INF / UFG

Disciplina

Banco de Dados

Conteúdo

Escalonamento Baseado em Serialização

Serialização

Alguns escalonamentos são considerados corretos quando transações simultâneas são executadas:

>> **escalonamentos serializáveis**.

Exemplo: dois usuários submetem ao DBMS as Transações T1 e T2. Execuções seriais:

1. Executar todas as operações da transação T1 (em seqüência), seguido por todas as operações da transação T2 (em seqüência).
2. Executar todas as operações da transação T2 (em seqüência), seguido por todas as operações da transação T1 (em seqüência).

Escalonamento serial: as operações de cada transação são executadas consecutivamente, sem quaisquer operações intercaladas da outra transação.

Serialização

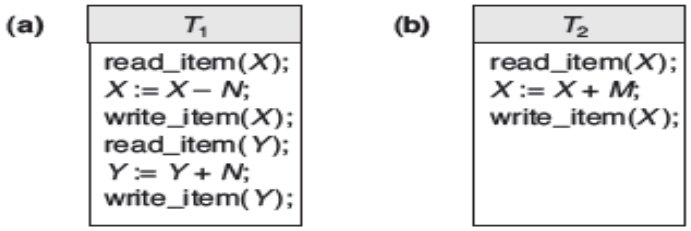


Figura 21.2
Duas transações de exemplo. (a) Transação T_1. (b) Transação T_2.

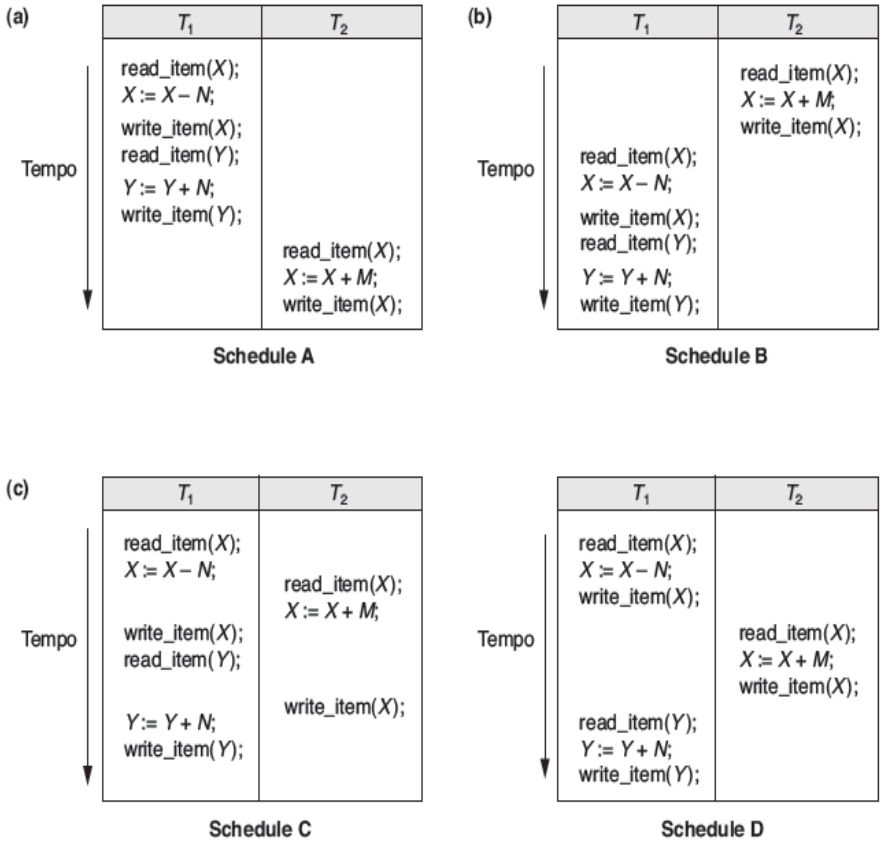


Figura 21.5
Exemplos de schedules seriais e não seriais envolvendo as transações T_1 e T_2. (a) Schedule serial A: T_1 seguida por T_2. (b) Schedule serial B: T_2 seguida por T_1. (c) Dois schedules não seriais C e D com intercalação de operações.

Serialização

O problema com os escalonamentos seriais é que eles limitam a concorrência, proibindo a intercalação das operações.

Em um escalonamento serial, se uma transação espera por uma operação de I / O para completar, não podemos mudar o recurso processador para outra transação, desperdiçando, assim, valioso tempo de processamento da CPU.

Além disso, se alguma transação T é bastante longa, as outras transações devem aguardar T completar todas as suas operações antes de iniciar.

Assim, os escalonamentos seriais são considerados inaceitáveis na prática:

>> que escalonamentos não seriais são equivalentes aos escalonamentos seriais ?

Serialização

Considere os Escalonamentos A, B, C e D.

Inicialmente, X = 90 e Y = 90, e n = 3 e m = 2.

Ao final dos Escalonamentos A e B, X = 89 e Y = 93.

Ao final do Escalonamento C, X = 92 e Y = 93.

Ao final do Escalonamento D, X = 89 e Y = 93.

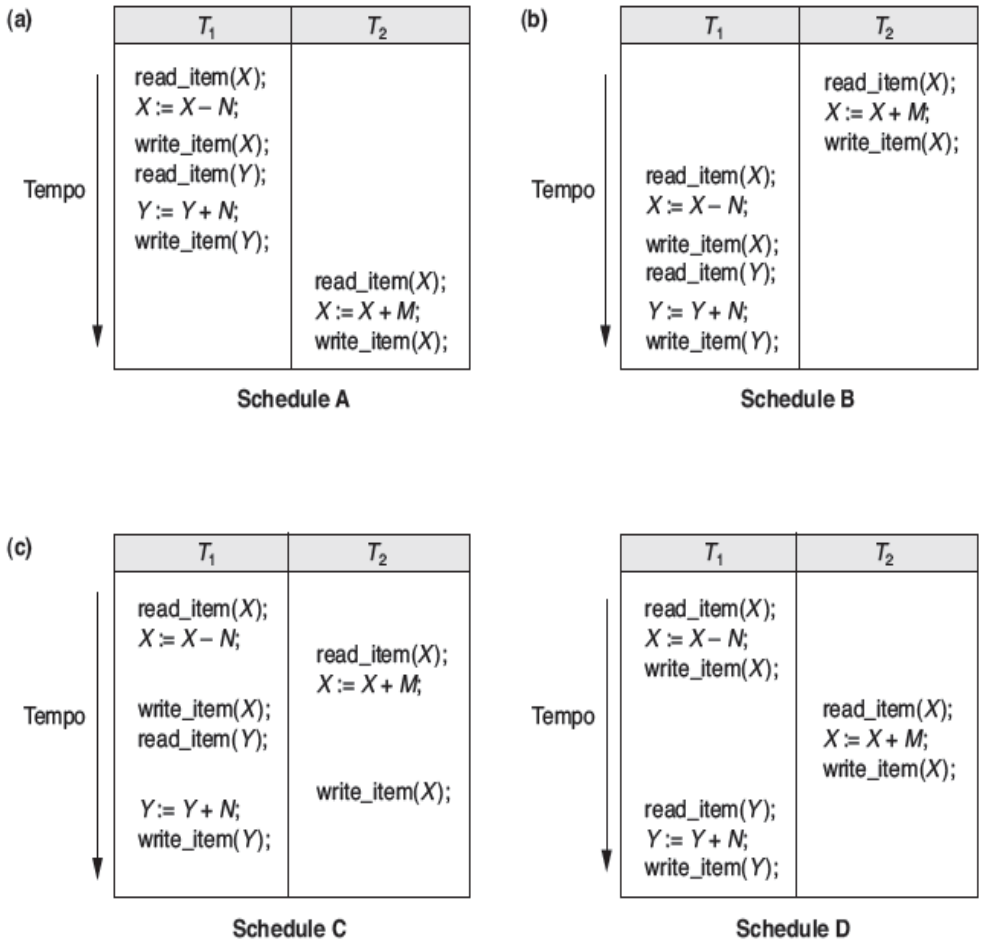


Figura 21.5

Exemplos de schedules seriais e não seriais envolvendo as transações T_1 e T_2. (a) Schedule serial A: T_1 seguida por T_2. (b) Schedule serial B: T_2 seguida por T_1. (c) Dois schedules não seriais C e D com intercalação de operações.

Serialização

Um Escalonamento S de n transações é **serializável** se for equivalente a algum escalonamento serial das mesmas n transações.

Há n! possíveis escalonamentos seriais de n transações, e muito mais escalonamentos não seriais.

Pode-se formar dois grupos disjuntos de escalonamentos não seriais:

serializáveis, e

não serializáveis

Serialização

Como testar se um escalonamento é serializável ?

A maioria dos métodos de controle de concorrência não realmente testam serialização.

Na prática, protocolos, ou regras, são desenvolvidos que garantem que qualquer escalonamento que segue estas regras será serializável.

Contudo, é importante saber testar serialização para obter uma melhor compreensão dos protocolos de controle de concorrência.

Grafo de Precedência de Transações

Utilizar um grafo (dirigido) de precedência de transações para observar conflito de serialização.

Algoritmo: Testar se há conflito de serialização no Escalonamento S.

1. Para cada transação Ti em S, crie um nó rotulado Ti no grafo.

2. Para os casos onde um **Ti executa um write_item (X)** e depois **Tj executa um read_item (X)**, crie uma aresta (Ti → Tj) no grafo.

3. Para os casos onde um **Ti executa um read_item (X)** e depois **Tj executa um write_item (X)**, crie uma aresta (Ti → Tj) no grafo.

4. Para os casos onde um **Ti executa um write_item (X)** e depois **Tj executa um write_item (X)**, crie uma aresta (Ti → Tj) no grafo.

5. O Escalonamento S é serializável se e somente se o grafo de precedência não possui ciclos.

Grafo de Precedência de Transações

Em um grafo dirigído, considere que cada arco (Ti → Tj) possui um nó inicial (Ti) e um nó final (Tj).

Um ciclo em um grafo dirigido é uma sequência de arestas

C = ((Tj → Tk), (Tk → Tp), ..., (Ti → Tj)),

com a propriedade de que:

(i) o nó inicial de cada arco (exceto a primeiro arco) é o mesmo nó que termina o arco anterior; e

(ii) o nó inicial do primeiro arco é o mesmo nó final do último arco.

Ou seja, a sequência começa e termina no mesmo nó.

Grafo de Precedência de Transações

Em um **escalonamento não serial S**, uma aresta de Ti para Tj significa que:

>> a Transação Ti devem vir antes da Transação Tj em qualquer escalonamento serial equivalente a S, pois duas operações em conflito aparecem em S naquela ordem.

Se não houver ciclo no grafo, é possível criar um escalonamento serial SS equivalente a S:

>> sempre que houver uma aresta (Ti → Tj) em S, Ti deve aparecer antes de Tj em SS.

Se o grafo de um Escalonamento S possui um ciclo, não podemos criar qualquer escalonamento serial equivalente a S.

Grafo de Precedência de Transações

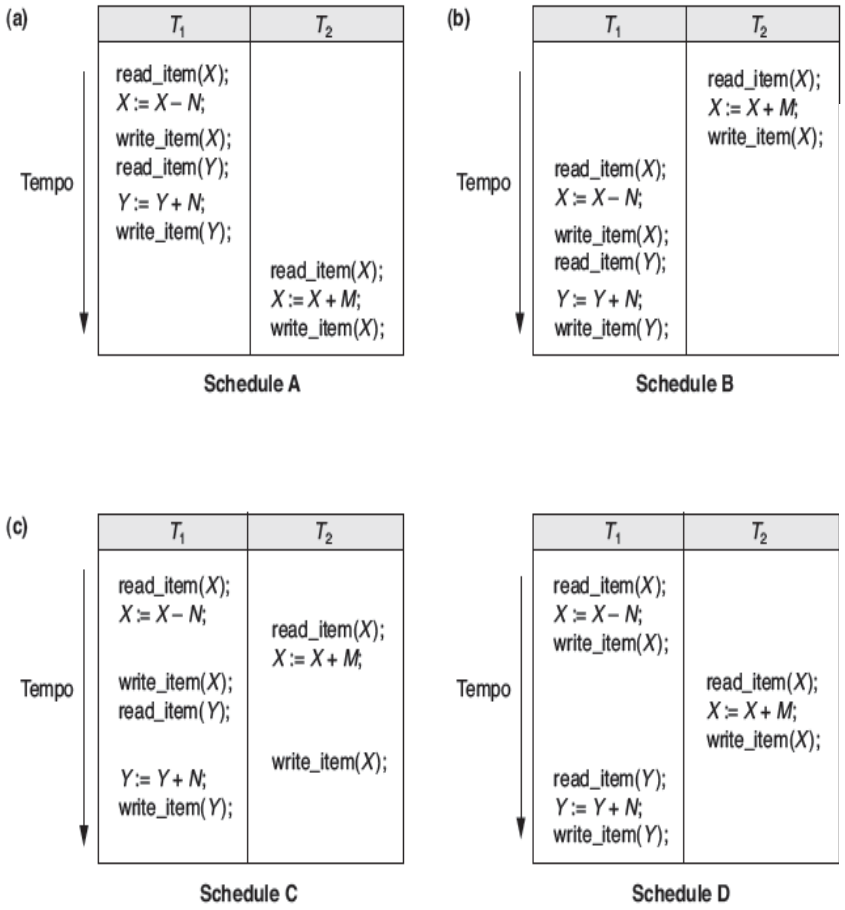


Figura 21.5

Exemplos de schedules seriais e não seriais envolvendo as transações T_1 e T_2. (a) Schedule serial A: T_1 seguida por T_2. (b) Schedule serial B: T_2 seguida por T_1. (c) Dois schedules não seriais C e D com intercalação de operações.

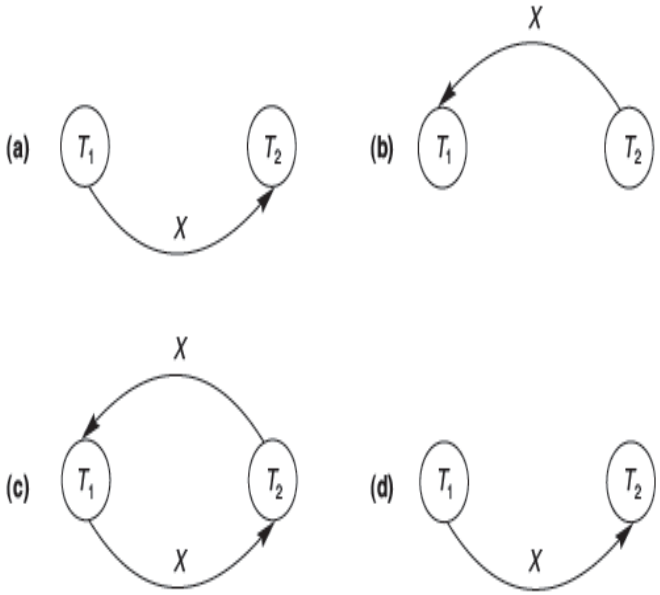


Figura 21.7

Construindo os grafos de precedência para os schedules A a D da Figura 21.5 para testar a serialização de conflito. (a) Grafo de precedência para o schedule serial A. (b) Grafo de precedência para o schedule serial B. (c) Grafo de precedência para o schedule C (não serializável). (d) Grafo de precedência para o schedule D (serializável, equivalente ao schedule A).

Grafo de Precedência de Transações

Transaction T_1
read_item(X);
write_item(X);
read_item(Y);
write_item(Y);

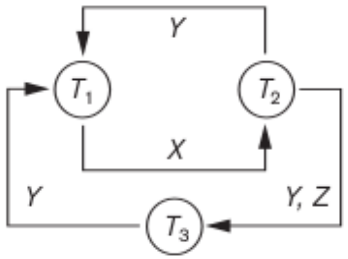
Transaction T_2
read_item(Z);
read_item(Y);
write_item(Y);
read_item(X);
write_item(X);

Transaction T_3
read_item(Y);
read_item(Z);
write_item(Y);
write_item(Z);

Time

Transaction T_1	Transaction T_2	Transaction T_3
	read_item(Z);	
	read_item(Y);	
	write_item(Y);	
		read_item(Y);
		read_item(Z);
read_item(X);		
write_item(X);		
		write_item(Y);
		write_item(Z);
	read_item(X);	
read_item(Y);		
write_item(Y);		
	write_item(X);	

Schedule E



Equivalent serial schedules

None

Reason

Cycle $X(T_1 \rightarrow T_2), Y(T_2 \rightarrow T_1)$
Cycle $X(T_1 \rightarrow T_2), YZ\ (T_2 \rightarrow T_3), Y(T_3 \rightarrow T_1)$

Grafo de Precedência de Transações

Transaction T_1
read_item(X);
write_item(X);
read_item(Y);
write_item(Y);

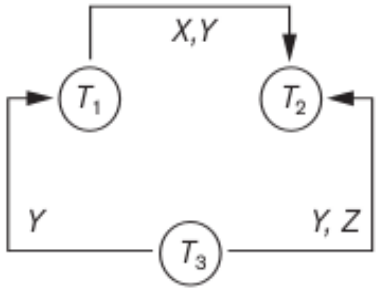
Transaction T_2
read_item(Z);
read_item(Y);
write_item(Y);
read_item(X);
write_item(X);

Transaction T_3
read_item(Y);
read_item(Z);
write_item(Y);
write_item(Z);

Time

Transaction T_1	Transaction T_2	Transaction T_3
		read_item(Y);
		read_item(Z);
read_item(X);		
write_item(X);		
		write_item(Y);
		write_item(Z);
	read_item(Z);	
read_item(Y);		
write_item(Y);		
	read_item(Y);	
	write_item(Y);	
	read_item(X);	
	write_item(X);	

Schedule F

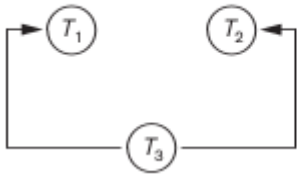


Equivalent serial schedules

$T_3 \rightarrow T_1 \rightarrow T_2$

Grafo de Precedência de Transações

Qual escalonamento não serial equivale ao grafo abaixo ?
Note que há dois escalonamentos seriais equivalentes.



Equivalent serial schedules

$T_3 \rightarrow T_1 \rightarrow T_2$

$T_3 \rightarrow T_2 \rightarrow T_1$

Transaction T_1
read_item(X);
write_item(X);
read_item(Y);
write_item(Y);

Transaction T_2
read_item(Z);
read_item(Y);
write_item(Y);
read_item(X);
write_item(X);

Transaction T_3
read_item(Y);
read_item(Z);
write_item(Y);
write_item(Z);

Como Serialização é usada para Controle de Concorrência

Na prática, é muito difícil testar a serialização de um escalonamento.

O que ocorreria se as transações submetidas ao SGBD fossem executadas, e somente depois o escalonamento resultante fosse testado com respeito à serialização?

>> caso o escalonamento não fosse serializável, as transações precisariam ser abortadas.

A abordagem acima é impraticável.

A abordagem adotada na maioria dos SGBDs é aplicar regras e protocolos que assegurem a serialização, sem ter que testar os escalonamentos.

Exercício

Quais dos seguintes escalonamentos são serializáveis ? Para cada escalonamento serializável, determine os escalonamentos seriais equivalentes.

(a) r1 (X); r3 (X); w1(X); r2(X); w3(X)

(b) r1 (X); r3 (X); w3(X); w1(X); r2(X)

(c) r3 (X); r2 (X); w3(X); r1(X); w1(X)

(d) r3 (X); r2 (X); r1(X); w3(X); w1(X)