Projeto de Banco de Dados

Transações

PROF. DR. THIAGO ELIAS

Introdução

- Um conjunto de várias operações no banco de dados deve ser uma única unidade do ponto de vista do usuário, apesar de envolver várias operações.
  - Ex: Transferência de valores numa instituição financeira.
- Um sistema de banco de dados precisa garantir a execução apropriada de uma transação, caso ocorra uma falha durante a sua execução.

Conceitos

- Uma transação é uma unidade de execução de programa que acessa e, possivelmente, atualiza vários itens de dados.
- Ela é delimitada por declarações da forma *begin transaction* e *end transaction*.

Conceitos

- Para assegurar a integridade dos dados, exige-se que o sistema de banco de dados mantenha as seguintes propriedades das transações:
  - **A**tomicidade: ou todas as operações da transação são refletidas corretamente no BD ou nenhuma o será.
  - **C**onsistência: A execução de uma transação isolada preserva a consistência do BD.
  - **I**solamento: Embora várias transações possam ocorrer concorrentemente, o sistema deve gerar o mesmo efeito que se as mesmas fossem executadas sequencialmente.
  - **D**urabilidade: Depois da transação completar-se com sucesso, as mudanças feitas no BD persistem, mesmo havendo falha posteriormente.

Conceitos

- Para um exemplo, considere que o banco reside no disco, mas alguma parte dele reside, temporariamente, na memória principal.
- O acesso ao BD é obtido pelas seguintes operações:
  - Read(X) – transfere o item de dado X do disco para a MP
  - Write(X) – atualiza o item de dado no BD no disco.
- Considere o exemplo que transfere x reais da conta A para a conta B.

Conceitos

- Agora vamos considerar cada uma das propriedades ACID:
  - Consistência
    - Significa que a soma de A e B deve permanecer inalterada após a execução da transação (“dinheiro não pode desaparecer”)
    - Isso é responsabilidade do desenvolvedor
  - Atomicidade
    - Caso aconteça uma falha entre o decremento da conta A e o incremento da conta B, entraríamos num estado inconsistente.
    - Qual a ação a ser tomada? Reexecutar ou desfazer a transação?
    - A atomicidade é responsabilidade do sistema de BD, através do *componente de gerenciamento de transações*.

Conceitos

- Durabilidade
  - Após o usuário ser notificado da transferência de fundos, o estado do banco persistirá, mesmo havendo falhas.
  - Suponha que houve uma falha que gerou a perda de dados da MP. Podemos garantir a durabilidade se:
    - As atualizações realizadas pela transação foram gravadas no BD antes da transação completar-se.
    - Informações gravadas no disco sobre as atualizações são suficientes para reconstruir o BD (atualizado) no caso de uma falha.
  - A durabilidade é responsabilidade do sistema de BD, através do *componente de gerenciamento de recuperação*.

Conceitos

- Isolamento
  - Mesmo assegurando as demais propriedade, no caso de execuções concorrentes, duas instruções podem ser executadas intercaladas de forma a gerar um estado inconsistente no BD.
    - Exemplo de duas transações realizando o saque numa mesma conta corrente.
  - É de responsabilidade do sistema de BD manter o isolamento através do *componente de controle de concorrência*.

Estado da Transação

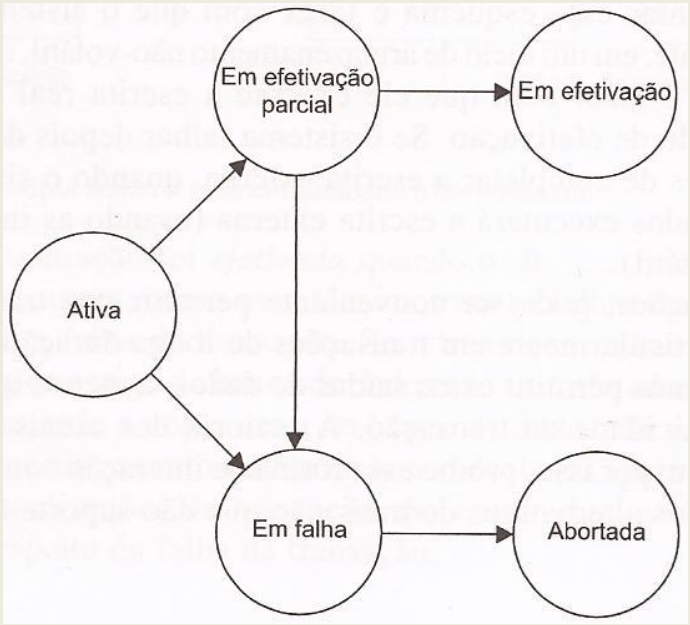
- Na ausência de falhas, todas as transações completam-se com sucesso.
- Porém, caso ocorra uma falha, a transação seria abortada.
- Para manter a propriedade de Atomicidade, caso uma transação seja abortada, as ações realizadas por ela no BD até então devem ser desfeitas (*rolled back*).
- Já uma transação completada (efetivada ou *committed*) deve manter o estado alterado do BD, mesmo ocorrendo uma falha.

Estado da Transação

- Resumindo, as transações devem estar em um dos seguintes estados abstratos:
  - Ativa: estado inicial. Permanece neste estado enquanto estiver executando.
  - Em efetivação parcial: após a execução da última declaração.
  - Em falha: quando descoberto que a execução normal já não pode se realizar.
  - Abortada: depois que a transação foi desfeita e o BD foi restabelecido ao estado anterior ao início da transação.
  - Em efetivação: após a conclusão com sucesso.

Estado da Transação

- Quando a transação termina sua última declaração, entra no estado de efetivação parcial. Ainda é possível ela ser abortada pois os seu efeitos ainda podem estar na MP, e pode haver uma falha.
- O sistema então grava informações da transação no disco. Assim, mesmo havendo uma falha, será possível recuperar os efeitos da transação.



A transação entra no estado de falha quando é determinado que a transação não pode mais concluir a sua execução e então entra no estado de abortado. Há duas soluções para o problema:

Reinicia a transação

Mata a transação

Implementação de Atomicidade e Durabilidade

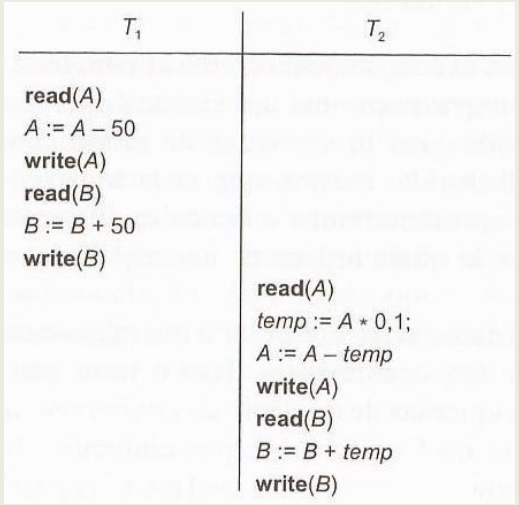
- Consideremos um esquema de BD simples onde cada transação é executada por vez, e que se baseia em cópias do BD (shadows).
- O esquema pressupõe que o BD é um arquivo do disco e que é apontado por um ponteiro chamado *db_pointer* mantido no disco.
- A transação que deseja atualizar o BD, primeiro cria uma cópia dele. Todas as atualizações são feitas na nova cópia.
- Em caso de falha, o BD original (shadow) está intacto.
- Para efetivação da transação, primeiro as páginas do BD são armazenadas no disco e só depois o *db_pointer* é atualizado.
- Em caso de falhas, como se comportaria o BD?
- Essa técnica é boa ou ruim? (concorrência e custo computacional)

Execuções Concorrentes

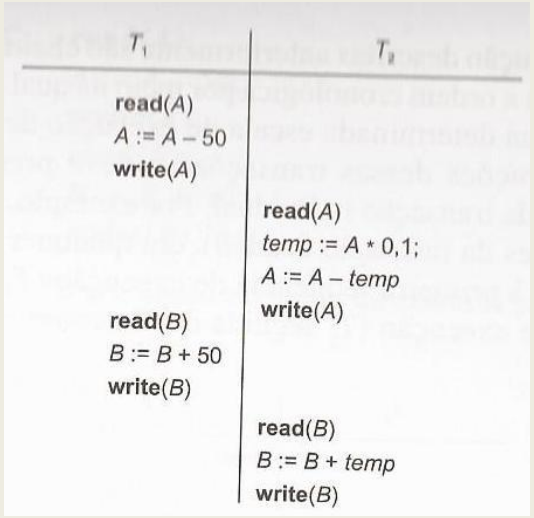
- É normal que sistemas de processamento de transações permita a execução concorrente.
- Por que implementar a concorrência?
  - Podem existir transações com muitas atividades de I/O
  - Podem existir transações longas que prejudiquem as transações curtas.
- Onde mora o perigo da concorrência, considerando que as transações ocorrem, individualmente, com correção? (consistência)

Execuções Concorrentes

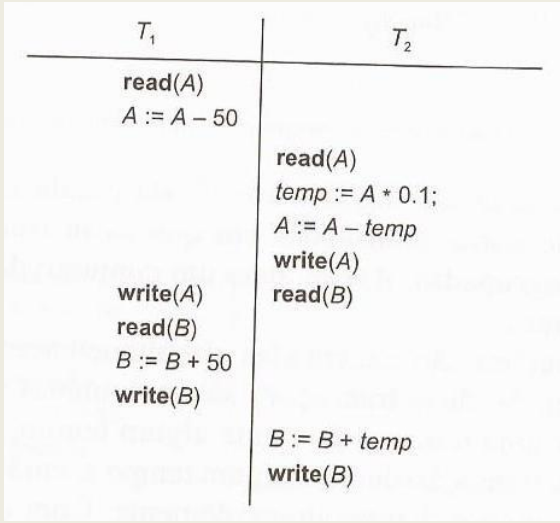
- Exemplos de duas transações transferindo fundos de uma instituição financeira (T1 transfere R$50 de A e T2 transfere 10% de A) .



T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$temp := A * 0,1;$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)



T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
	read(A)
	$temp := A * 0,1;$
	$A := A - temp$
read(B)	write(A)
$B := B + 50$	
write(B)	
	read(B)
	$B := B + temp$
	write(B)



T_1	T_2
read(A)	
$A := A - 50$	
	read(A)
	$temp := A * 0.1;$
	$A := A - temp$
write(A)	write(A)
read(B)	read(B)
$B := B + 50$	
write(B)	
	$B := B + temp$
	write(B)

Serialização

- Antes de entendermos como o BD implementa protocolos para a execução concorrente de transações, vamos entender quais *esquemas de execução* garantem a consistência do BD.
- Consideremos apenas as operações significativas sobre um item de dado: read (leitura) e write (escrita)
  - Exemplos anteriores considerando apenas as instruções de leitura e escrita no BD.
- Assim, estudaremos formas de equivalência entre escalas de execução (*Serialização de conflito* e *visão serializada*)

Serialização de Conflito

- Quando duas transações manipulam itens de dados diferentes, nada precisa ser feito.
- Quando os mesmos itens são manipulados, há 4 casos a considerar:
  - I1 = Read e I2 = Read - A sequência de execução não importa.
  - I1 = Read e I2 = Write - A sequência de execução importa.
  - I1 = Write e I2 = Read - Semelhante a anterior.
  - I1 = Write e I2 = Write - Uma transação não afeta a outra, porém o estado final do BD é afetado. Assim, a sequência de execução também importa.
- Exemplos de serialização de conflito (escalas anteriores).
- Quando uma escala de execução é *conflito serializável*?

Serialização de Conflito

- Problema:

T_1	T_5
read(A)	
$A := A - 50$	
write(A)	
	read(B)
	$B := B - 10$
	write(B)
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$A := A + 10$
	write(A)

Visão Serializada

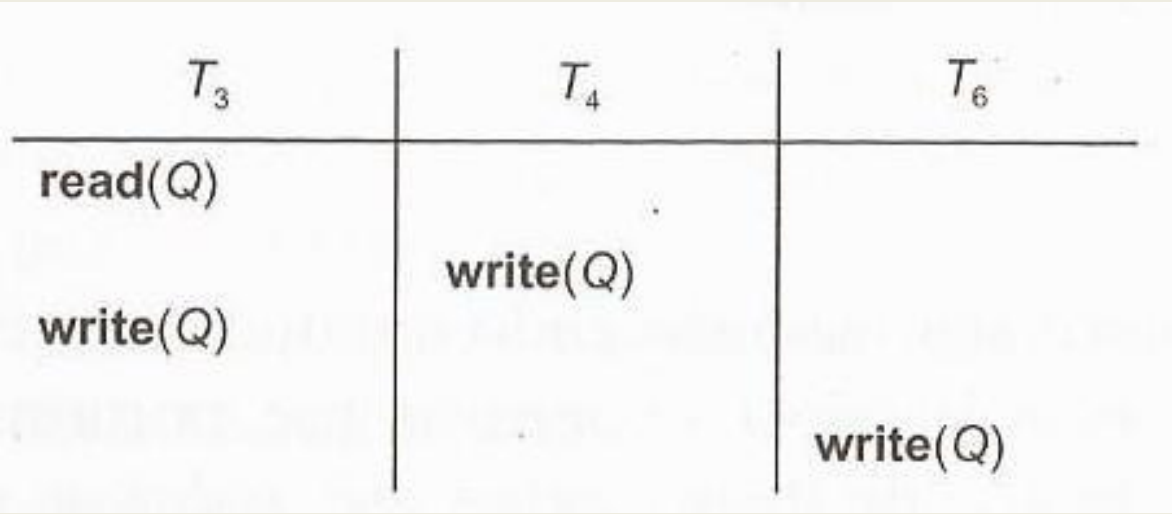
- É uma forma de equivalência menos restrita, mesmo baseando-se também apenas nas instruções leitura e escrita (read e write).

Visão Serializada

- Duas escalas de execução S e S' (por exemplo , com duas transações cada) são equivalentes em visão se:
  - Para cada item de dado Q, se a transação T1 fizer uma leitura do valor inicial de Q na escala S, então a transação T1 também deve, na escala S', ler o valor inicial de Q.
  - Para cada item de dado Q, se a transação T1 executar um read(Q) na escala S, e aquele valor foi produzido pela transação T2, então a transação T1 na escala S' também deve ler um valor alterado por T2.
  - A transação que executa a operação final write (Q) em S deve ser a mesma em S'.
- As condições 1 e 2 asseguram que as transações leem os mesmo valores e a 3 que ambas as escalas produzam os mesmos efeitos no BD.

Visão Serializada

- Exemplo



Visão Serializada

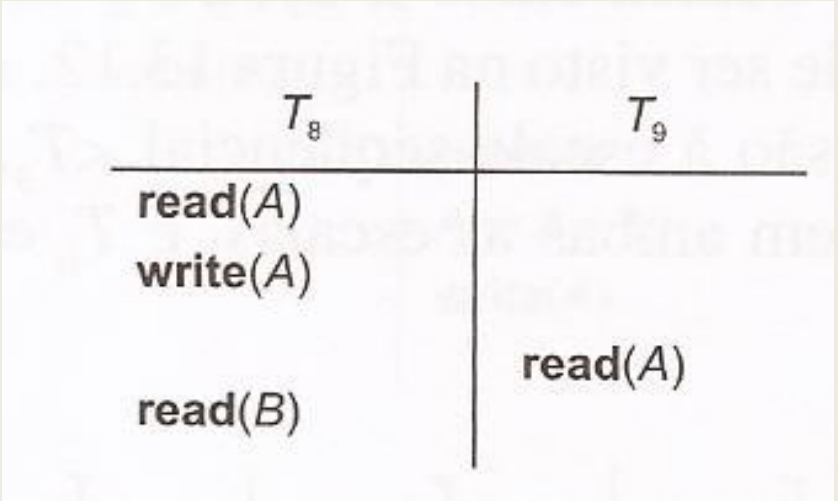
- Concluímos que toda escala conflito serializável é visão serializável, mas há escalas visão serializável que não são conflito serializável.

Recuperação

- Até o momento não consideramos falhas durante a execução das transações.
- Se uma transação falhar, é necessário desfazer os seus efeitos para garantirmos a consistência do BD (atomicidade).
- Mas se essa transação estiver sendo executada concorrentemente?
  - É necessário criar mecanismos para permitir apenas escalas que, em caso de falhas, a integridade do BD seja mantida.

Escalas de Execução Recuperáveis

- A escala ao lado é recuperável em caso de falha? (imagine que T9 é efetivada logo após a leitura de A)

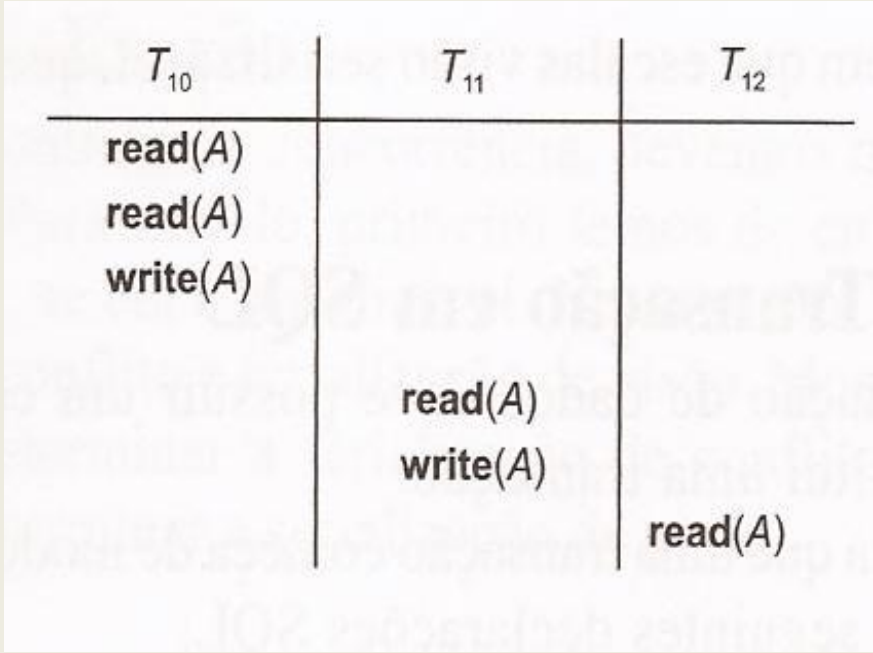


Esse é um exemplo de um a escala de execução não recuperável!!
Ela não poderia ter sido autorizada pelo sistema de BD.

Uma escala recuperável é aquela que, caso T1 leia um valor de T2, esta (T1) só poderá ser efetivada depois de T2.

Escalas sem Cascata

- Considerando a escala ao lado (a efetivação de todas as transações ocorrerá apenas depois de T12), se T10 falhar logo após a execução de T12, todas as transações deverão ser desfeitas.
- O sistema de BD deve evitar escalas que possibilitem o *Retorno em Cascata* (*cascading rollback*)



T_{10}	T_{11}	T_{12}
read(A)		
read(A)		
write(A)		
	read(A)	
	write(A)	
		read(A)

Implementação do Isolamento

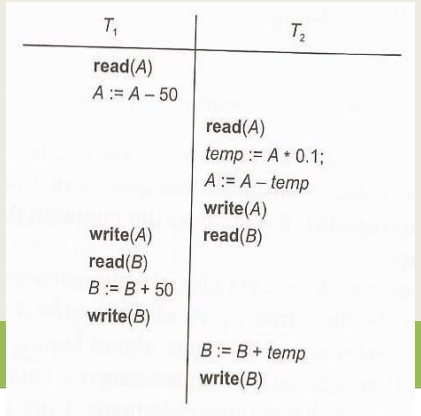
- Como implementar o isolamento de tal forma que sempre tenhamos esquemas de execução que garantam a consistência do BD?
  - E se cada transação bloqueasse todo o banco sempre que fizesse acesso a ele?
- O objetivo de um esquema de controle de concorrência é proporcionar um alto grau de concorrência, enquanto garante que todas as escalas geradas sejam conflito ou visão serializável, e também sejam sem cascata.
- Estudaremos esquemas diferentes que garantam a execução concorrente.

Definição de Transação em SQL

- Começam de modo subentendido.
- Terminam com um das duas declarações:
  - Commit work: efetivação da transação
  - Rollback work aborta a transação

Teste para Serialização de Conflito

- Cria-se um Gráfico de Precedência de S.
  - Composto por vértices (transações) e arestas (precedência entre Ti e Tj)
- O conjunto de arestas Ti → Tj consiste em uma das seguintes condições:
  - Ti executa write(Q) antes de Tj executar read(Q)
  - Ti executa read(Q) antes de Tj executar write(Q)
  - Ti executa write(Q) antes de Tj executar write(Q)



T_1	T_2
read(A)	
$A := A - 50$	
	read(A)
	$temp := A * 0.1$;
	$A := A - temp$
	write(A)
write(A)	read(B)
read(B)	
$B := B + 50$	
write(B)	
	$B := B + temp$
	write(B)