Técnicas de Controle de Concorrência

Laboratório de Bancos de Dados

Introdução

- Discutiremos técnicas de controle de concorrência usadas para assegurar a propriedade de isolamento. Assegurar a serialização usando um conjunto de regras. Ex: técnicas de bloqueio. Mais usado na prática.

Agenda

- Técnicas de bloqueio.
- Técnicas de controle de concorrência de Multiversão.
- Granularidade.
- Usando Bloqueios para o controle de concorrência em índices.
- Tópicos adicionais.

Técnicas de Bloqueio

- Um bloqueio (lock) é uma variável associada a um item de dados que descreve a condição do item em relação às possíveis operações que podem ser aplicadas a ele.
- A seguir discutiremos a natureza e os tipos de bloqueio.
- Depois apresentaremos protocolos que usam bloqueio para garantir serialização de planos.
- Finalmente apresentaremos dois problemas associados ao uso de bloqueios.

Tipos de Bloqueio e Tabelas de Bloqueio de Sistema

- Apresentaremos os diversos tipos de bloqueios gradualmente, começando desde os mais simples e restritivos.
- **Bloqueios binários.** São de dois estados ou valores: bloqueios (1) e desbloqueios (0). Se o valor do bloqueio em X for 1, o item não pode ser acessado. Se for 0, o item pode ser acessado quando solicitado → LOCK(X).
- Duas operações, lock_item e unlock_item, são usadas com o bloqueio binário.
- Uma transação solicita o acesso a um ítem a través do lock_item. Tendo acesso ao item ela bloqueia o acesso às outras transações até fazer um unlock_item, permitindo o acesso ao item. Permite exclusão mutua. Veja fig.

Figura 18.1 Operações de bloqueio e desbloqueio para bloqueios binários.

```
lock_item(X)
B: if LOCK(X)=0 (*item está desbloqueado*)
   then LOCK (X)←1 (*bloquear o item*)
   else begin
     wait (até que lock(X) = 0 e
     o gerenciador de bloqueio reinicia a transação);
     go to B
     end;

unlock_item (X):
LOCK (X)←0; (*desbloquear o item*)
 se nenhuma transação estiver esperando
 então reiniciar uma das transações em espera;
```

Unidades indivisíveis – seções críticas

Existe uma fila de espera associada ao comando wait.

Bloqueios binários

- É muito simples implementar um bloqueio binário; é necessário uma variável binária LOCK associada a cada item. Simplificando, cada bloqueio pode ser um registro de 3 campos <nome do item,LOCK, transação de bloqueio>, mais uma fila para as transações que estão esperando acessar o item.
- Esses registros podem ser mantidos para os items bloqueados numa tabela de bloqueio. Pode estar organizada como uma tabela hash. O sistema gerenciador de bloqueio encarregado.

Bloqueios binários

- Com este esquema, toda transação deve obedecer as seguintes regras (impostas pelo módulo gerenciador de bloqueios):
  - Uma transação T deve garantir a operação lock_item(X) antes que qualquer operação ler_item(X) ou escrever_item(X) seja executada.
  - Uma transação T deve garantir a operação unlock_item(X) depois que todas as operações ler_item(X) ou escrever_item(X) sejam completadas.
  - Uma transação T não executará uma operação lock_item(X) se ela já tiver o bloqueio no item X.
  - Uma transação T não executará uma operação unlock_item(X), a menos que ela já tenha o bloqueio no item X.
- Neste caso duas transações não podem acessar o mesmo item ao mesmo tempo.

Bloqueios Compartilhados/Exclusivos (ou Leitura/Escrita)

- Bloqueio binário muito restritivo. Um item X poderia ser acessado por várias transações no caso de leitura. No caso de alteração, uma transação deve ter acesso exclusivo.
- Os bloqueios compartilhados têm essa função.
- Têm três operações de bloqueio: read_lock(X), write_lock(X) e unlock(X).
- Um item “read_locked” é chamado de bloqueado-compartilhado
- Um item “write_locked” é chamado de bloqueado-exclusivo.

Bloqueios Compartilhados/Exclusivos (ou Leitura/Escrita)

- Implementar: cada registro da tabela de bloqueio terá 4 campos: < nome do item,LOCK, num_de_leituras, transação(ões)_bloqueio>. O valor de LOCK ou é bloqueado para leitura ou escrita.
- Se LOCK(X) = write_locked, o valor de transação(ões)_bloqueio é uma única transação que controla o bloqueio exclusivo em X.
- Se LOCK(X) = read_locked, o valor de transação(ões)_bloqueio é uma lista de uma ou mais transações que controlam o bloqueio compartilhado em X.
- Veja figura descrevendo as três operações.

Figura 18.2 Operação de bloqueio e desbloqueio para bloqueios de dois modos (leitura-escrita ou compartilhado-exclusivo).

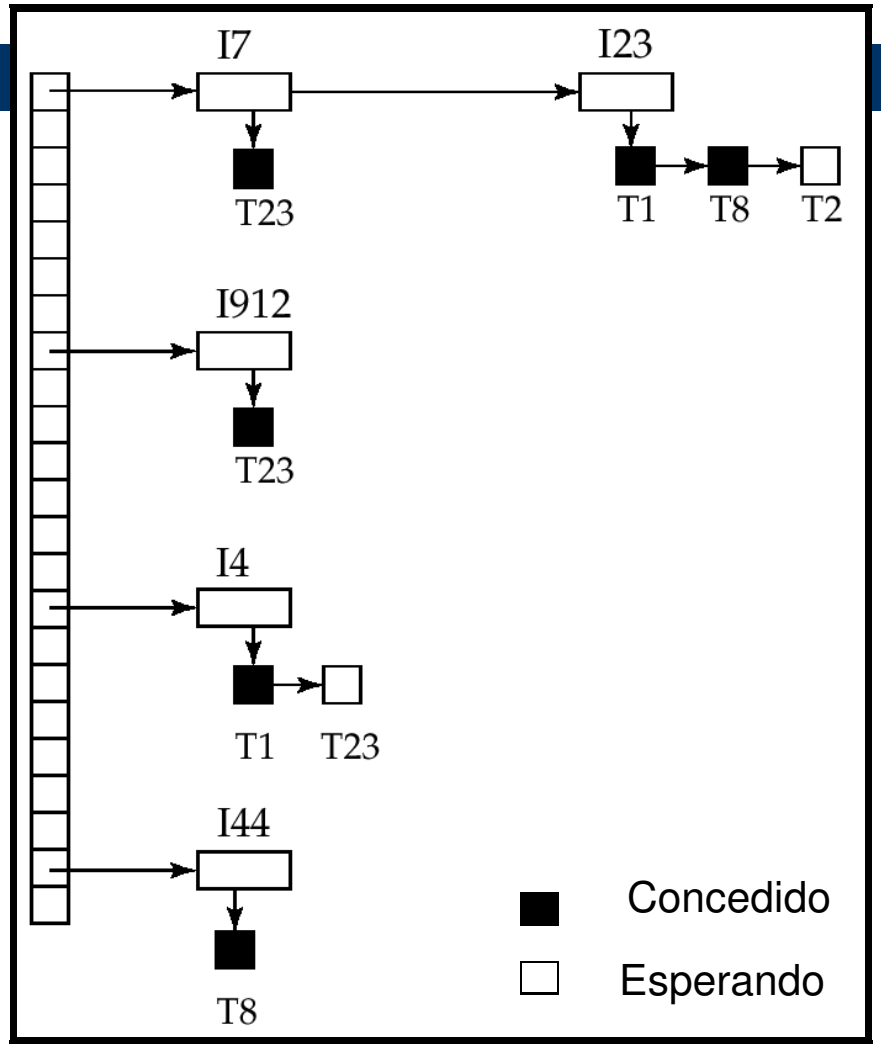
```
read_lock (X):
  B: if LOCK (X)="unlocked"
     then begin LOCK (X)← "read-locked";
                no_of_reads(X)← 1
          end
     else if LOCK(X)="read-locked"
          then no_of_reads(X)← no_of _reads(X) + 1
          else begin wait (until LOCK (X)="unlocked" and
                    the lock manager wakes up the transaction);
               go to B
               end;

write_lock (X):
  B: if LOCK (X)="unlocked"
     then LOCK (X)← "write-locked";
     else begin
          wait (until LOCK(X)="unlocked" and
            the lock manager wakes up the transaction);
          go to B
          end;

unlock (X):
  if LOCK (X)="write-locked"
     then begin LOCK (X)← "unlocked";
          wakeup one of the waiting transactions, if any
          end
     else if LOCK(X)="read-locked"
          then begin
               no_of_reads(X)← no_of_reads(X) – 1;
               if no_of_reads(X)=0
                 then begin LOCK (X)="unlocked";
                      wakeup one of the waiting transactions, if any
                      end
                 end;
```

Figura 3 Tabela hashing de bloqueios.

Tabela de Bloqueios



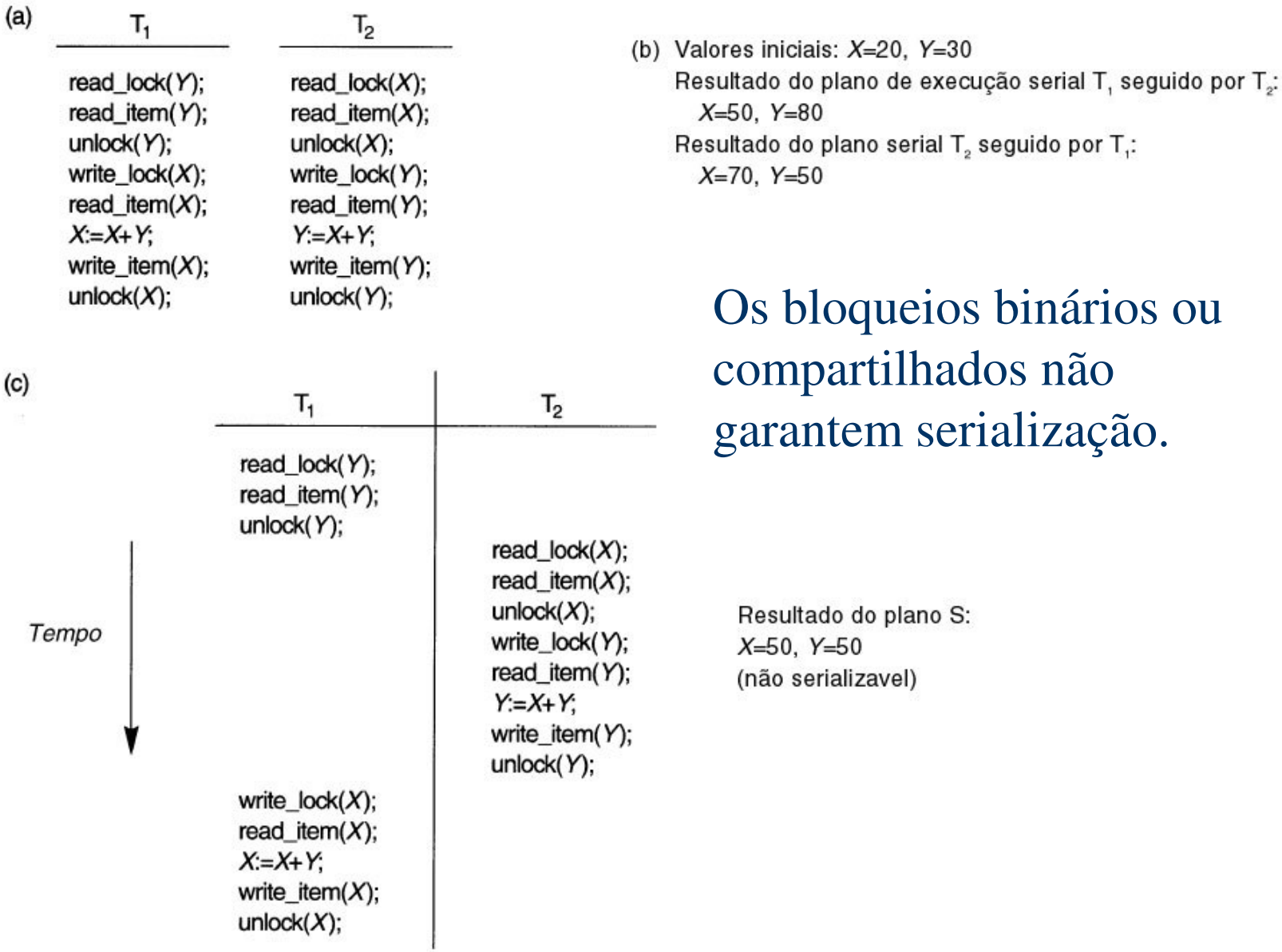
Bloqueios Compartilhados/Exclusivos (ou Leitura/Escrita)

- Quando usamos o esquema de bloqueio Compartilhados/Exclusivo, o sistema deve impor as regras:
  1. Uma transação T deve garantir a operação read_lock(X) ou write_lock(X) antes de qualquer operação ler_item(X) ser executada em T.
  2. Uma transação T deve garantir a operação write_lock(X) antes de qualquer operação escrever_item(X) ser executada em T.
  3. Uma transação T deve garantir a operação unlock(X) depois que todas as operações ler_item(X) e escrever_item(X) são completadas em T.
  4. Uma transação T não usará uma operação read_lock(X) se ela já controlar um bloqueio de leitura ou um bloqueio de escrita no item X. Relaxada.
  5. Uma transação T não usará uma operação write_lock(X) se ela já controlar um bloqueio de leitura ou um bloqueio de escrita no item X. Relaxada.
  6. Uma transação T não usará uma operação unlock(X) a menos que ela já controle um bloqueio de leitura ou de escrita em um item X.

Conversão de Bloqueios

- Relaxamento das condições 4 e 5, permitir a conversão do bloqueio: **Promoção**, uma transação T ter um bloqueio de leitura sobre um item X e depois passar para um bloqueio de escrita; **rebaixamento**, processo contrário.
- Quando é usada a conversão de bloqueios, a tabela de bloqueio deve incluir identificadores de transação na estrutura de registro. As operações devem ser mudadas.

Figura 4 Transações que não obedecem ao bloqueio em duas fases. (a) Duas transações T_1 e T_2. (b) Resultado dos possíveis planos de execução seriais de T_1 e T_2. (c) Um plano de execução não serializável S que usa bloqueios.

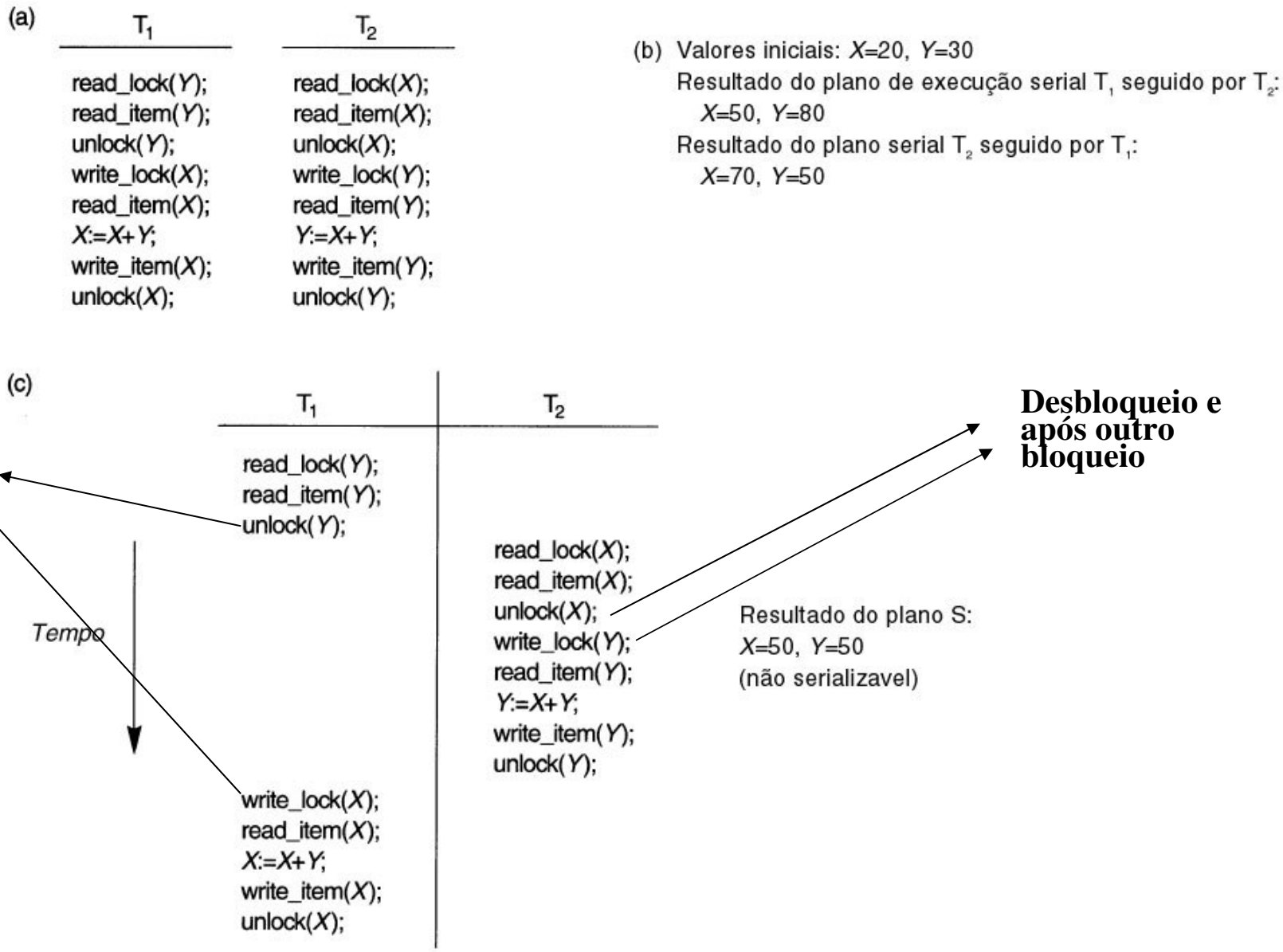


Os bloqueios binários ou compartilhados não garantem serialização.

Garantindo serialização pelo Bloqueio em Duas fases

- Uma transação segue o protocolo de bloqueio em duas fases se todas as operações de bloqueio precedem a primeira operação de desbloqueio. Tem duas fases:
  1. expansão ou crescimento: novos bloqueios nos itens podem ser adquiridos, mas não podem ser liberados; e
  2. encolhimento: os bloqueios existentes podem ser liberados, mas novos bloqueios não podem ser adquiridos.
- Com conversão de bloqueios, a promoção é feita durante a expansão, e o rebaixamento na fase de encolhimento.
- As transações T1 e T2 da figura passada não seguem o protocolo em duas fases. Porquê?

Figura 4 Transações que não obedecem ao bloqueio em duas fases. (a) Duas transações T_1 e T_2. (b) Resultado dos possíveis planos de execução seriais de T_1 e T_2. (c) Um plano de execução não serializável S que usa bloqueios.



Garantindo serialização pelo Bloqueio em Duas fases

- As transações T1´ e T2´ (reescritas de T1 e T2) para impor o bloqueio em duas fases (Veja Fig. 18.4). O plano de execução acima não pode ser gerado.
- Pode ser provado que, se toda transação em um plano seguir o protocolo de bloqueio em duas fases, é garantido que o plano seja serializável. Não há necessidade de testes.
- O bloqueio em duas fases pode limitar a quantidade de concorrência que pode surgir em um plano de execução.

Figura 5 Transações T_1' e T_2', que são as mesmas que T_1 e T_2 da Figura 18.3, mas que seguem o protocolo de bloqueio de duas fases. Observe que elas podem produzir um deadlock.

	T_1'	T_2'	
Expansão	read_lock (*Y*);	read_lock (*X*);	Expansão
	read_item (*Y*);	read_item (*X*);	
	write_lock (*X*);	write_lock (*Y*);	
Encolhimento	unlock (*Y*);	unlock (*X*);	Encolhimento
	read_item (*X*);	read_item (*Y*);	
	X:=*X*+*Y*;	*Y*:=*X*+*Y*;	
	write_item (*X*);	write_item (*Y*);	
	unlock (*X*);	unlock (*Y*);	

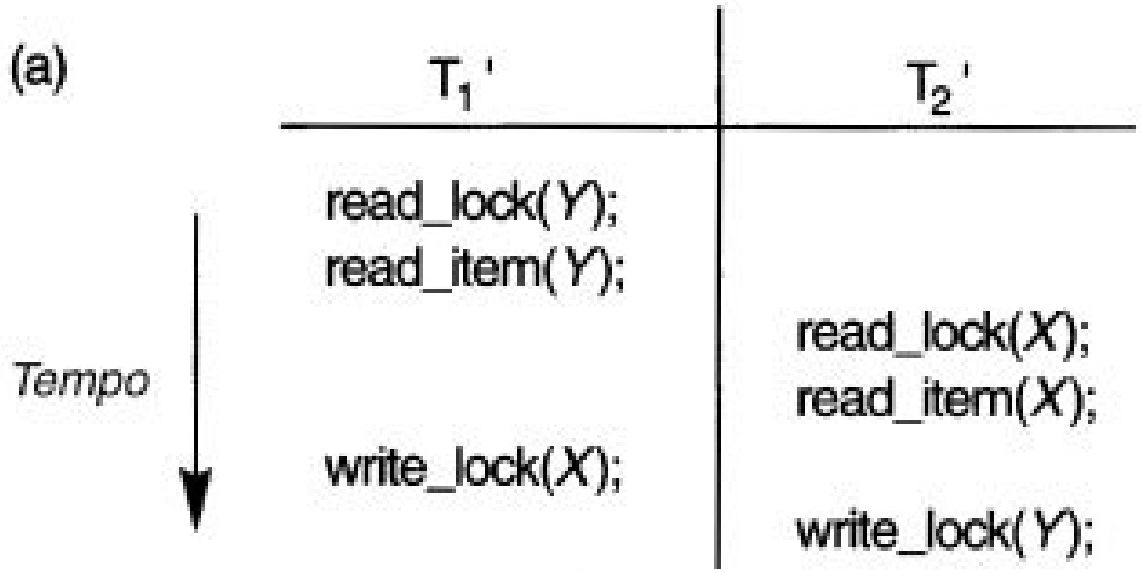
Bloqueio em Duas Fases Básico, Conservador, Estrito e Rigoroso

- Vários tipos de bloqueios em duas fases (2PL), o mencionado acima é o **básico**.
- 2PL **Conservador** ou estático: a transação deve bloquear todos os itens que ela acessa antes de iniciar a sua execução. Difícil de usar na prática devido à necessidade da pré-declaração de read-set e write-set que não é possível na maioria das situações.
- 2PL **Estrito**, garante planos estritos. T não libera nenhum de seus bloqueios exclusivos até que ela efetive ou aborte.

Lidando com Deadlock (Impasse) e Starvation (Inanição)

- O deadlock ocorre quando cada transação em um conjunto de duas ou mais transações espera por algum item que esteja bloqueado por alguma outra transação T´ no conjunto. Ex: fig. 18.5 as transações T1´ e T2´ estão em deadlock. Nenhuma transação pode acessar os itens X e Y.

Figura 18.5 Ilustração do problema de deadlock: plano de execução parcial de T_1' e T_2' que está em um estado de deadlock.



Protocolos de Prevenção de Deadlock

- Um protocolo de prevenção para bloqueios em duas fases conservador, requer que cada transação bloqueie todos os itens que ela necessita no avanço (não prático). Ela não começa até não conseguir bloquear todos os itens que precisa. Limita a concorrência.
- Outra opção envolve a ordenação de todos os itens no BD. A transação que os necessita, os bloqueará nessa ordem. Não é prático.

Protocolos de Prevenção de Deadlock

- Outros esquemas, permitem decidir o que fazer com uma transação envolvida em uma possível situação de deadlock: ela deveria ser bloqueada e ter de esperar ser abortada ou assumir e abortar uma outra transação?
- Usam o conceito de timestamp da transação TS(T), id único para cada transação. Baseados na ordem em que as transações são iniciadas.
- $\therefore$ TS(T1)<TS(T2), se T1 inicia antes de T2.
- Dois esquemas: esperar-morrer (wait-die) e ferir-esperar (wound-wait)

Protocolos de Prevenção de Deadlock

- Suponha que Ti tente bloquear um item X mas não pode porque X está bloqueado por Tj com um bloqueio conflitante. As regras seguida são:
  - **Esperar-morrer**: Se TS(Ti)<TS(Tj), Ti pode esperar; senão (Ti mais nova) aborta Ti e reinicia mais tarde com o mesmo timestamp. As transações mais velhas são as que esperam, nenhum ciclo é criado.
  - **Ferir-esperar**: Se TS(Ti)<TS(Tj), aborta Tj (Ti fere Tj) e reinicia mais tarde com o mesmo timestamp; senão (Ti mais nova) Ti espera. As transações somente esperam as mais velhas, nenhum ciclo é criado.
- Ambos os esquemas finalizam abortando a transação mais jovem, que pode estar envolvida em um deadlock. Problema de aborto desnecessário.

Protocolos de Prevenção de Deadlock

- Outro grupo que não usa timestamp: algoritmos “no waiting” e “cautious waiting”.
  - “No waiting”, se uma transação não estiver apta a obter um bloqueio, ela será imediatamente abortada e, então, reiniciada depois de um certo tempo.
  - “cautious waiting”, diminui o número de abortos desnecessários. A transação Ti tenta bloquear X, mas não pode porque Tj o bloqueou.
    - Se Tj não está bloqueado (não está esperando por algum outro item bloqueado), então Ti é bloqueado e poderá esperar; caso contrário, aborta Ti.

    É dead-lock free.

Detecção de Deadlock e Timeouts

- Neste caso o sistema verifica se um estado de deadlock realmente existe. É atrativa se soubermos que existem poucas interferências entre as transações (transações curtas, bloqueando poucos itens). Caso contrário, melhor prevenção.
- Grafo “wait-for”. Um nó por cada transação sendo executada. Quando uma transação Ti está esperando bloquear um item X bloqueado por Tj, uma ligação (Ti → Tj) é criada no grafo. A ligação é eliminada quando X for liberada. Existe um deadlock quando este grafo tiver um ciclo.
- O problema quando verificar? : número de transações concorrentes e tempo de espera médio.

Detecção de Deadlock e Timeouts

- Em estado de deadlock, algumas transações devem ser abortadas. Seleção de vitimas. Evitar selecionar transações em execução há muito tempo e com muitas atualizações.
- Outro esquema: timeouts. Método prático, se uma transação esperar por um período maior que um tempo definido, se assume que está em deadlock e deve abortar.

Inanição (Starvation)

- Ocorre quando uma transação não pode continuar por um período indefinido de tempo, enquanto que outras sim.
- Parcialização do esquema de espera.
- Solução um esquema de espera imparcial → uma fila.
- Outro esquema é o de prioridade por tempo de espera ou abortadas muitas vezes. Os esquemas “wait-die” e “wound-wait” evitam inanição.

Controle de concorrência baseado em ordenação de Timestamp

- Timestamp → identificador único criado pelo SGBD para identificar uma transação → momento de início da transação. TS(T).
- Não são usados bloqueios → não ocorrem deadlocks.

O algoritmo de ordenação por Timestamp

- Se uma transação velha T_i tem um time-stamp TS(T_i), uma transação nova T_j é atribuida um time-stamp TS(T_j) tal que TS(T_i) <TS(T_j).
- O algoritmo gerencia a execução concorrente de tal forma que os time-stamps determinam a ordem de serializabilidade.
- Para fazer isso, o algoritmo associa a cada item X do BD, dois valores de timestamp:
  - **Read_TS**(*X*) é o maior timestamp entre todos os timestamps de transações que tenham lido o item X com sucesso. Read_TS(X) = TS(T), onde T é a mais nova que tenha lido X com sucesso.
  - **Write_TS**(*X*) é o maior timestamp entre todos os timestamps de transações que tenham escrito o item X com sucesso. Write_TS(*X*) = TS(T), onde T é a mais nova que tenha escrito X com sucesso.

O algoritmo de ordenação por Timestamp (Cont.)

- O algoritmo se certifica de que qualquer operação de leitura ou escrita conflitante é executada em ordem de timestamp.
- Suponhamos que uma transação T_i tenta executar um **escrever_item**(*X*)
  1. Se **read_TS**(*X*) > TS(T_i) ou se write_TS(X) > TS(T_i), então T_i aborta e reverte, e a operação é rejeitada. Uma operação mais nova que T_i já leu ou escreveu o valor do item X antes de T_i, violando a ordenação por timestamp.
  2. Senão, T_i executa a operação **escrever_item**(*X*) e grava write_TS(X) com TS(T_i).

O algoritmo de ordenação por Timestamp (Cont.)

- Suponhamos que uma transação T_i tenta executar um **ler_item**(X).
  1. Se write_TS(X) > TS(T_i), então T_i aborta, reverte e rejeita a operação. Isso ocorre porque alguma transação mais nova, já teria escrito o valor no item X antes que T_i tivesse a chance de ler X.
  2. Se write_TS(X) ≤ TS(T_i), então executa a operação de ler_item(X) de $T_{i,}$, e atualiza read_TS(X) com max(TS(T_i), read_TS(X)).

Técnicas de Controle de Concorrência de Multiversão

- Armazenam valores antigos dos itens atualizados. Uma transação acessa uma versão apropriada do item com o objetivo de manter, se possível a serialização.
- A idéia é que algumas operações de leitura que foram rejeitadas por outras técnicas, possam ser aceitas acessando uma versão mais antiga do item, garantindo a serialização.
- Uma desvantagem é o espaço necessário para armazenar as múltiplas versões. Entretanto isto é já feito para a recuperação e em BDs temporais.

Técnica de multiversão Baseada em ordenação por Timestamp.

- Neste método, diversas versões X1, X2,...,Xk de cada item de dado X são armazenadas. Para cada versão de X, os dois timestamps são mantidos: read_TS(Xi) e write_TS(Xi).
- Se uma transação T pode executar uma operação escrever_item(X), uma nova versão Xk+1 do X será criada com read_TS(Xk+1) e write_TS(Xk+1), atualizadas para TS(T).
- Quando T pode ler o valor da versão Xi, o valor de read_TS(Xi) será atualizado para max(read_TS(Xi), TS(T))

Técnica de multiversão Baseada em ordenação por Timestamp

- Suponhamos que a transação T_i executa uma operação ler_item(X) ou escrever_item(X). Seja X_k a versão de X cujo write_TS(X) é o mais alto de todas as versões de X, que também é menor ou igual TS(T_i).
  1. Se a transação T_i executa um **ler_item**(X), então o valor retornado é o conteúdo da versão X_k.
  2. Se a transação T_i executa um **escrever_item**(X)
     1. Se TS(T_i) < read_TS(X_k), então a transação T_i é abortada.
     2. Se TS(T_i) = write_TS(X_k), o conteúdo de X_k é sobre-escrito
     3. Senão uma nova versão de X é criada.
- Observe que
  - A leitura sempre tem sucesso
  - Uma escrita de T_i é rejeitada se alguma outra transação T_j que (na ordem de serialização definida pelos valores de timestamp) deveria ler o escrito po T_i, tem já lido a versão já criada por uma transação mais velha que T_i.
- O Protocolo garante serializabilidade

Bloqueio em Duas fases de Multiversão Usando Bloqueios de Certificação

- Há 3 modos de bloqueio: ler, gravar e certificar. LOCK(X) = read_locked ou write_locked ou certify_locked ou unlocked.
- Fig. 18.6a. → tabela de compatibilidade de bloqueio padrão.
- A idéia deste protocolo é permitir que outras transações T´ leiam um item X enquanto uma transação T mantem um bloqueio de escrita sobre X.

Bloqueio em Duas fases de Multiversão Usando Bloqueios de Certificação

- Isto é possível utilizando-se duas versões para cada item X; uma versão deve sempre ser gravada por alguma transação efetivada. A segunda versão X´ é criada quando uma transação T adquire um bloqueio em um item.
- Outras transações podem continuar lendo a versão efetivada de X, enquanto T controla o bloqueio de escrita em X´ sem afetar a versão efetivada.
- Uma vez T esteja pronta para efetivar, ela deve obter um bloqueio de certificação de todos os itens bloqueados para gravação.
- O bloqueio de certificação não é compatível com os bloqueios de leitura, assim, a transação pode atrasar a sua efetivação até que todos os itens de escrita bloqueados sejam liberados → para obter os certificados de bloqueio.

Bloqueio em Duas fases de Multiversão Usando Bloqueios de Certificação

- Uma vez os certificados são adquiridos, a versão X efetivada é atribuído o valor X´, que é descartada e os bloqueios de certificação são liberados. Veja a nova tabela de compatibilidade de bloqueio → Fig. 18.6b.

Figura 18.6 Tabelas de compatibilidade de bloqueio. (a) Uma tabela de compatibilidade para o esquema de bloqueio read/write. (b) Uma tabela de compatibilidade para o esquema de bloqueio ler/escrever/certificar.

(a)

	Ler	Escrever
Ler	sim	não
Escrever	não	não

(b)

	Ler	Escrever	Certificação
Ler	sim	sim	não
Escrever	sim	não	não
Certificação	não	não	não

Granularidade

- Um item de BD pode ser:
  - Um registro.
  - Um valor de um campo de um registro.
  - Um bloco de disco.
  - Um arquivo.
  - Um banco de dados inteiro.
- A granularidade pode afetar a execução do controle de concorrência e recuperação.

Considerações sobre Níveis de Granularidade para Bloqueio

- O tamanho dos itens de = granularidade.
- Quanto maior o tamanho do item de dados mais baixo é o grau de concorrência permitido
- Porém, quanto menor o tamanho do item de dados, maior será o número de itens em um BD → maior número de bloqueios ativos, mais operações de bloqueio e desbloqueio serão executadas: maior sobrecarga do sistema.
- ∴Qual é o melhor tamanho do item?
- Depende dos tipos de transações envolvidas. Se uma transação típica acessa muitos registros em um mesmo arquivo → granularidade maior.

Bloqueio de Nível de Granularidade Múltipla

- Já que o melhor tamanho de granularidade depende da transação dada ➡ múltiplos níveis de granularidades.
- Veja fig. 18.7. Suporte ao protocolo 2PL de nível de granularidades múltipla. Ex: T1 bloqueando f1 e T2 r1nj. Tentar bloquear T1 complicado.
- Para tornar prático este protocolo ➡ bloqueios de intenção.
- A idéia é que a transação indique, ao longo do caminho da raíz ao nó desejado, qual tipo de bloqueio ele irá solicitar.

Bloqueio de Nível de Granularidade Múltipla

- Três tipos de bloqueio:
  - Intenção-compartilhada (**IS**) indica que bloqueio(s) compartlhado(s) será(ão) solicitado(s) em algum(ns) nó(s) descendentes(s)
  - Intenção-exclusíva (**IX**) indica que bloqueio(s) exclusívo(s) será(ão) solicitado(s) em algum(ns) nó(s) descendentes(s)
  - Intenção-compartilhada-exclusíva (**SIX**) indica que o nó corrente está bloqueado compartilhado, mas bloqueio(s) exclusivo(s) será(ão) solicitado(s) em algum(ns) nó(s) descendentes(s)

Bloqueio de Nível de Granularidade Múltipla

- Veja a tabela de compatibilidade dos bloqueios de intenção. Fig. 18.8
- O protocolo de bloqueio de granularidade múltipla (MGL) consiste das seguintes regras:
  1. Cumprir a compatibilidade definida na tabela 18.8
  2. A raíz da árvore deve ser bloqueada primeiro (qualquer modo)
  3. Um nó pode ser bloqueado no modo S ou IS por uma transação T, só se o pai estiver bloqueado pela transação T no modo IS ou IX.
  4. Um nó pode ser bloqueado no modo X, IX ou SIX por uma transação T, só se o pai estiver bloqueado pela transação T no modo IX ou SIX.
  5. Uma transação pode bloquear um nó apenas se ela não tiver nenhum nó desbloqueado (2PL)
  6. Uma transação T pode bloquear um nó apenas se nenhum dos seus nós filhos estiverem correntemente bloqueados por T.

Figura 18.7 Uma hierarquia de granularidade para ilustrar bloqueio de nível de granularidade múltipla.

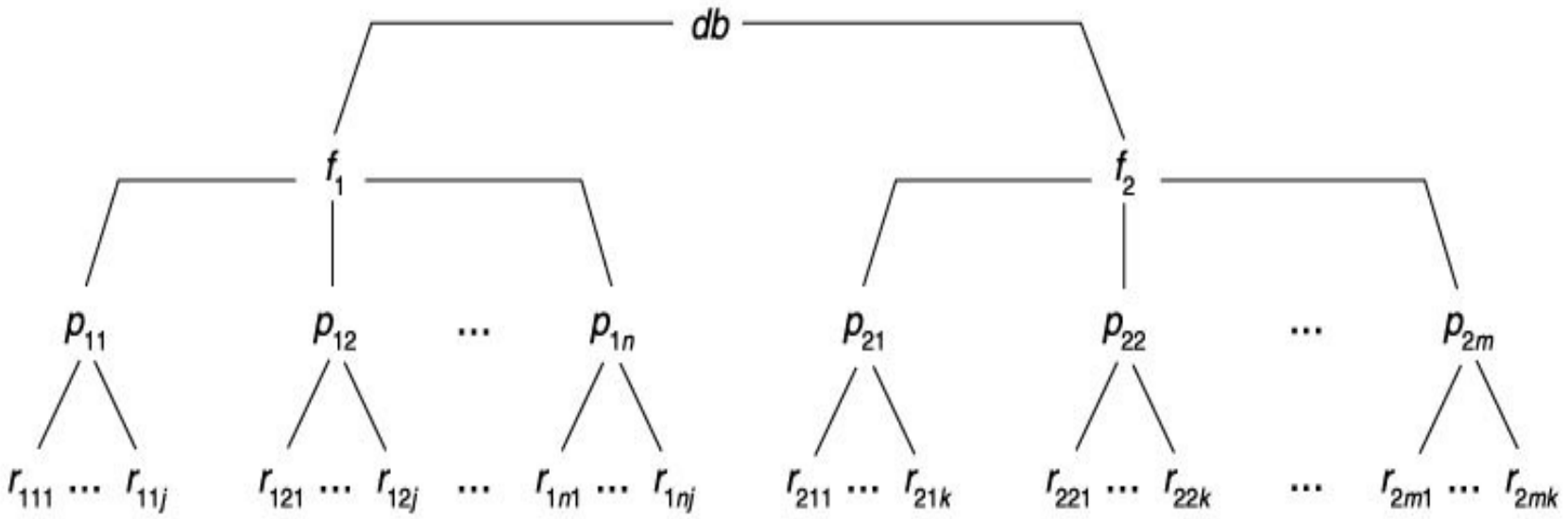


Figura 18.8 Uma matriz de compatibilidade de bloqueio para bloqueio de granularidade múltipla.

	IS	IX	S	SIX	X
IS	sim	sim	sim	sim	não
IX	sim	sim	não	não	não
S	sim	não	sim	não	não
SIX	sim	não	não	não	não
X	não	não	não	não	não

Figura 10 Operações de bloqueio para ilustrar um plano de execução serializável.

T_1	T_2	T_3
IX(db)		
IX(f_1)		
	IX(db)	
		IS(db)
		IS(f_1)
		IS(p_{11})
IX(p_{11})		
X(r_{111})		
	IX(f_1)	
	X(p_{12})	
		S(r_{11j})
IX(f_2)		
IX(p_{21})		
X(r_{211})		
unlock(r_{211})		
unlock(p_{21})		
unlock(f_2)		
		S(f_2)
	unlock(p_{12})	
	unlock(f_1)	
	unlock(db)	
unlock(r_{111})		
unlock(p_{11})		
unlock(f_1)		
unlock(db)		
		unlock(r_{11j})
		unlock(p_{11})
		unlock(f_1)
		unlock(f_2)
		unlock(db)

T1 quer atualizar os registros r111 até r211;

T2 quer atualizar todos os registros na página p12; e

T3 quer ler o registro r11j e o arquivo f2.

Este protocolo é útil quando existem uma mistura de transações: longas e curtas.

Usando Bloqueios para Controle de Concorrência em Índices

- O bloqueio de duas fases pode ser aplicado para os índices. Cuidado, muitos bloqueios, porque a pesquisa de um índice começa pela raiz, então se uma T quiser inserir um registro, a raiz deveria ser bloqueada em modo X.
- Uma abordagem conservadora, seria bloquear o nó raiz em modo X e, então, acessar o nó filho apropriado. Se o nó não estiver cheio, então desbloqueio a raiz, e assim por diante.