Banco de Dados

Pós-Graduação em Ciência da Computação

Prof. Dr. Ronaldo Celso Messias Correia
ronaldo.correia@unesp.br

Concorrência

Controle de concorrência

- Protocolos baseados em bloqueio
- Protocolos baseados em timestamp
- Granularidade múltipla
- Esquemas multiversão
- Tratamento de deadlock

Protocolos baseados em bloqueio

- Um bloqueio é um mecanismo para controlar o acesso concorrente a um item de dados
- Bloqueio Binário:
  - Pode ter dois estados ou valores: locked (bloqueado) ou unlocked (desbloqueado)
  - No máximo uma transação pode manter o bloqueio de um determinado item
  - Duas transações não podem acessar o mesmo item de maneira concomitante
- Bloqueio Múltiplo: podem ser bloqueados em dois modos:
  - 1. Modo exclusivo (X). O item de dados pode ser lido e também escrito. O bloqueio X é solicitado pela instrução lock-X.
  - 2. Modo compartilhado (S). O item de dados só pode ser lido. O bloqueio S é solicitado pela instrução lock-S.
- As solicitações de bloqueio são feitas ao gerenciador de controle de concorrência. A transação só pode prosseguir após a concessão da solicitação.

Protocolos baseados em bloqueio (cont.)

- Uma transação pode receber um bloqueio sobre um item se o bloqueio solicitado for compatível com os bloqueios já mantidos sobre o item por outras transações
- Matriz de compatibilidade de bloqueio

	S	X
S	true	false
X	false	false

- Qualquer quantidade de transações pode manter bloqueios compartilhados sobre um item, mas se qualquer transação mantiver um bloqueio exclusivo sobre um item, nenhuma outra pode manter qualquer bloqueio sobre o item.
- Se um bloqueio não puder ser concedido, a transação solicitante deve esperar até que todos os bloqueios incompatíveis mantidos por outras transações tenham sido liberados. O bloqueio é então concedido.

Protocolos baseados em bloqueio (cont.)

- Exemplo de uma transação realizando bloqueio:

T_1	T_2
lock-X (B) read(B) $B := B - 50$ write (B) unlock(B) lock-X(A) read(A) $A := A + 50$ write (A) unlock(A)	lock-S(A) read(A) unlock(A) lock-S(B) read(B) unlock(B) display($A + B$)

- O bloqueio acima não é suficiente para garantir a seriação - se A e B fossem atualizados entre a leitura de A e B, a soma exibida estaria errada.
- Um protocolo de bloqueio é um conjunto de regras seguidas por todas as transações enquanto solicita e libera bloqueios. Os protocolos de bloqueio restringem o conjunto de escalonamentos possíveis.

Protocolos baseados em bloqueio (cont.)

- Transações T_1 e T_2 executadas concorrentemente

T_1	T_2	Gerenciador de Controle de concorrência
lock-X (B)		
		grant-X(B,T_1)
read(B)		
$B := B - 50$		
write (B)		
unlock(B)		
	lock-S(A)	
		grant-S(A,T_2)
	read(A)	
	unlock(A)	
	lock-S(B)	
		grant-S(B,T_2)
	read(B)	
	unlock(B)	
	display($A + B$)	
lock-X(A)		
		grant-X(A,T_2)
read(A)		
$A := A + 50$		
write (A)		
unlock(A)		

Armadilhas dos protocolos baseados em bloqueio

- Considere o escalonamento parcial

T_3	T_4
lock-X(B)	
read(B)	
$B := B - 50$	
write(B)	
	lock-S(A)
	read(A)
	lock-S(B)
lock-X(A)	

- Nem T_3 e nem T_4 podem ter progresso - a execução de lock-S(B) faz com que T_4 espere que T_3 libere seu bloqueio sobre B, enquanto a execução de lock-X(A) faz com que T_3 espere que T_4 libere seu bloqueio sobre A.
- Essa situação é chamada de deadlock
  - Para controlar o deadlock, um dentre T_3 ou T_4 precisa ser revertido (rollback) e seus bloqueios liberados.

Armadilhas dos protocolos baseados em bloqueio (cont.)

- A maioria dos protocolos de bloqueio têm potencial para gerar deadlocks. Deadlocks são um mal necessário.
- Inanição também é possível se o gerenciador de controle de concorrência for mal projetado. Por exemplo:
  - Uma transação pode estar esperando por um X-lock em um item, enquanto uma sequência de outras transações solicitam e recebem um S-lock no mesmo item.
  - A mesma transação sofre repetidos rollbacks devido a deadlocks.
- O gerenciador de controle de concorrência pode ser projetado para prevenir a inanição (starvation).

O protocolo de bloqueio em duas fases

- Esse é um protocolo que garante escalonamentos seriáveis por conflito.
- Requer que cada transação emita solicitações de bloqueio e desbloqueio em duas fases:
  - Fase 1: Fase de expansão / crescimento
    - transação pode obter bloqueios
    - transação não pode liberar bloqueios
  - Fase 2: Fase de Encolhimento
    - transação pode liberar bloqueios
    - transação não pode obter bloqueios
- O protocolo garante serialização. Pode-se provar que as transações podem ser serializáveis na ordem dos seus pontos de bloqueio (lock points) (ou seja, o ponto onde a transação adquire o seu último bloqueio).

O protocolo de bloqueio em duas fases (cont.)

- Exemplo de bloqueio em duas fases

T_1	T_2

Lock-X (Aplic);
Read (Aplic);
Aplic.Saldo = Aplic.Saldo – 500;
Write (Aplic);
Lock-X (Conta);
Unlock (Aplic); // Inicia 2ª fase
Read (Conta);	

	Lock-S (Conta);
Conta.Saldo = Conta.Saldo + 500; Write (Conta); Unlock (Conta);	*Bloqueada*

	Read (Conta);
	Lock-S (Aplic);
	Unlock (Conta); // Inicia 2ª fase
	Read (Aplic);
	Print (Conta.Saldo + Aplic.Saldo);
	Unlock (Aplic);

O protocolo de bloqueio em duas fases (cont.)

- O bloqueio em duas fases não garante ausência de deadlocks.

T_1	T_2
Lock-X (Aplic); Read (Aplic); Aplic.Saldo = Aplic.Saldo – 500;	
	Lock-S (Conta); Read (Conta); Lock-S (Aplic);
Write (Aplic); Lock-X (Conta);	*Bloqueada*
Bloqueada	*Bloqueada*
Não executa: Unlock (Aplic); Read (Conta); Conta.Saldo = Conta.Saldo + 500; Write (Conta); Unlock (Conta);	**Não executa:** Unlock (Conta); Read (Aplic); Print (Conta.Saldo + Aplic.Saldo); Unlock (Aplic);

O protocolo de bloqueio em duas fases (cont.)

- Bloqueio em duas fases não evita rollback em cascata

T_1	T_2
Lock-X (Aplic); Read (Aplic); Aplic.Saldo = Aplic.Saldo – 500; Write (Aplic); Lock-X (Conta); Unlock (Aplic); Read (Conta);	
	Lock-S (Aplic); Read (Aplic); Lock-S (Conta);

Conta.Saldo = Conta.Saldo + 500;
Write (Conta); // ABORTA
Unlock(Conta);
	Bloqueada

	Read (Conta); // ABORTA
	Unlock (Conta);
	Print (Conta.Saldo + Aplic.Saldo);
	Unlock (Aplic);

Variantes do bloqueio em duas fases (cont.)

- Rollback em cascata é possível sob o bloqueio em duas fases. Para evitar isso, usa-se um protocolo modificado chamado bloqueio em duas fases severo (strict two-phase locking). Aqui, uma transação deve manter todos os seus bloqueios exclusivos até que ela execute o commit ou o abort.
- O bloqueio em duas fases rigoroso é ainda mais restrito: todos os bloqueios são mantidos até que a transação execute commit ou abort. Nesse protocolo, as transações podem ser serializadas na ordem em que elas executam commit.

T_1	T_2
Lock-X (Aplic); Read (Aplic); Aplic.Saldo = Aplic.Saldo – 500; Write (Aplic); Lock-X (Conta); Read (Conta);	
	Lock-S (Aplic);
Conta.Saldo = Conta.Saldo + 500; Write (Conta); **Unlock (Aplic);** Unlock(Conta);	*Bloqueada*
	Read (Aplic); Lock-S (Conta); **Unlock (Aplic);** Read (Conta); Print (Conta.Saldo + Aplic.Saldo); Unlock (Conta);

T_1	T_2
Lock-X (Aplic); Read (Aplic); Aplic.Saldo = Aplic.Saldo – 500; Write (Aplic); Lock-X (Conta); Read (Conta);	
	Lock-S (Aplic);
Conta.Saldo = Conta.Saldo + 500; Write (Conta); **Unlock (Aplic);** Unlock(Conta);	*Bloqueada*
	Read (Aplic); Lock-S (Conta); Read (Conta); Print (Conta.Saldo + Aplic.Saldo); **Unlock (Aplic);** Unlock (Conta);

Conversões de bloqueio

- Bloqueio em duas fases com conversões de bloqueio:
  - Primeira fase:
    - pode adquirir um bloqueio-S sobre o item
    - pode adquirir um bloqueio-X sobre o item
    - pode converter um bloqueio-S para um bloqueio-X (upgrade)
  - Segunda fase:
    - pode liberar um bloqueio-S
    - pode liberar um bloqueio-X
    - pode converter um bloqueio-X para um bloqueio-S (downgrade)
- Esse protocolo garante a seriação. Mas ainda conta com o programador para inserir as diversas instruções de bloqueio.

Aquisição automática de bloqueios

- Uma transação T_i emite a instrução de leitura/escrita padrão, sem chamadas de bloqueio explícitas.
- A operação read(D) é processada como:

```
if Ti tem um bloqueio sobre D
  then
        read(D)
  else
        begin
          se necessário, espera até que nenhuma outra
              transação tenha um bloqueio-X sobre D
          concede a Ti um bloqueio-S sobre D;
          read(D)
        end
```

Aquisição automática de bloqueios (cont.)

- write(D) é processado como:

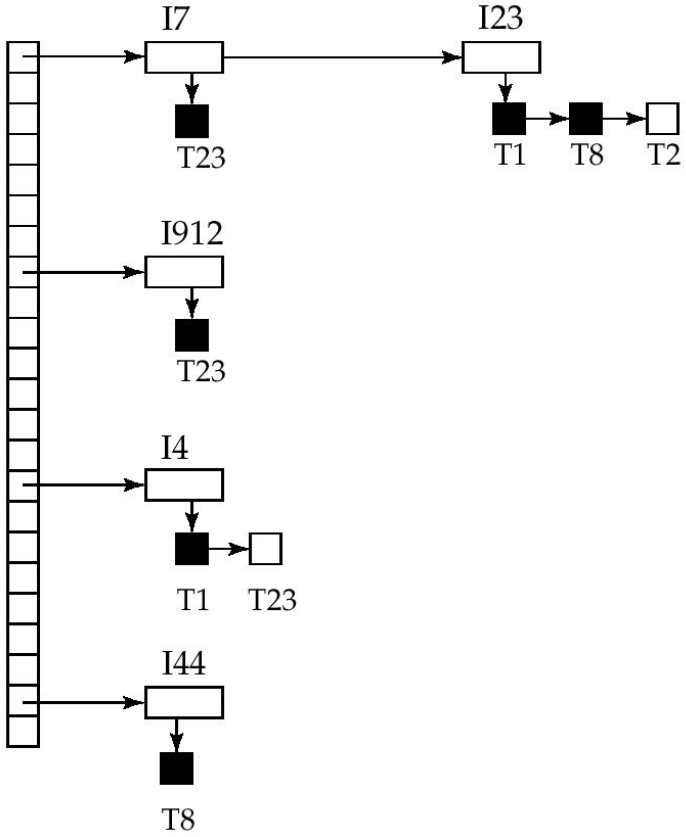
```
if Ti tem um bloqueio-X sobre D
      then
        write(D)
     else
      begin
        se for preciso, esperar até que nenhuma outra transação tenha um bloqueio sobre D,
        if Ti tem um bloqueio-S sobre D
            then
               upgrade do bloqueio sobre D para bloqueio-X
            else
               concede a Ti um bloqueio-X sobre D
            write(D)
      end;
```

- Todos os bloqueios são liberados após o commit ou abort

Implementação do bloqueio

- Um Gerenciador de Bloqueios pode ser implementado como um processo separado, ao qual as transações enviam pedidos de bloqueios e desbloqueios.
- O gerenciador de bloqueios responde a um pedido de bloqueio enviando uma mensagem de concessão de bloqueio (ou uma mensagem pedindo à transação para executar rollback, em caso de um deadlock).
- A transação solicitante espera até que o seu pedido seja respondido.
- O gerenciador de bloqueios mantém uma estrutura de dados chamada tabela de bloqueios para registrar os bloqueios concedidos e os pedidos pendentes.
- A tabela de bloqueios normalmente é implementada como uma tabela hash em memória, indexada pelo nome do item de dados que está sendo bloqueado.

Tabela de bloqueio



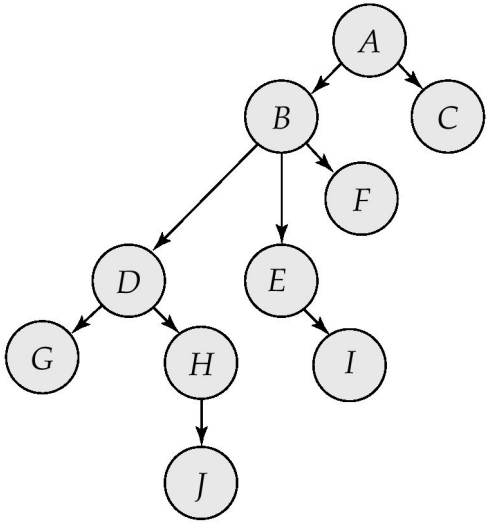
- Retângulos pretos indicam bloqueios concedidos, brancos indicam solicitações aguardando
- A tabela de bloqueio também registra o tipo de bloqueio concedido ou solicitado
- A nova solicitação é acrescentada ao final da fila de solicitações para o item de dados, e concedida se for compatível com todos os bloqueios anteriores
- As solicitações de desbloqueio resultam na solicitação sendo excluída e solicitações posteriores são verificadas para saber se agora podem ser concedidas
- Se a transação abortar, todas as solicitações aguardando ou concedidas da transação são excluídas
  - o gerenciador de bloqueio pode manter uma lista de bloqueios mantidos por cada transação, para implementar isso de forma eficiente

Protocolos baseados em grafos

- Os protocolos baseados em grafos são uma alternativa ao bloqueio em duas fases
- Imponha uma ordenação parcial $\rightarrow$ sobre o conjunto D = {d1, d2 ,..., dh} de todos os itens de dados.
  - Se $d_i \rightarrow d_j$ então qualquer transação acessando d_i e d_j precisa acessar d_i antes de acessar d_j.
  - Implica que o novo conjunto D agora pode ser visto como um grafo acíclico direcionado, chamado grafo de banco de dados.
- O protocolo de árvore é um tipo simples de protocolo de grafo.
  - Somente bloqueios exclusivos são permitidos.

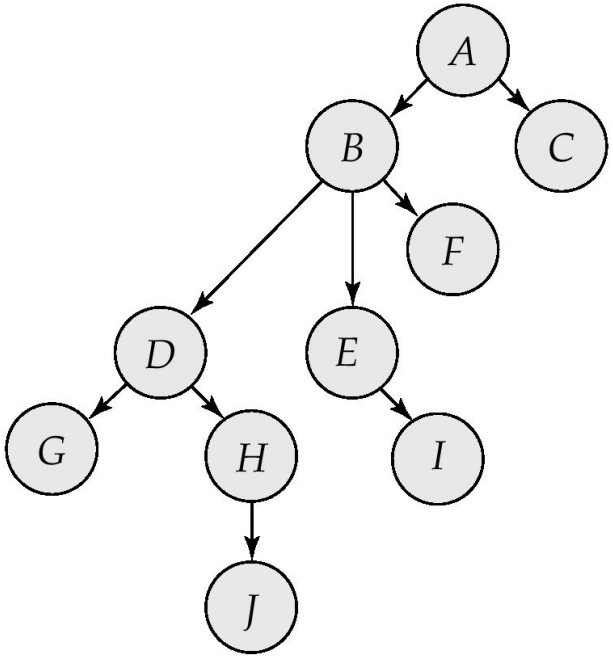
Protocolo de árvore

- A única instrução de bloqueio permitida é lock-x.
- Cada transação Ti pode bloquear um item de dados no máximo uma vez, e precisa observar as seguintes regras:
  - O primeiro bloqueio por Ti pode ser sobre qualquer item de dados.
  - Subsequentemente, um item de dado Q pode ser bloqueado por Ti somente se o pai de Q for
  - Os itens de dados podem ser desbloqueados a qualquer momento.
  - Um item de dados que foi bloqueado e desbloqueado por Ti não pode mais tarde ser bloqueado novamente por Ti
- Apresenta a vantagem de realizar o desbloqueio mais cedo do que é feito no protocolo em duas fases (reduz o tempo de espera e aumenta a concorrência)
- Todos os escalonamentos que são legais sob o protocolo de árvore são séries de conflito



Schedule seriável sob o protocolo de árvore

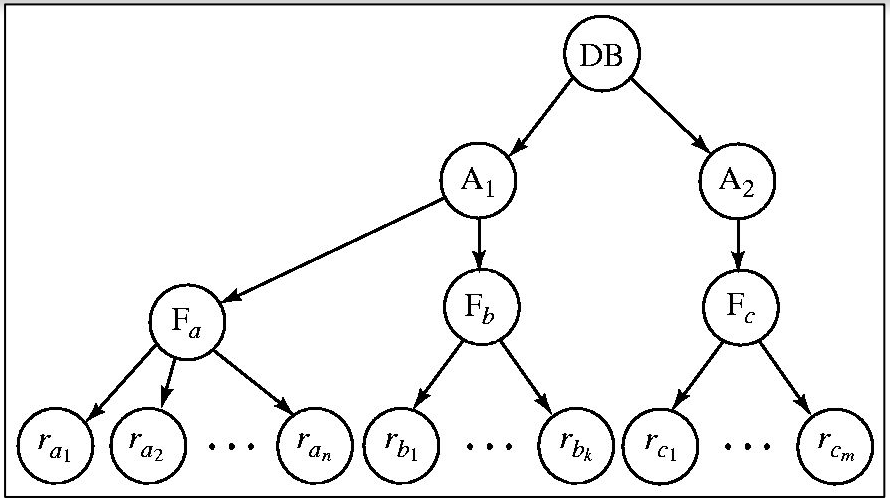
T_{10}	T_{11}	T_{12}	T_{13}
lock-X(B)			
	lock-X(D)		
	lock-X(H)		
	unlock(D)		
lock-X(E)			
lock-X(D)			
unlock(B)			
unlock(E)			
		lock-X(B)	
		lock-X(E)	
	unlock(H)		
lock-X(G)			
unlock(D)			
			lock-X(D)
			lock-X(H)
			unlock(D)
			unlock(H)
		unlock(E)	
		unlock(B)	
unlock (G)			



Granularidades múltiplas

- Permitem que os itens de dados sejam de vários tamanhos e definem uma hierarquia de granularidades de dados, onde as granularidades pequenas são aninhadas dentro das maiores. Agrupar vários itens de dados e tratá-los como uma unidade de sincronismo individual
- Podem ser representadas graficamente como uma árvore (mas não confundir com o protocolo de bloqueio em árvore)
- Quando uma transação bloqueia um nó na árvore explicitamente, ela implicitamente bloqueia os descendentes do nó no mesmo modo.
- Ao invés de bloquear um item de dados, podemos bloquear tuplas, tabelas, blocos de disco ou DBs
- Granularidade do bloqueio (nível na árvore onde o bloqueio é feito):
  - granularidade menor (mais baixo na árvore): alta concorrência, alta sobrecarga de bloqueio
  - granularidade maior (mais alto na árvore): baixa sobrecarga de bloqueio, baixa concorrência

Exemplo de hierarquia de granularidade



- O nível mais alto na hierarquia de exemplo é o banco de dados inteiro.
- Os níveis abaixo são do tipo área, arquivo e registro, nessa ordem.

Modos de bloqueio de intenção

- Além dos modos de bloqueio S e X, existem três modos de bloqueio adicionais com granularidade múltipla:
  - Intenção de compartilhamento (IS): indica o bloqueio explícito em um nível inferior da árvore, mas apenas com bloqueios compartilhados.
  - Intenção de exclusividade (IX): indica o bloqueio explícito em um nível mais baixo com bloqueios exclusivos ou compartilhados
  - Compartilhamento com Intenção de exclusividade (SIX): a sub-árvore com raiz nesse nó é bloqueada explicitamente no modo compartilhado e o bloqueio explícito está sendo feito em um nível inferior com bloqueios no modo exclusivo.
- os bloqueios de intenção permitem que um nó de nível mais alto seja bloqueado no modo S ou X sem ter que verificar todos os nós descendentes.

Matriz de compatibilidade com modos de bloqueio de intenção

➢ A matriz de compatibilidade para todos os modos de bloqueio é:

	IS	IX	S	S IX	X
IS	✓	✓	✓	✓	×
IX	✓	✓	×	×	×
S	✓	×	✓	×	×
S IX	✓	×	×	×	×
X	×	×	×	×	×

Protocolos baseados em Timestamp

- Cada transação tem uma timestamp emitido quando entra no sistema. Se uma transação antiga Ti tem timestamp TS(Ti), uma nova transação Tj recebe a timestamp TS(Tj) de modo que TS(Ti) < TS(Tj).
- O protocolo gerencia a execução concorrente tal que os timestamp determinam a ordem de seriação.
- Existem dois mecanismos simples:
  - Clock do Sistema
  - Contador Lógico
- Para garantir esse comportamento, o protocolo mantém para cada dado Q dois valores timestamp:
  - W-timestamp(Q) é o maior timestamp de qualquer transação que executou write(Q) com sucesso.
  - R-timestamp(Q) é a maior timestamp de qualquer transação que executou read(Q) com sucesso.

Protocolos baseados em Timestamp

- O protocolo de ordenação por timestamp garante que quaisquer operações read e write em conflito sejam executadas por ordem de timestamp.
- Suponha que uma transação Ti emita um read(Q)

1. Se TS(Ti) < W-timestamp(Q), então Ti precisa ler um valor de Q que já foi modificado. Logo, a operação read é rejeitada, e Ti é revertida.
2. Se TS(Ti) > W-timestamp(Q), então a operação read é executada, e R-timestamp(Q) é definido como o máximo de R-timestamp(Q) e TS(Ti).

Protocolos baseados em Timestamp

- Suponha que a transação Ti emita write(Q).
  - Se TS(Ti) < R-timestamp(Q), então o valor de Q que Ti está produzindo foi necessário anteriormente, e o sistema considerou que esse valor nunca seria produzido. Logo, a operação write é rejeitada, e Ti é revertido.
  - Se TS(Ti) < W-timestamp(Q), então Ti está tentando escrever um valor obsoleto de Q. Logo, essa operação write é rejeitada, e Ti é revertida.
  - Caso contrário, a operação write é executada, e W-timestamp(Q) é definida como TS(Ti).

Exemplo de uso do protocolo

- Um escalonamento parcial para vários itens de dados para transações com timestamp 1, 2, 3, 4, 5

T_1	T_2	T_3	T_4	T_5
				read(X)
read(Y)	read(Y)			
		write(Y)		
		write(Z)		
				read(Z)
	read(X)			
	abort			
read(X)				
		write(Z)		
		abort		
				write(Y)
				write(Z)

Exatidão do protocolo de ordenação por timestamp

- O protocolo de ordenação por timestamp garante a seriação, pois todos os arcos no grafo de precedência são da forma:

Assim, não haverá ciclos no gráfico de precedência

- O protocolo de timestamp garante liberdade de impasse, pois nenhuma transação precisa esperar.
- Mas o escalonamento pode não ser livre de cascata, e pode nem sequer ser recuperável.

Facilidade de recuperação e liberdade de cascata

- Problema com protocolo de ordenação por timestamp:
  - Suponha que Ti aborte, mas Tj tenha lido um item de dados escrito por Ti
  - Então, Tj precisa abortar; se Tj tivesse permitido o commit anterior, o escalonamento não seria recuperável.
  - Além do mais, qualquer transação que tenha lido um item de dados escrito por Tj precisa abortar
  - Isso pode levar ao rollback em cascata - ou seja, uma cadeia de rollbacks
- Solução:
  - Uma transação é estruturada de modo que suas escritas sejam todas realizadas no final de seu processamento
  - Todas as escritas de uma transação formam uma ação atômica; e nenhuma transação pode ser executada enquanto uma transação estiver sendo escrita
  - Uma transação que aborta é reiniciada com um novo timestamp

Manuseio de Deadlock

- Considere as duas transações a seguir:

  T1: write (X) T2: write(Y)

  write (Y) write (X)

- Escalonamento com deadlock

T_1	T_2
lock-X on X write (X)	
	lock-X *on* Y write (Y)
wait for **lock-X** on Y	wait for **lock-X** on X

Manuseio de Deadlock

- O sistema está em deadlock se existe um conjunto de transações tal que cada transação no conjunto está esperando por outra transação no conjunto.
- Protocolos de Prevenção de Deadlock garantem que o sistema nunca entrará em um estado de deadlock. Algumas estratégias de prevenção são:
  - Exigir que cada transação bloqueie todos os seus itens de dados antes de iniciar a execução (pré-declaração).
  - Impor ordenação parcial de todos os itens de dados e exigir que uma transação possa bloquear itens de dados somente na ordem especificada pela ordem parcial (protocolo baseado em grafos).

Outras Estratégias de Prevenção de Deadlock

- Os esquemas a seguir usam apenas timestamps de transação para garantir a prevenção de deadlocks.
- esquema **wait-die** (esperar-morrer) - não preemptivo
  - Transações mais antigas podem esperar que transações mais novas liberem itens de dados. Transações mais novas nunca esperam pelas mais antigas. Em vez disso, elas sofrem rollback.
  - Uma transação pode morrer várias vezes antes de adquirir um item de dado.
- esquema **wound-wait** (ferir-esperar) - preemptivo
  - Transações mais antigas ferem (forçam o rollback) de transações mais novas em vez de esperar por elas. Transações mais novas podem esperar pelas mais antigas.
  - Pode ocorrer menos rollbacks que o esquema wait-die.

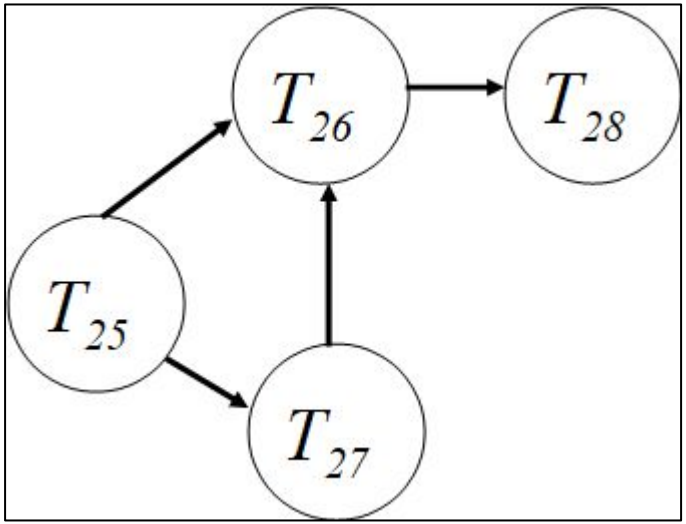
Prevenção de Deadlock (Cont.)

- Em ambos os esquemas wait-die e wound-wait, uma transação que sofreu rollback é reiniciada com o seu timestamp original. Assim, transações mais antigas têm precedência sobre as mais novas, e a inanição é evitada.
- Esquemas com base em Timeout:
  - Uma transação espera por um bloqueio somente por uma quantidade especificada de tempo. Após esse tempo, ocorre um timeout e a transação sofre rollback.
  - Assim, deadlocks são impossíveis.
  - Simples de implementar, mas pode ocorrer inanição. Também é difícil de determinar um valor ideal para o intervalo de timeout.

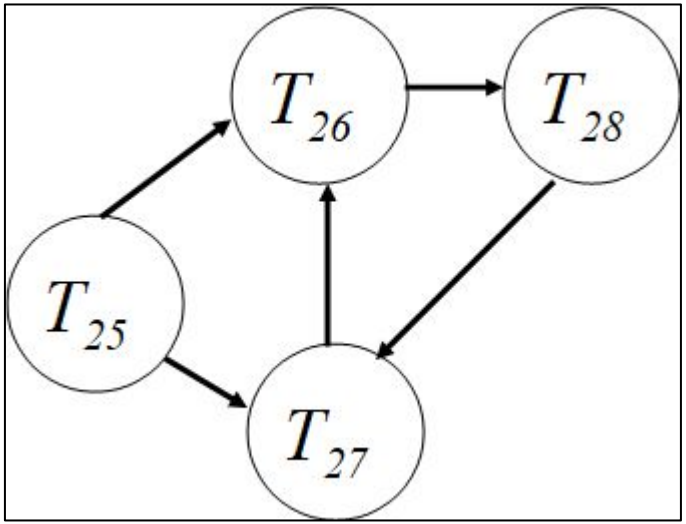
Detecção de Deadlock (Cont.)

- Deadlocks podem ser descritos como um grafo de espera (wait-for graph), que consiste de um par **G = (V,E),**
  - V é um conjunto de vértices (todas as transações do sistema)
  - E é um conjunto de arestas; cada elemento é um par ordenado Ti --> Tj.
- Se Ti --> Tj está em E, então existe uma aresta direcionada de Ti para Tj, indicando que Ti está esperando que Tj libere um item.
- Quando Ti solicita um item de dado que está sendo mantido por Tj, então uma aresta Ti --> Tj é inserida no grafo de espera. Esta aresta será removida somente quando Tj não mais estiver mantendo um item de dado exigido por Ti.
- O sistema está em um estado de deadlock se e somente se o grafo de espera tiver um ciclo. Deve-se invocar um algoritmo de detecção de deadlocks periodicamente para procurar por ciclos.

Detecção de Deadlock (Cont.)



Grafo de espera sem ciclo



Grafo de espera com ciclo

Recuperação de Deadlocks

- Quando um deadlock é detectado:
  - Alguma transação terá que sofrer rollback (“fazer uma vítima”) para quebrar o deadlock. Escolher como vítima a transação que irá incorrer em um custo mínimo.
  - Rollback - determinar quanto da transação deve ser desfeito Rollback total: Abortar a transação e então reiniciá-la.
  - O mais eficiente é fazer o rollback da transação somente o necessário para quebrar o deadlock.
  - Ocorrerá a inanição se a mesma transação for escolhida sempre como vítima. Incluir o número de rollbacks como fator de custo para evitar inanição.