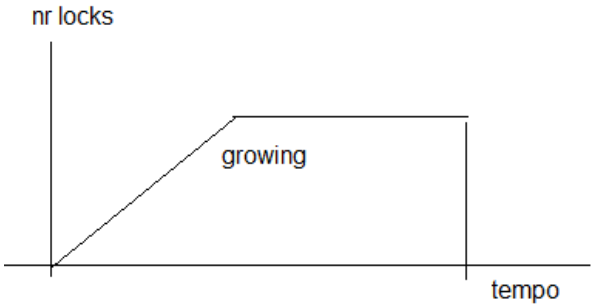


Para a transação T1: R(X), W(X), R(Y),W(Y) teremos um 2PL

T1: S-lock(X), R(X), X-lock(X), W(X), S-lock(Y), R(Y), X-lock(Y), W(Y), Lock-Point, Commit, Unlock(X), Unlock(Y)

O protocolo "two-phase locking", dado que é muito restritivo, é muito usual a ocorrência de "deadlocks". No "deadlock" pelo menos uma das transações tem de ser abortada. Se estiverem envolvidas várias transações "cascading rollback (abort)" irá ocorrer.

Para o evitar, pode utilizar uma variante estrita deste protocolo denominado "strict two-phase locking", onde a transação retém todos os seus *locks* eXclusivos até que efetue *Commit* ou *Abort*.



T1: S-lock(X), R(X), X-lock(X), W(X), S-lock(Y), R(Y), X-lock(Y), W(Y), Lock-Point, Unlock-All, Commit,

2.b) Execução das transações.

A gestão dos *locks* é realizada pelo *Lock Manager* do SGBD. Quando existe um pedido de *lock* verificar o estado do item:

- se não existe *lock* do item, é dada permissão de *lock* (*Grant*)
- caso contrário, existindo um *lock* do item, o pedido fica em fila de espera (*Enqueue*); quando o item é libertado o pedido sai da fila de espera (*Dequeue*)

	T1	T2	Gestão dos locks
1	BEGIN	BEGIN	
2	S-LOCK(A)		Grant (T1, S, A) permite o acesso partilhado da transação T1 ao item A
3	R(A)		
4		S-LOCK(A)	Grant (T2, S, A) permite o acesso partilhado da transação T2 ao item A
5		R(A)	
6	X-LOCK(A)		Grant (T1, X, A) permite o acesso exclusivo da transação T1 ao item A
7	W(A)		
8		X-LOCK(A)	existe lock do item A, pedido vai para fila de espera, Enqueue(A)
9	UNLOCK(A)		lock(A) é libertado
10	COMMIT		
11		W(A)	Dequeue(A), X-lock(A) é permitido
12		UNLOCK(A)	lock(A) é libertado
13		COMMIT	

No instante 8, dado que existe um *lock* do item A, o pedido vai para fila de espera, sendo retirado da fila depois do UNLOCK (A).

Critério de correção:
- (0,5) 2.a) definições
- (0,5) 2.b) execução transações considerando os pedidos em fila de espera

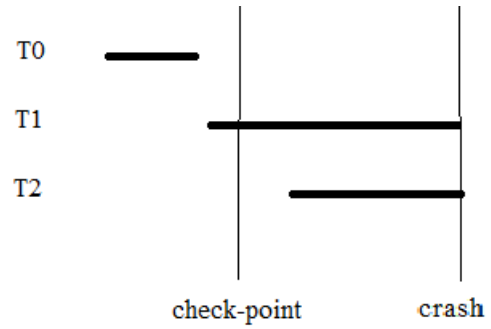
3) Relativo ao Cap. 16- Sistemas de Recuperação

Na recuperação da falha do seguinte registo (*log*) que operações se seguem? Explique detalhadamente os passos de Redo e Undo.

<T_0 start>
<T_0, B, 2000, 2050>
<T_0 commit>
<T_1 start>
<T_1, B, 2050, 2100>
<T_1, O_4, operation-begin>
<checkpoint {T_1}>
<T_1, C, 700, 400>
<T_1, O_4, operation-end, (C, +300)>
<T_2 start>
<T_2, O_5, operation-begin>
<T_2, C, 400, 300>

Resposta:

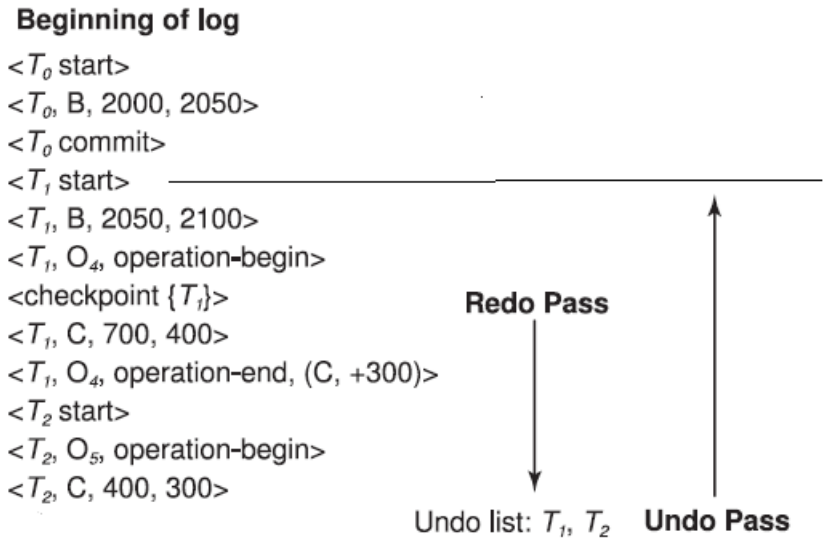
Na recuperação aplicam-se a seguintes regras às transações ativas depois do último *check-point*:
- para todas as transações Tk que não têm registo de <Tk commit> no "log", é executado undo(Tk);
- para todas as transações Tk que têm registo de <Tk commit> no "log", é executado redo(Tk).



T0 terminou antes do *check-point*, tendo toda a informação sido gravada em disco, pelo que nada é preciso fazer.

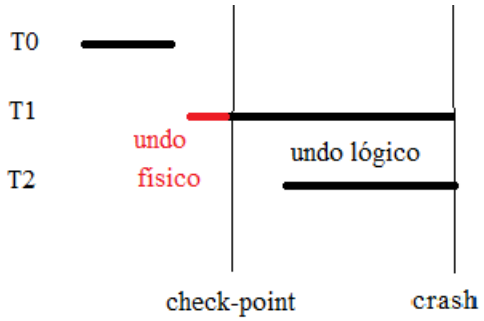
Para as transações T1 e T2 com foram interrompidas pelo *crash*, é necessário proceder ao undo(Tk).

Assim teremos na fase de recuperação da falha:



Fim do Log, Crash
Fase de recuperação são acrescentados os seguintes registos para *undo*(T1) e *undo*(T2)

<T2, C, 400>
<T2 abort>
<T1, C, 400, 700> undo lógico, soma 300 a C
<T1, O4, operation-abort>
<T1, B, 2050> undo físico na DB, repor o valor antes do *chek-point*
<T1, abort>



Para a transação T1 as operações realizadas antes do *check-point*, gravadas em disco, devem set desfeitas fisicamente; as operações depois do *check-point* são desfeitas a o nível lógico.

Critério de correção:
- (0,5) explicação geral
- (0,5) explicação detalhada diferenciando o undo lógico do físico