Apresentação baseada em:

- Ramez Elmasri e Shamkant B. Navathe 6° Ed (2010)
- Material da Profa. Dra. Sarajane Marques Peres (UNIVESP)

Banco de Dados

Introdução aos conceitos de Processamento de Transação (parte 1)

FACOM – UFMS

Vanessa Borges

vanessa.a.borges@ufms.br

Capítulo 21: Introdução aos conceitos e teoria de processamento de transações

Transações

- Um dos papéis de um SGBD é **processar transações**: um conjunto de várias operações que são executadas sobre os dados do banco de dados, e que devem ser vistas pelo usuário do sistema como uma única unidade de processamento

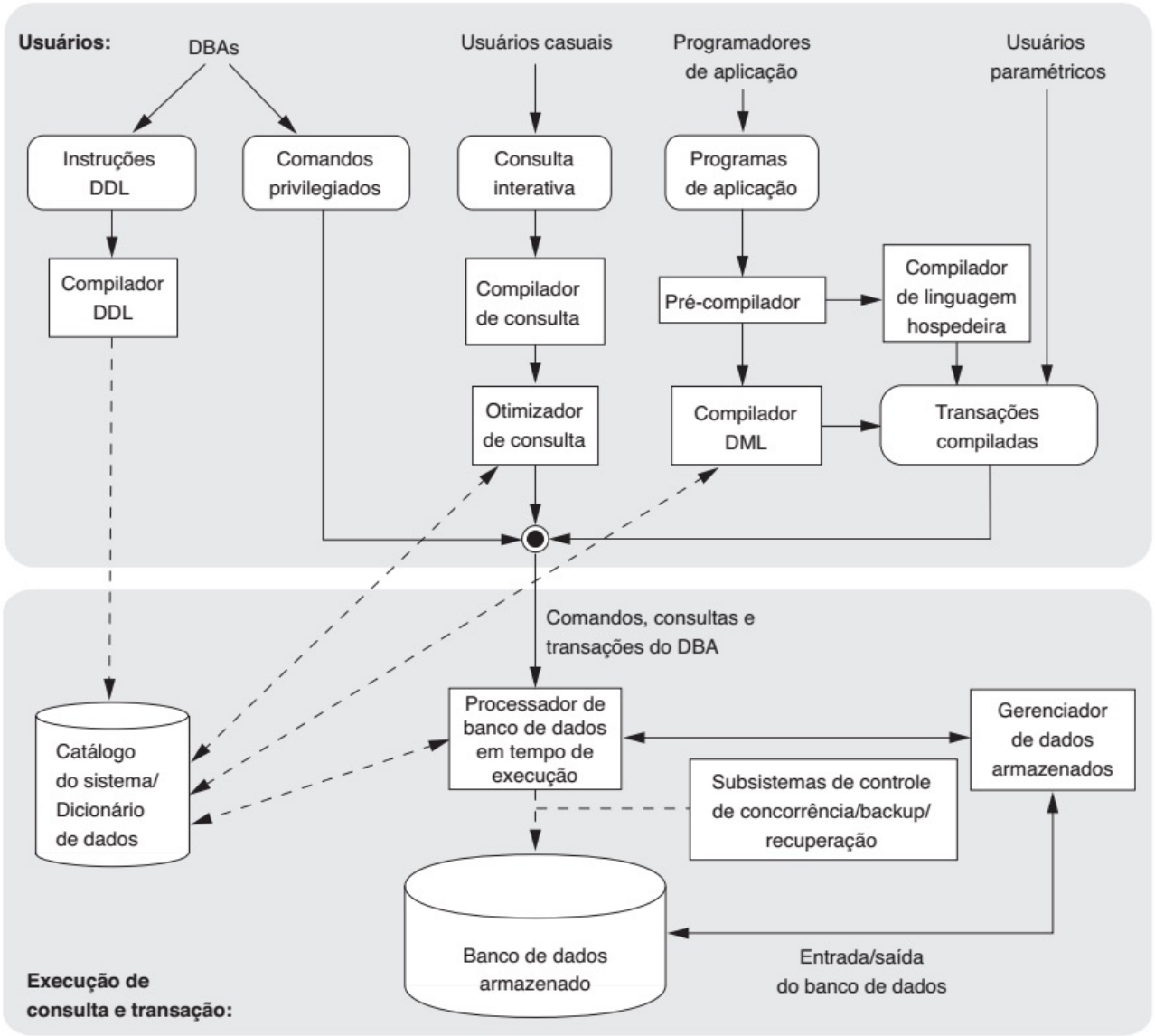
  - **Exemplo:**
    - A transferência de valores entre contas correntes é uma operação única do ponto de vista de um cliente de um sistema bancário, porém, dentro do sistema de banco de dados, essa transferência envolve várias operações.

Transações

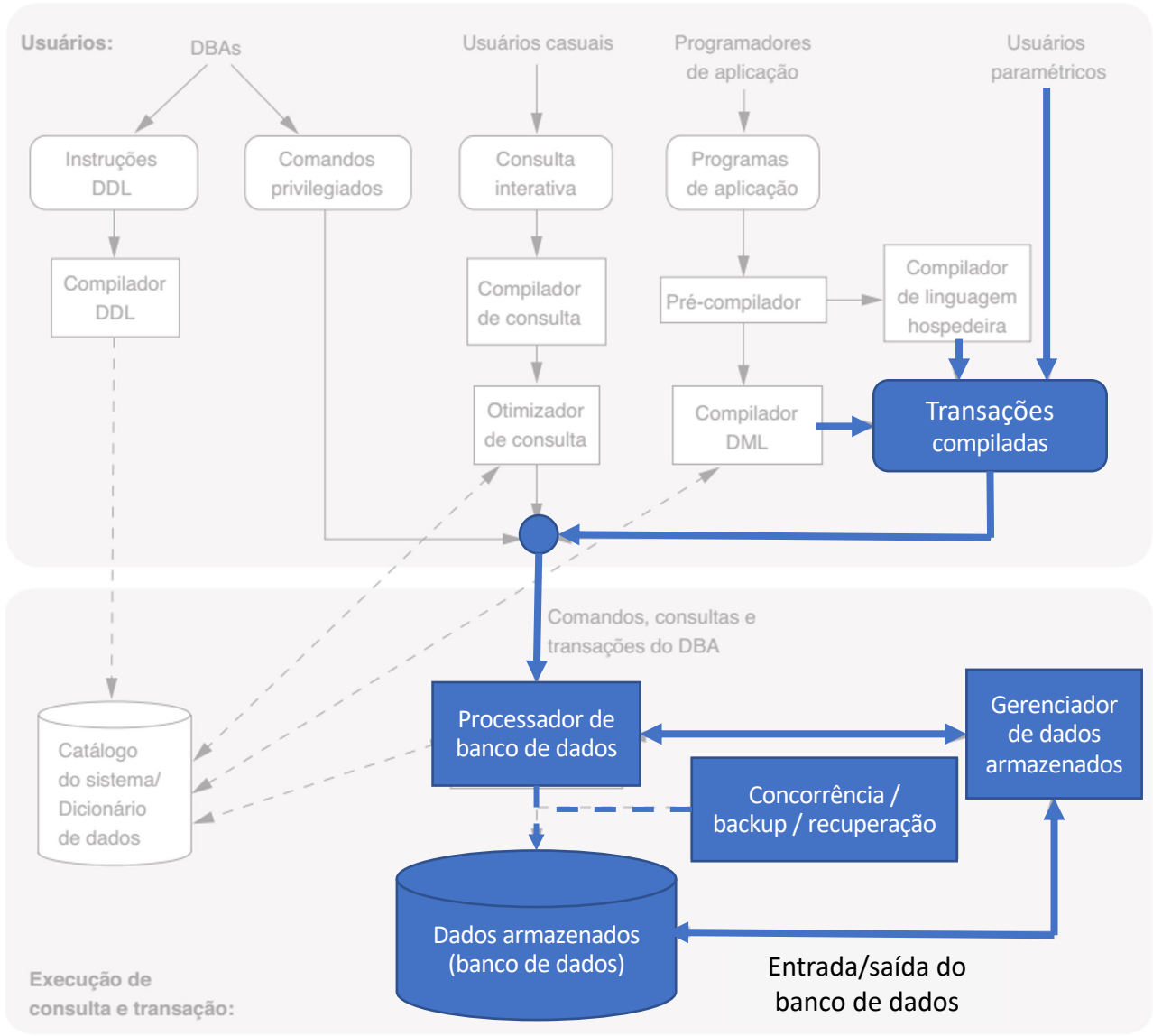
- Uma transação é um programa em execução que forma uma unidade lógica de trabalho
  - É **essencial que todo o conjunto de operações de transação seja concluído**, ou que, no caso de um problema (falha), nenhuma das operações do conjunto tenha efeito sobre os dados

- O processamento das transações, realizado pelo SGBD deve garantir que:
  - A execução de uma transação seja completa
  - Seja possível executar várias transações de forma simultânea, sem gerar inconsistência nos dados

Processamento de Transações
Componentes do SGBD



Processamento de Transações
Componentes do SGBD



Operações básicas de uma transação

- As operações básicas de acesso ao banco de dados que uma transação pode incluir são as seguintes:

  - **read_item(*X*):** transfere o item de dados ***X*** do BD para um *buffer* local alocado à transação que executou a operação read_item. O valor de X é colocado dentro de uma variável de programa.

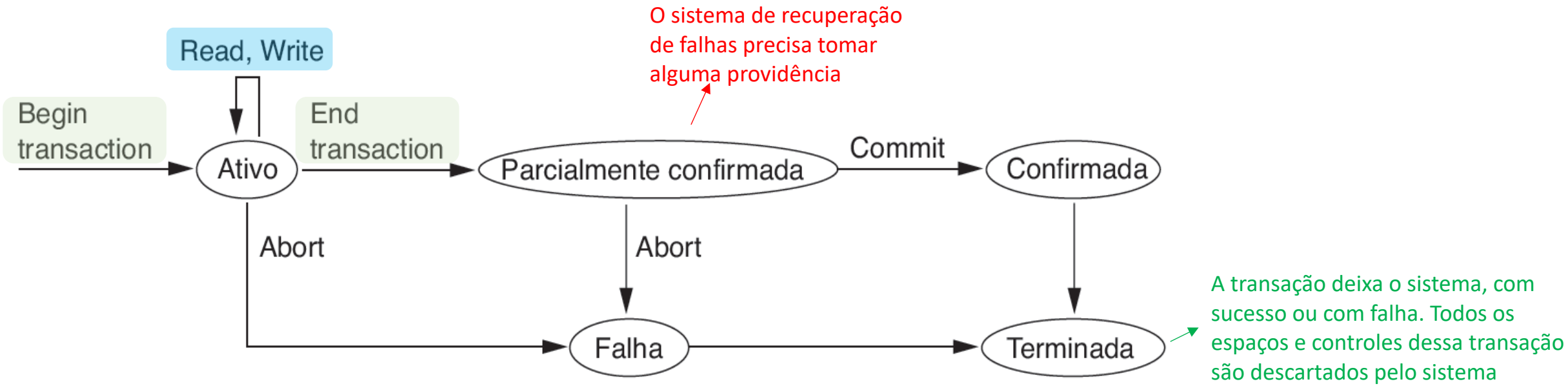
  - **write_item(*X*)**: transfere o item de dados ***X*** do *buffer* local da transação que executou o write_item de volta para o BD. O valor da variável de programa é passado para o item de dado no BD (ainda no buffer – a gravação efetiva pode não ser imediata)

Estados de execução de uma transação

Uma transação **começa** quando for executada a 1ª instrução SQL executável e **termina** com um dos seguintes eventos:

- Comando *COMMIT* ou *ROLLBACK* é emitido;
  - Instrução DDL ou DCL é executada (*commit* automático);
  - O usuário desconecta do banco de dados (*commit* automático);
- O sistema falha (*rollback* automático).

Quando uma transação termina, o próximo comando SQL inicia automaticamente a próxima transação.



Propriedade das transações - ACID

- As transações devem possuir propriedades denominadas **ACID**
- Essas propriedades devem ser impostas pelos métodos de controle de concorrência e recuperação do SGBD.

  - **Atomicidade**
  - **Consistência**
  - **Isolamento**
  - **Durabilidade ou permanência**

Propriedade das transações - ACID

- **Atomicidade**
  - Uma transação é uma unidade indivisível (atômica)
  - Garante que todos os efeitos de uma transação em um banco de dados ou nenhum deles é aceito
  - Uma falha não pode deixar o banco de dados em um estado em que a transação é parcialmente executada

- Unidade atômica
- Executa em sua totalidade (nunca parcialmente)

```
read_item(A);
A:=A-50
write_item(A);
                    Falha!
....
```

Assegurar a **ATOMICIDADE** de uma transação é **responsabilidade do SGBD**, mais especificamente dos componentes de **Gerenciamento de Transações** e de **Recuperação de Falhas**.

Propriedade das transações - ACID

- **Consistência**
  - Garante que se o banco de dados for inicialmente consistente, a execução da transação (por si só) deixa o banco de dados em um estado consistente

Assegurar a **CONSISTÊNCIA** de uma transação é **responsabilidade do programador.**

Propriedade das transações - ACID

- **Isolamento**
  - Transações são isoladas umas das outras
  - Garante que transações executadas concorrentemente sejam isoladas umas das outras, de modo que cada uma tenha a impressão de que nenhum outra transação está sendo executada concorrentemente a ela

T1

```
read_item(A);
A:=A-50
write_item(A);
read_item(B);
B:=B+50
write_item(B);
```

T2

```
read_item(A);
A:=A*1,5;
write_item(A);
```

Assegurar o **ISOLAMENTO** de uma transação é **responsabilidade do Controle de Concorrência.**

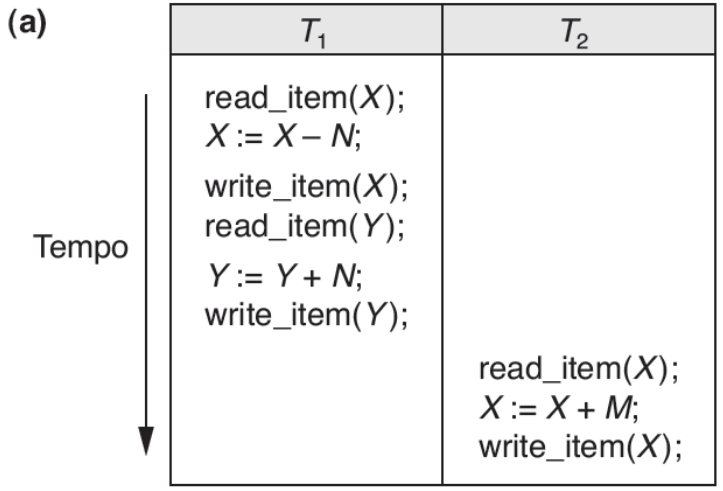
Propriedade das transações - ACID

- **Durabilidade**
  - Garante que quando uma transação tiver sido **confirmada** (***commit***), as atualizações dessa transação **não são perdidas**, mesmo que haja uma falha no sistema
    - As transações finalizadas são gravadas em dispositivos de memória permanente (não-volátil), como discos rígidos, de modo que os dados estejam sempre disponíveis, mesmo que a instância do BD seja reiniciada

É responsabilidade do subsistema de **recuperação a falhas** do SGBD.

Execução concorrente de transações
Schedule: plano de execução

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



Schedule A

Execução concorrente de transações

T1

```
read_item(X);
X:=X-50
write_item(X);
read_item(Y);
Y:=Y+50
write_item(Y);
```

T2

```
read_item(X);
X:=X*1,5;
write_item(X);
```

Valores iniciais

X = 100 Y = 30

T1	T2
read_item(X);	
X:=X-50	
write_item(X);	
read_item(Y);	
Y:=Y+50	
write_item(Y);	
	read_item(X);
	X:=X*1,5;
	write_item(X);

Valores finais

X = 75 Y = 80

Estado consistente do banco de dados após a execução das duas transações

Execução concorrente de transações

T1

```
read_item(X);
X:=X-50
write_item(X);
read_item(Y);
Y:=Y+50
write_item(Y);
```

T2

```
read_item(X);
X:=X*1,5;
write_item(X);
```

Valores iniciais

X = 100 Y = 30

T1	T2
	read_item(X);
	X:=X*1,5;
	write_item(X);
read_item(X);	
X:=X-50	
write_item(X);	
read_item(Y);	
Y:=Y+50	
write_item(Y);	

Valores finais

X = 100 Y = 80

Estado consistente do banco de dados após a execução das duas transações

Execução concorrente de transações

T1

```
read_item(X);
X:=X-50
write_item(X);
read_item(Y);
Y:=Y+50
write_item(Y);
```

T2

```
read_item(X);
X:=X*1,5;
write_item(X);
```

Valores iniciais

X = 100 Y = 30

T1	T2
read_item(X);	
X:=X-50	
	read_item(X);
	X:=X*1,5;
	write_item(X);
write_item(X);	
read_item(Y);	
Y:=Y+50	
write_item(Y);	

X=150*1,5=150

Valores finais

X = 150 Y = 80

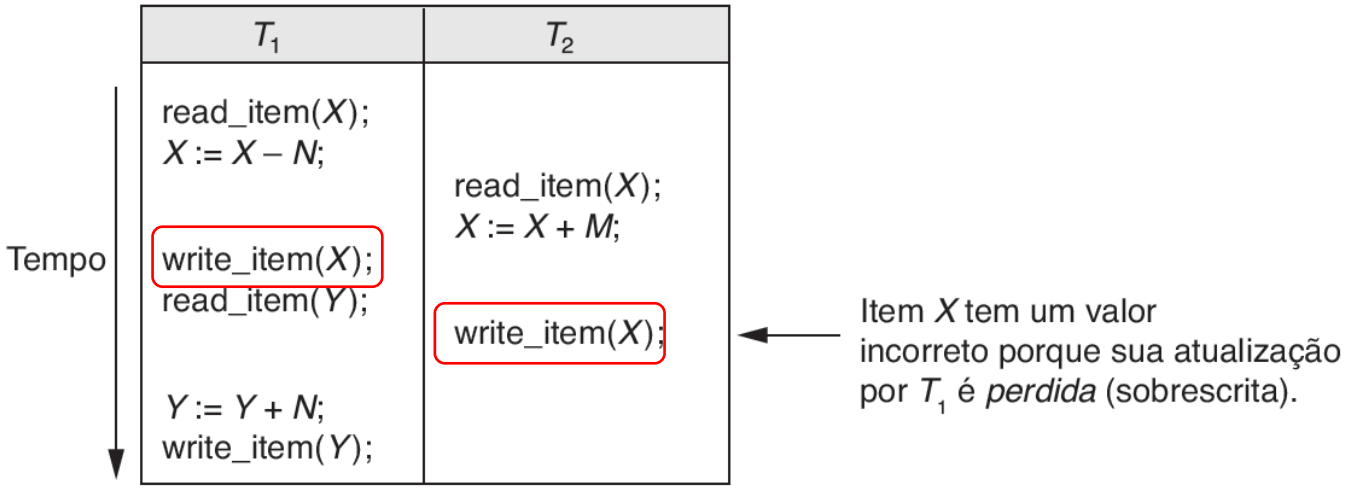
Resultado inconsistente

Por que o controle de concorrência é necessário?

- Vários problemas podem acontecer quando transações simultâneas são executadas de uma maneira descontrolada
  - O problema da **atualização perdida**
  - O problema da **atualização temporária** (ou leitura suja)
  - O problema do **resumo incorreto**
  - O problema da **leitura não repetitiva**

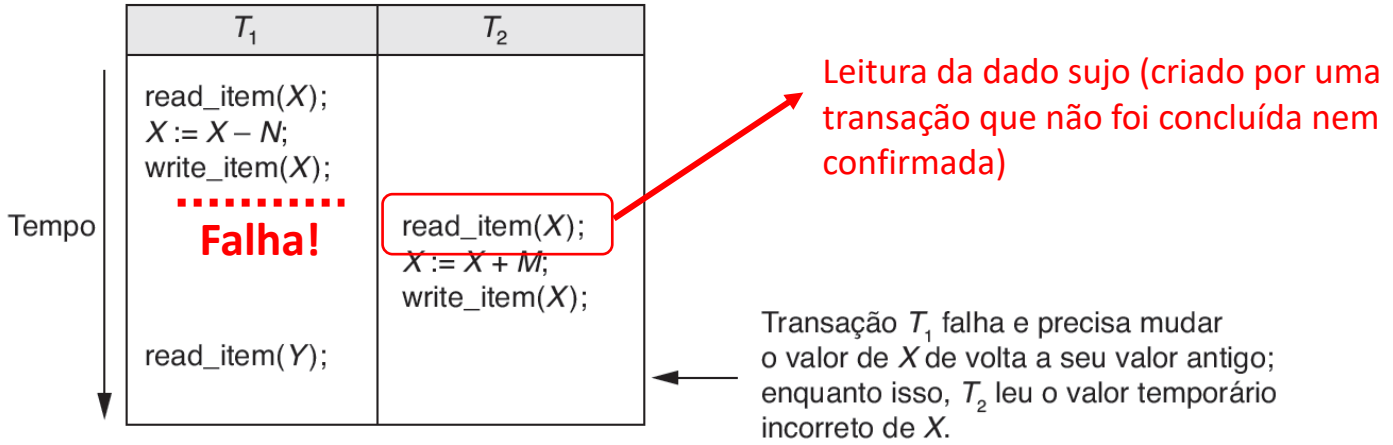
Por que o controle de concorrência é necessário?

- **O problema da atualização perdida**
  - Ocorre quando duas transações que acessam os mesmos itens do banco de dados têm suas operações intercaladas podendo resultar em itens do banco de dados com valor incorreto



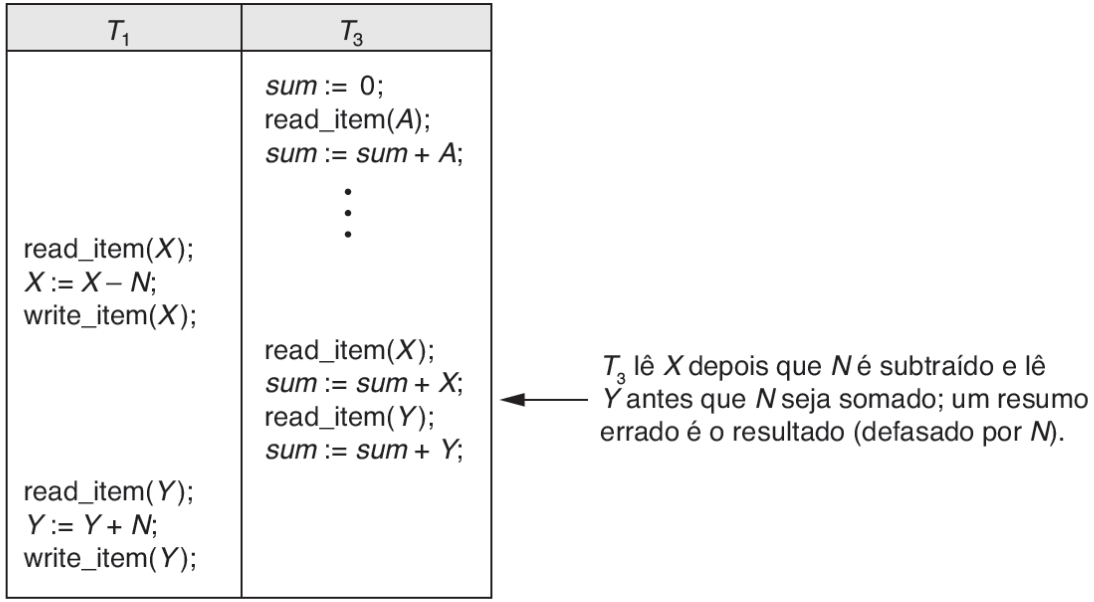
Por que o controle de concorrência é necessário?

- **O problema da atualização temporária (problema de leitura suja)**
  - Ocorre quando uma transação atualiza um item do banco de dados e depois a transação falha por algum motivo.
    - Nesse meio tempo, o item atualizado é acessado (lido) por outra transação, antes de ser alterado de volta para o seu valor original



Por que o controle de concorrência é necessário?

- **O problema do resumo incorreto**
  - Se **uma transação está calculando uma função de resumo de agregação** em uma série de itens de banco de dados, enquanto **outras transações estão atualizando** alguns desses itens, a função de agregação pode calcular alguns valores antes que eles sejam atualizados e outros, depois que eles forem atualizados



Por que o controle de concorrência é necessário?

- **O problema da leitura não repetitiva**
  - Uma transação *T* lê o mesmo item duas vezes e o item é alterado por outra transação *T'* entre as duas leituras
    - *T* recebe valores diferentes para suas duas leituras do mesmo item
  - Exemplo:
    - Durante uma transação de reserva aérea um cliente consulta a disponibilidade do assento em vários voos. Quando o cliente decide sobre um voo em particular a transação então lê o número de assentos nesse voo pela segunda vez antes de completar a reserva, e pode acabar lendo um valor diferente para o item

Apresentação baseada em:

- Ramez Elmasri e Shamkant B. Navathe 6° Ed (2010)
- Material da Profa. Dra. Sarajane Marques Peres (UNIVESP)

Banco de Dados

Introdução aos conceitos de Processamento de Transação (parte 2)

FACOM – UFMS

Vanessa Borges

vanessa.a.borges@ufms.br

Capítulo 21: Introdução aos conceitos e teoria de processamento de transações

Teoria da Serialização

- Um plano de execução **S** de **n transações** é **serializável** se for **equivalente** a um plano de execução serial das mesmas **n transações**

Valores iniciais: **X= 90, Y=90, N=3, M=2**

Plano Serial - A

T1	T2

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

Resultado: X=89, Y=93

Plano Serial - B

T1	T2

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

Resultado: X=89, Y=93

Plano Intercalado – S2

T1	T2

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

Resultado: X=89, Y=93
Serializável

Teoria da Serialização

- Um plano de execução **S** de **n transações** é **serializável** se for **equivalente** a um plano de execução serial das mesmas **n transações**

Valores iniciais: **X= 90, Y=90, N=3, M=2**

Plano Serial - A

T1	T2

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

Resultado: X=89, Y=93

Plano Serial - B

T1	T2

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

Resultado: X=89, Y=93

Plano Intercalado – S1

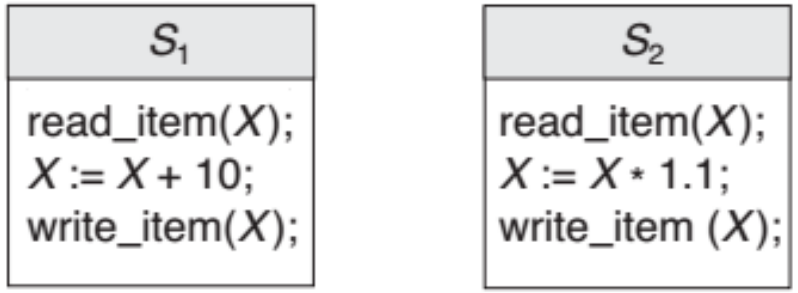
T1	T2

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

Resultado: X=92, Y=93
(Não serializável)
Problema da atualização perdida

Teoria da Serialização

- Equivalência no resultado não pode ser utilizado para definir a equivalência de planos de execução



- X=100 produzem o mesmo resultado
- Com outros valores pode gerar resultados diferentes

- Como testar se um plano de execução intercalado é equivalente a um plano serial?

- **Planos de execução equivalentes**
  - Equivalência de conflito (mais utilizado)
  - Equivalência de visão

Plano de execução: conflito de operações

- Duas operações são consideradas em **conflito** se:
  1. Pertencerem a transações diferentes
  2. Acessam o mesmo item do banco de dados
  3. Se forem operações:
     - write_item, write_item
     - write_item, read_item
     - read_item, write_item

T1	T2
read_item(X);	
write_item(X);	
	read_item(X);
	write_item(X);
read_item(Y);	
...	...

- A instrução write_item(X) de T1 entra em conflito com a instrução read_item(X) de T2.
- Porém, a instrução write_item(X) e T2 não está em conflito com a instrução read_item(Y) de T1.

Schedule serializável: equivalência por conflito

- Dois planos são considerados **equivalentes em conflito** se a **ordem das duas operações em conflito quaisquer for a mesma nos dois schedules**

Plano Serial - A

T1	T2

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

Plano Intercalado – S1

T1	T2
read_item(X); X:=X-N	
	read_item(X); X:=X+M;
write_item(X); read_item(Y);	
	write_item(X);
Y:=Y+N; write_item(Y);	

Não é equivalente por conflito
Problema da atualização perdida

Schedule serializável: equivalência por conflito

- Dois planos são considerados **equivalentes em conflito** se a **ordem das duas operações em conflito quaisquer for a mesma nos dois schedules**

Plano Serial - A

T1	T2

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

Plano Intercalado – S2

T1	T2

read_item(X);
X:=X-N
write_item(X);	

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

read_item(Y);
Y:=Y+N;
write_item(Y);	

Equivalente por conflito

Schedule serializável: equivalência por conflito

- Um plano de execução **S** de **n transações** é **serializável** se for **equivalente** a um plano de execução serial das mesmas **n transações**

Dois planos são considerados conflito serializáveis se forem equivalentes por conflito a um plano serial.

Plano Serial - A

T1	T2

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

Plano Serial - B

T1	T2

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

Plano Intercalado – S2

T1	T2

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

Equivalente por conflito

Como determinar se um schedule é serializável por conflito?

- Algoritmo que testa a serialização por conflito dos schedules
- Testa as operações de **read_item** e **write_item** para construir um grafo de precedência
  - **Algoritmo:**
    - Para cada transação T_i participante do Schedule S crie um nó no grafo
    - Para cada caso em S
      - T_j → read_item(X) depois de T_i → write_item(X)
        - Aresta $T_i \rightarrow T_j$
      - T_j → write_item(X) depois de T_i → read_item(X)
        - Aresta $T_i \rightarrow T_j$
      - T_j → write_item(X) depois de T_i → write_item(X)
        - Aresta $T_i \rightarrow T_j$
    - **É serializável se não houver ciclos no grafo**

Schedule serializável: serialização por conflito

- Algoritmo que testa a serialização por conflito dos schedules

Plano Serial - A

T1	T2

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

Plano Serial - B

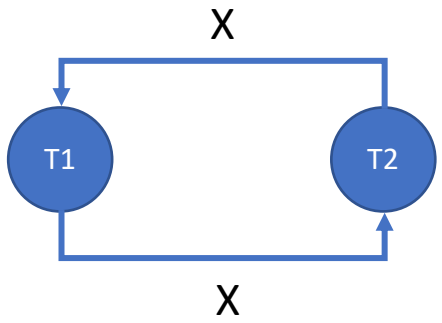
T1	T2

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

Plano Intercalado – S1

T1	T2
read_item(X); X:=X-N	
	read_item(X); X:=X+M;
write_item(X); read_item(Y);	
	write_item(X)
Y:=Y+N; write_item(Y);	

Problema da atualização perdida



Não é Serializável por conflito

Schedule serializável: serialização por conflito

- Algoritmo que testa a serialização por conflito dos schedules

Plano Serial - A

T1	T2

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

Plano Serial - B

T1	T2

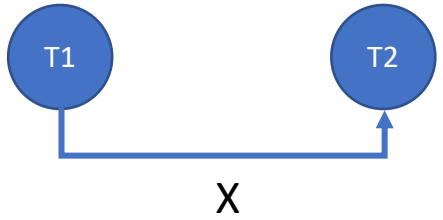
read_item(X);
X:=X-N
write_item(X);
read_item(Y);
Y:=Y+N;
write_item(Y);
	read_item(X);
	X:=X+M;
	write_item(X);

Plano Intercalado – S2

T1	T2

read_item(X);
X=X+N;
write_item(X)
 	
read_item(Y);
Y=Y+N;
write_item(Y);
	
	read_item(X);
	X=X+M;
	write_item(X);

Equivalente por conflito



Serializável por conflito

Schedule serializável: equivalência de visão

- Dois schedules S e S’ são considerados **equivalentes de visão** se as três condições a seguir forem satisfeitas:

1. Possuem as mesmas transações e operações

2. No schedule S, se há um read_item(X) em T_i, que seja valor original (antes de S ter iniciado) ou gravado por um write_item(X) em T_j, o mesmo acontece em S’

3. No schedule S, se write_item(Y) é a última operação em Y a gravar em T_k, o mesmo acontece em S’

Schedule serializável: equivalência por visão

Plano Serial - A

T1	T2

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

Plano Intercalado – S1

T1	T2

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

Plano Intercalado – S2

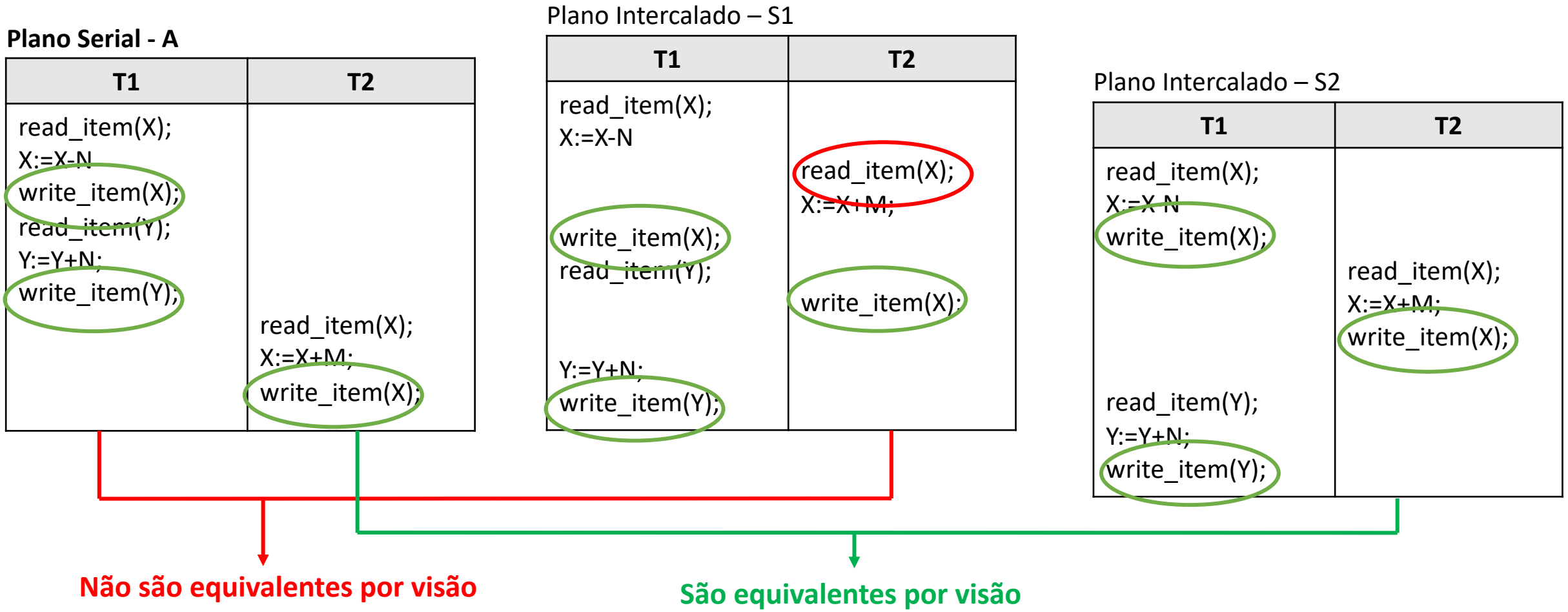
T1	T2

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

Não são equivalentes por visão

São equivalentes por visão

Schedule serializável: equivalência por visão



Schedule serializável: serialização de visão

Um schedule S é considerado serializável de visão se for equivalente de visão a um schedule serial.

Plano Serial - A

T1	T2

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

Plano Intercalado – S1

T1	T2

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

Plano Intercalado – S2

T1	T2

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

Não são equivalentes por visão

São equivalentes por visão

Conclusões

- Na prática, é impraticável testar a serialização de um plano de execução (schedule).

- A técnica usada na maioria dos SGBDs comerciais é projetar protocolos (conjuntos de regras) que — se seguidos por toda transação individual ou se impostos por um subsistema de controle de concorrência do SGBD — garantirão a serialização de todos os schedules em que as transações participam.
  - A técnica mais comum, chamada bloqueio em duas fases, é baseada no bloqueio de itens de dados para impedir que transações concorrentes interfiram umas com as outras, e na imposição de uma condição adicional que garanta a serialização.

Conclusões

- Na prática, é muito difícil testar a serialização de um schedule. A intercalação de operações de transações concorrentes — que normalmente são executadas como processos pelo sistema operacional — costuma ser determinada pelo Schedule do sistema operacional, que aloca recursos para todos os processos.
  - Se as transações forem executadas à vontade e depois o schedule resultante tiver a serialização testada, temos de cancelar o efeito do schedule se ele não for serializável. Esse é um problema sério, que torna essa **técnica impraticável.**

- A técnica usada na maioria dos SGBDs comerciais é projetar protocolos (conjuntos de regras) que — se seguidos por toda transação individual ou se impostos por um subsistema de controle de concorrência do SGBD — garantirão a serialização de todos os schedules em que as transações participam.

- No Capítulo 22, discutimos uma série de protocolos de controle de concorrência diferentes, que garantem a serialização.
  - A técnica mais comum, chamada bloqueio em duas fases, é baseada no bloqueio de itens de dados para impedir que transações concorrentes interfiram umas com as outras, e na imposição de uma condição adicional que garanta a serialização.