Banco de Dados II

Profª Elyda Laisa

Aula 02 – Processamento de Transações

SLIDES ADAPTADOS DAQUELES DISPONIBILIZADOS PELOS AUTORES ELMASRI E NAVATHE

Introdução

- Na aula de hoje veremos os conceitos básicos e teoria necessários para garantir o funcionamento correto do BD

Sistemas de monousuário *versus* multiusuário

- Um critério para classificar um sistema de banco de dados é de acordo com o número de usuários que podem usar o sistema **simultaneamente**
  - **Um** SGBD pode ser **monousuário - no máximo um usuário de** cada vez pode utilizar o sistema
  - Ou **multiusuário** se muitos usuários puderem acessar o banco de dados simultaneamente.

Sistemas de monousuário *versus* multiusuário (cont.)

- Os SGBDs monousuário são principalmente restritos a sistemas de computador pessoal
  - Você pode me dar um exemplo?
- A maioria dos outros SGBDs é multiusuário
  - Múltiplos usuários podem acessar os bancos de dados simultaneamente devido ao conceito da **multiprogramação**
    - O processador executa vários processos ao mesmo tempo

Sistemas de monousuário *versus* multiusuário (cont.)

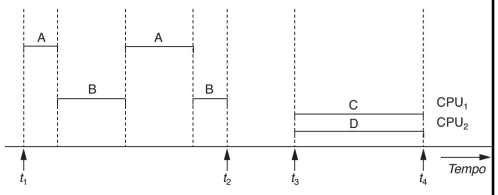


Figura 21.1
Processamento intercalado *versus* processamento paralelo de transações simultâneas.

Transações

- Uma **transação é um programa em execução** que forma uma unidade lógica de processamento de banco de dados
- Ela inclui uma ou mais operações de acesso ao banco de dados
  - Podem incluir operações de inserção, exclusão, modificação ou recuperação

Transações

- Os limites de uma transação são definidos pelas instruções explícitas BEGIN TRANSACTION e END TRANSACTION
- Uma transação pode ser do tipo somente leitura ou leitura e gravação

Transações

(a)

T_1

read_item(X);
$X := X - N$;
write_item(X);
read_item(Y);
$Y := Y + N$;
write_item(Y);

(b)

T_2

read_item(X);
$X := X + M$;
write_item(X);

Figura 21.2
Duas transações de exemplo. (a) Transação T_1. (b) Transação T_2.

Transações

- As operações básicas de acesso ao banco de dados que uma transação pode incluir são as seguintes:
  - **read_item(*X*).** ***Lê um item do banco de dados*** chamado *X para uma variável do programa.* Para simplificar nossa notação, consideramos que *a variável de programa também é chamada X.*
  - **write_item(*X*).** ***Grava o valor da variável*** *X no item de banco de dados chamado X.*

Transações

- A transação de leitura read_item(X) inclui as seguintes etapas:
  - Ache o endereço no bloco do disco que contém o item X
  - Copie esse bloco para um buffer na memória principal
  - Copie o item X do buffer para a variável chamada X

Transações

- A transação de escrita write_item(X) inclui as seguintes etapas:
  - Ache o endereço no bloco do disco que contém o item X
  - Copie esse bloco para um buffer na memória principal
  - Copie o item X da variável chamada X para o buffer
  - Armazene o bloco do buffer no disco

Transações

- Muitas transações podem ocorrer simultaneamente em um SGBD

POR QUE O CONTROLE DE CONCORRÊNCIA É NECESSÁRIO?

Porque o controle de concorrência é necessário

- Vários problemas podem acontecer quando transações simultâneas são executadas de uma maneira descontrolada
- Ilustramos alguns desses problemas em um banco de dados muito simplificado

Porque o controle de concorrência é necessário (cont.)

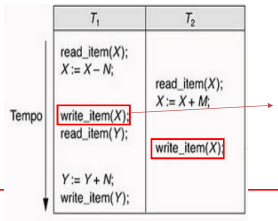
- **O problema da atualização perdida**
- **O problema da atualização temporária (ou leitura suja)**
- **O problema do resumo incorreto**
- **O problema da leitura não repetitiva**

Porque o controle de concorrência é necessário (cont.)

- **Problema da Atualização Perdida**: Ocorre quando duas transações que acessam os mesmos itens do BD têm suas operações intercaladas de modo incorreto

X = 80
N=5
M = 4

Valor final correto = 79
Mas o valor final acaba sendo 84

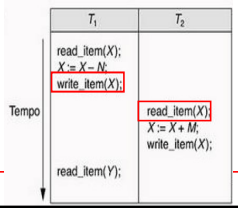


O valor de X resultante de T1 é perdido

17

Porque o controle de concorrência é necessário (cont.)

- **Problema da Atualização Temporária (ou leitura suja):** Ocorre quando uma transação atualiza um item do BD e depois a transação falha. Nesse meio tempo, um item atualizado é lido por outra transação



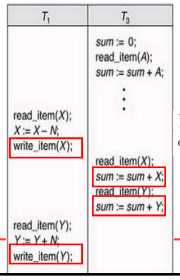
Transação T1 falhou

E T2 leu o valor incorreto

18

Porque o controle de concorrência é necessário (cont.)

- **Problema do Resumo Incorreto:** Se uma transação está calculando uma função de resumo de agregação enquanto outras transações estão atualizando os mesmos itens a função pode retornar um valor incorreto



T_3 lê X depois que N é subtraído e lê Y antes que N seja somado; um resumo errado é o resultado (defasado por N).

19

Porque o controle de concorrência é necessário (cont.)

- **Problema da Leitura não Repetitiva**: Ocorre quando uma transação T lê o mesmo item de dados duas vezes e o item é atualizado por outra transação entre as duas leituras

20

POR QUE A RECUPERAÇÃO É NECESSÁRIA?

Por que a recuperação é necessária

- Sempre que uma transação é submetida a um SGBD para execução, o sistema é responsável:
  - Por garantir que todas as operações na transação sejam concluídas com sucesso e seu efeito seja registrado permanentemente no banco de dados
  - Ou que a transação não tenha qualquer efeito no banco de dados ou quaisquer outras transações.

Por que a recuperação é necessária (cont.)

Existem vários motivos possíveis para uma transação falhar no meio da execução:

1. **Uma falha do computador (falha do sistema)**
2. **Um erro de transação ou do sistema**
   - Estouro de inteiro, divisão por 0
3. **Erros locais ou condições de exceção detectadas pela transação**
4. **Imposição de controle de concorrência**
5. **Falha de disco**
6. **Problemas físicos e catástrofes**

Por que a recuperação é necessária (cont.)

Sempre que ocorrerem falhas do tipo 1 ao tipo 4 o sistema precisa manter informações para que possa se recuperar da falha

UM POUCO MAIS SOBRE AS TRANSAÇÕES

Conceitos de transação e sistema

- **Estados de transação e operações adicionais**
  - Uma transação é uma unidade atômica de trabalho, que deve ser concluída totalmente ou não ser feita de forma alguma
- Para fins de recuperação, o sistema precisa registrar quando cada transação começa, termina e confirma ou aborta
  - Então o SGBD precisa acompanhar as seguintes operações

Conceitos de transação e sistema (cont.)

- BEGIN_TRANSACTION. Esta marca o início da execução da transação.
- READ ou WRITE. Estas especificam operações de leitura ou gravação nos itens do banco de dados que são executados como parte de uma transação.
- END_TRANSACTION. Esta especifica que operações de transação READ e WRITE terminaram e marca o final da execução da transa-

Conceitos de transação e sistema (cont.)

- COMMIT_TRANSACTION. Esta sinaliza um *final bem-sucedido* da transação, de modo que quaisquer mudanças (atualizações) executadas pela transação podem ser seguramente **confirmadas** (committed) ao banco de dados e não serão desfeitas.
- ROLLBACK (ou ABORT). Esta operação sinaliza que a transação *foi encerrada sem sucesso*, de modo que quaisquer mudanças ou efeitos que a transação possa ter aplicado ao banco de dados precisam ser **desfeitos**.

Conceitos de transação e sistema

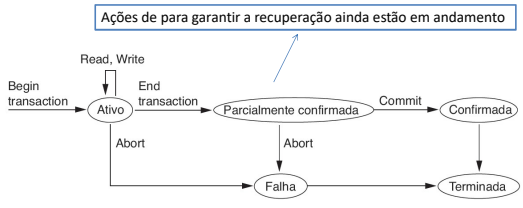


Figura 21.4
Diagrama de transição de estado ilustrando os estados para execução da transação.

Conceitos de transação e sistema

- Para poder recuperar-se de uma falha o SGBD mantém um log
  - Ele não deve ser afetado por nenhum tipo de falha (exceto as de disco ou catastróficas)
  - Ele alcança isto refazendo ou desfazendo transações
- O ponto de confirmação (COMMIT) indica que as alterações realizadas pelas transações devem ser gravadas permanentemente

Propriedades desejáveis das transações

- Uma das funções mais importantes do SGBD é garantir a execução correta das transações
  - Evita os problemas descritos anteriormente
  - Para isso, o SGBD deve garantir as propriedades ACID das transações
- As propriedades ACID são importantes para manter a corretude dos dados

31

Propriedades desejáveis das transações

- Elas devem ser impostas pelos métodos de controle de concorrência e recuperação do SGBD. A seguir estão as propriedades ACID:
  - **Atomicidade**
  - **Preservação da consistência**
  - **Isolamento**
  - **Durabilidade ou permanência**

SCHEDULES

Introdução

- Quando as transações estão executando simultaneamente em um padrão intercalado a ordem da execução das operações de todas as diversas transações é conhecida como um **schedule** (ou **histórico**)

Schedules de Transações

- Um **schedule *S de n transações*** *T1, T2, ..., Tn é uma ordenação das operações das transações*
  - As operações das diferentes transações podem ser intercaladas no schedule *S*
- No entanto, para cada transação T_i que participa do Schedule S, suas operações precisam aparecer na mesma ordem em que ocorrem em T_i
  - Consideremos, por ora, o schedule como organizado em ordenação total

Schedules de Transações

- As ações em um schedule podem ser:
  - begin_transaction b
  - read_item r
  - write_item w
  - end_transaction e
  - commit c
  - abort a
- O valor subscrito é o número da transação

$$S_a: r_1(X); r_2(X); w_1(X); r_1(Y); w_2(X); w_1(Y);$$

$$S_b: r_1(X); w_1(X); r_2(X); w_2(X); r_1(Y); a_1;$$

Schedules de Transações

- Duas operações estão entrando conflito em um schedule se:
  1. Elas pertencerem a diferentes transações
  2. Elas acessam o mesmo item X
  3. Pelo menos uma das operações é um write_item(X)
- Quem está entrando em conflito?

S_a: $r_1(X)$; $r_2(X)$; $w_1(X)$; $r_1(Y)$; $w_2(X)$; $w_1(Y)$;

Schedules de Transações

- Duas operações também estão em conflito se a mudança de ordem causar um resultado diferente
  - Por exemplo, se invertermos a ordem de $r_1(X)$ e $W_2(X)$ o valor de X lido por T1 muda
  - Isso porque o valor de X é mudado por $W_2(X)$ antes que seja lido por $r_1(X)$

S_a: $r_1(X)$; $r_2(X)$; $w_1(X)$; $r_1(Y)$; $w_2(X)$; $w_1(Y)$;

- Esse conflito é chamado **leitura-gravação**

Schedules de Transações

- Um schedule é considerado completo se:
  1. As operações em S são exatamente aquelas operações em T_1, T_2, ..., T_n, incluindo uma operação de confirmação ou cancelamento como última operação em cada transação no schedule.
  2. Para qualquer par de operações da mesma transação T_i, sua ordem de aparecimento relativa em S é a mesma que sua ordem de aparecimento em T_i.
  3. Para duas operações quaisquer em conflito, uma das duas precisa ocorrer antes da outra no schedule.[10]

[10]: Se não houver conflito não é preciso determinar ordem (operações podem ocorrer livremente, respeitando o item 2), resultando em um **schedule parcial**

Caracterizando schedules com base na facilidade de serialização

- Para uma transação confirmada nunca deve ser necessário cancelá-la
  - Os schedules que atendem essa condição são **recuperáveis**
  - Do contrário, são **não recuperáveis**

Caracterizando schedules com base na facilidade de serialização

- Os schedules sem cascata não permitem que uma transação leia itens de transações não confirmadas
  - Todos os schedules sem cascata são recuperáveis

Caracterizando schedules com base na facilidade de serialização

- Agora, caracterizamos os tipos de schedules que são sempre considerados *corretos* quando transações concorrentes estão sendo executadas

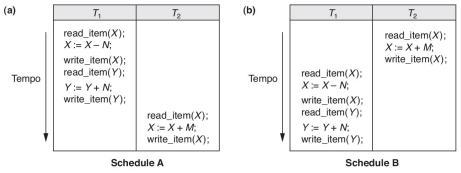
Caracterizando schedules com base na facilidade de serialização

- Se nenhuma intercalação de operações for permitida, existem apenas dois resultados possíveis:

1. Executar todas as operações da transação T_1 (em sequência) seguidas por todas as operações da transação T_2 (em sequência).
2. Executar todas as operações da transação T_2 (em sequência) seguidas por todas as operações da transação T_1 (em sequência).

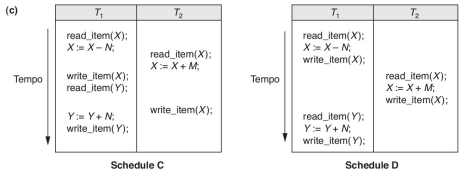
Schedules seriais, não seriais e serializáveis por conflito

- Os schedules A e B da Figura 21.5(a) e (b) são chamados de *seriais*



Schedules seriais, não seriais e serializáveis por conflito

- Os schedules C e D da Figura 21.5(c) são chamados de *não seriais,* pois cada sequência intercala operações das duas transações.



Schedules seriais, não seriais e serializáveis por conflito

- Os schedules seriais limitam a concorrência
- Desejamos ter schedules que produzem resultado correto e são intercaláveis

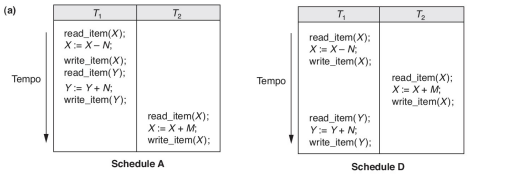
Schedules seriais, não seriais e serializáveis por conflito

- Um schedule é **serializável** se for equivalente a algum schedule serial das mesmas transações
  - Devem produzir o mesmo resultado
  - As operações devem estar na mesma ordem nos dois schedules

Schedules seriais, não seriais e serializáveis por conflito

- Os schedules podem ser
  - **Equivalentes em conflito**: Se a ordem das operações em conflito for a mesma nos dois schedules

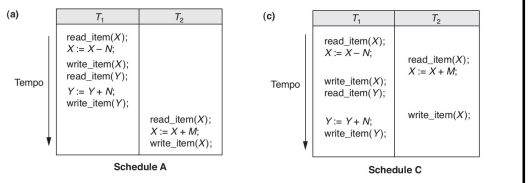
Serializável



Schedules seriais, não seriais e serializáveis por conflito

- Os schedules podem ser (cont.)
  - **Equivalentes em conflito**: Se a ordem das operações em conflito for a mesma nos dois schedules

Não serializável

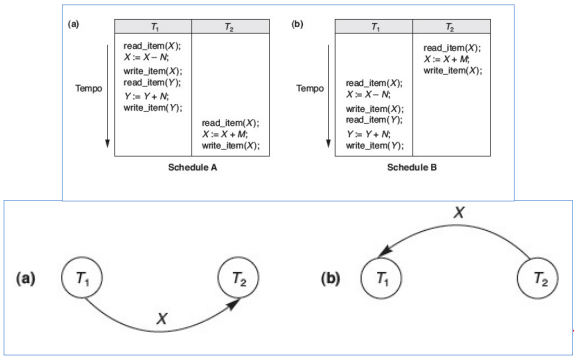


Schedules seriais, não seriais e serializáveis por conflito

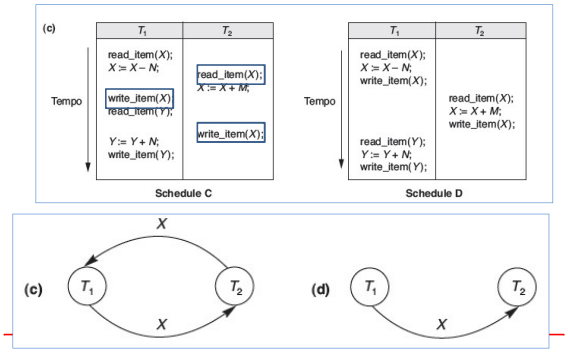
- Testando a serialização de conflito

1. Para cada transação T_i participante no schedule S, crie um nó rotulado com T_i no grafo de precedência.
2. Para cada caso em S onde T_j executa um read_item(X) depois de T_i executar um write_item(X), crie uma aresta ($T_i \rightarrow T_j$) no grafo de precedência.
3. Para cada caso em S onde T_j executa um write_item(X) após T_i executar um read_item(X), crie uma aresta ($T_i \rightarrow T_j$) no grafo de precedência.
4. Para cada caso em S onde T_j executa um write_item(X) após T_i executar um write_item(X), crie uma aresta ($T_i \rightarrow T_j$) no grafo de precedência.
5. O schedule S é serializável se, e somente se, o grafo de precedência não tiver ciclos.

Schedules seriais, não seriais e serializáveis por conflito



Schedules seriais, não seriais e serializáveis por conflito



Schedules seriais, não seriais e serializáveis por conflito

- Na prática isso é muito difícil de se prever
- E deixar as operações ocorrendo naturalmente pode levar à necessidade de muitos rollbacks, o que torna a técnica quase impraticável
- Na prática, os SGBD comerciais utilizam protocolos (isto é, um conjunto de regras) que, se seguido levam a schedules serializáveis

Schedules seriais, não seriais e serializáveis por conflito

- Veremos isto na próxima aula quando estudaremos o controle de concorrência

Referência

- Elmasri e Navathe. Sistemas de Banco de Dados. Pearson, 2011.