Lista de Exercícios

MC536 – Bancos de Dados: Teoria e Prática Instituto de Computação Universidade Estadual de Campinas		Transações, Concorrência e Stored Procedures 2012 André Santanchè

Considere a seguinte stored procedure:

```
DELIMITER |
CREATE PROCEDURE executaTransferencia(
    transferId VARCHAR(30),  valor FLOAT,
    contaOrigem VARCHAR(15), contaDestino VARCHAR(15))
BEGIN
    UPDATE ContaCorrente
           SET Saldo = Saldo - valor
           WHERE contaid = contaOrigem;
    UPDATE ContaCorrente
           SET Saldo = Saldo + valor
           WHERE contaid = contaDestino;
    INSERT INTO Transferencia
           VALUES (transferId, contaOrigem, contaDestino, valor);
END|
```

Questão 1

Considere que as seguintes duas chamadas estão sendo feitas em paralelo a esta stored procedure, em que cada uma está associada a uma transação:

```
CALL executaTransferencia('1122', 50, '12345', '54321');

CALL executaTransferencia('7070', 30, '54321', '12345');
```

Considerando que cada tupla da tabela é um item de dados a ser controlado independentemente:

a) Escreva um plano de execução serial relacionado a estas duas chamadas.

b) Há algum outro plano execução para estas chamadas que seja serializável? Se existir escreva os possíveis planos usando técnicas de equivalência baseadas em conflito e em visão. Justifique como pode comprovar que são equivalentes.

c) Escreva um possível plano que gere problemas de Isolamento e justifique como isso pode acontecer.

d) Escreva um possível plano que gere deadlock e justifique como isso pode acontecer.

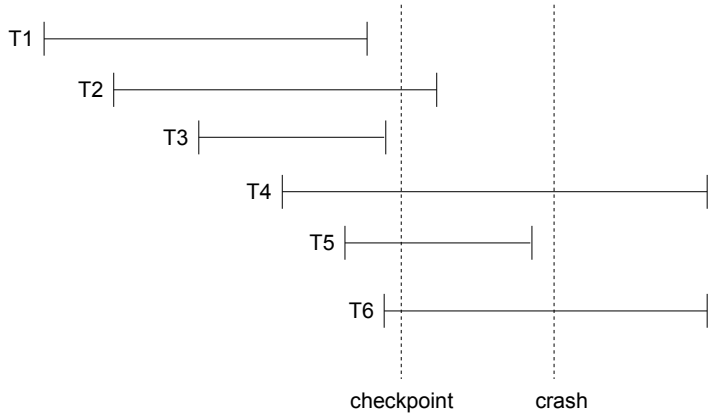
Questão 2

Considere as seguintes chamadas da stored procedure:

```
CALL executaTransferencia('1122', 50, '12345', '54321'); -- T1

CALL executaTransferencia('7070', 30, '54321', '12345'); -- T2

CALL executaTransferencia('1756', 70, '54321', '22222'); -- T3

CALL executaTransferencia('2198', 85, '87654', '54637'); -- T4

CALL executaTransferencia('4563', 90, '17322', '54321'); -- T5

CALL executaTransferencia('5968', 10, '17322', '33333'); -- T6
```

Os saldos das contas antes de sua aplicação são: (12345 → 700); (54321 → 450); (22222 → 1.500); (87654 → 200); (54637 → 820); (17322 → 330); (33333 → 100)

Analise o diagrama abaixo representando a execução concorrente de transações e responda (cada transação está associada a uma stored procedure indicada pelo comentário):



a) Qual será o saldo das contas após a restauração do banco.

As operações podem ter sido executadas em ordem diferente, mas no caso específico destas operações de transferência, no final de tudo não importará a ordem para obtermos os mesmos resultados. Se não houvesse crash, os resultados seriam:

Conta	Antes	Operações	Depois
12345	700	(T1,'1122',50->,650);(T2,'7070',30<-,680)	680
54321	450	(T1,'1122',50<-,500);(T2,'7070',30->,470); (T3,'1756',70->,400);(T5,'4563',90<-,490)	490
22222	1500	(T3,'1756',70<-,1570)	1570
87654	200	(T4,'2198',85->,115)	115
54637	820	(T4,'2198',85<-,905)	905
17322	330	(T5,'4563',90->,240);(T6,'5968',10->,230)	230
33333	100	(T6,'5968',10<-,110)	110

Se houver crash e for possível recuperar as transações, os efeitos de T1, T2, T3 e T5 têm que se tornar permanentes, já que sofreram COMMIT antes do crash e os efeitos de T4 e T6 têm que ser desfeitos. A planilha a seguir mostra em vermelho as operações que teriam que ser desfeitas e qual seria o saldo das contas após a restauração do banco.

Conta	Antes	Operações	Depois
12345	700	(T1,'1122',50->,650);(T2,'7070',30<-,680)	680
54321	450	(T1,'1122',50<-,500);(T2,'7070',30->,470); (T3,'1756',70->,400);(T5,'4563',90<-,490)	490
22222	1500	(T3,'1756',70<-,1570)	1570
87654	200	(T4,'2198',85->,115)	200
54637	820	(T4,'2198',85<-,905)	820
17322	330	(T5,'4563',90->,240);(T6,'5968',10->,230)	240
33333	100	(T6,'5968',10<-,110)	100

b) Considerando que este é um plano serializável ele é: Restaurável? Livre de cascata? Estrito? Justifique.

Há várias possibilidades e cenários a serem analisados aqui. A seguir apresento um exemplo de cada.

Para que o plano seja restaurável:

- T realiza commit somente depois que todas as transações cujos valores T leu realizam commit

A seguir uma situação em que este plano não seria restaurável:

(i) Em um plano serial T1 seria completamente executado antes de T2 ou vice-versa. Se a serialização for equivalente a T2 sendo executado antes de T1, então T1 leu valores modificados por T2 e realizou o COMMIT antes de T2, o que o tornaria não restaurável.

Para que o plano seja livre de cascata:

- T só lê valores que foram alterados por transações que já realizaram COMMIT

Neste caso, uma situação em que o plano não seria livre de cascata:

(ii) Se T2 executar a modificação na conta '54321' antes de T3 lê-la, então T3 necessariamente leria dados não commitados de T2 (já que T2 executa o COMMIT depois). Outra possibilidade é se T3 executar a modificação na conta '54321' antes de T2 e T2 ler este valor antes do COMMIT de T3.

Para que o plano seja estrito:

- T só lê e/ou grava valores que foram alterados por transações que já realizaram commit

Então valem as mesmas observações feitas para livre de cascata mas considerando a leitura e a gravação (não apenas a leitura) de quem leu o dado depois de modificado.

c) Se não for um dos planos de (b) que modificações teriam que ser feitas para alcançá-lo?

Em relação aos casos observados na letra (b). Para o plano restaurável:

(i) Em relação a T1 x T2, há duas modificações possíveis para tornar o plano restaurável, caso ele não esteja: ao produzir uma serialização equivalente T1 teria que realizar suas operações sobre as contas 12345 e 54321 antes de T2, caso contrário, T2 teria que realizar COMMIT antes de T1.

Para o plano livre de cascata:

(ii) Em relação a T2xT3, T3 teria que executar a modificação na conta '54321' antes de T2 e T2 só leria os dados desta conta depois que T3 executasse o COMMIT.

Para que o plano se torne estrito vale a mesma observação, já que a gravação de T2 só acontece depois da leitura.

d) Quais os valores após a restauração do banco nos planos que você fez em (c).