[MAC0426] Sistemas de Bancos de Dados
[IBI5013] Bancos de Dados para Bioinformática
Aula 23
Técnicas de Controle de Concorrência

Kelly Rosa Braghetto

DCC–IME–USP

22 de junho de 2017

Técnicas de controle de concorrência

- São usadas para **garantir a propriedade de isolamento** (não interferência) de transações executadas simultaneamente
- Em geral, garantem a seriação de escalonamentos usando **protocolos de controle de concorrência**
  - protocolo = conjunto de regras

Tipos de protocolos mais usados:

- **Protocolos de bloqueio bifásico** (usados na maioria dos SGBDs)
- Protocolos de ordenação de rótulo de tempo
- Protocolos multiversão
- Protocolos otimistas

Técnicas de bloqueio

- As principais técnicas de controle de concorrência de transações se baseiam em **bloqueio de itens de dados**
- Bloqueios são usados para **sincronizar o acesso** de transações concorrentes aos itens do BD

Bloqueio

- É uma variável associada a um item de dado do BD
- Descreve o status do item em relação a possíveis operações que podem ser aplicadas sobre ele

Tipos de bloqueios

- Bloqueios binários
- Bloqueios de leitura/gravação
- Bloqueios de certificação (não veremos neste curso)

Tipos de bloqueios

Bloqueio binário

- Um bloqueio distinto é associado a cada item de dado X do BD
- Cada bloqueio pode assumir um dos seguintes estados: **bloqueado (1)** ou **desbloqueado** (0)
- Notação: **lock(X)** = estado atual do bloqueio associado a X
  - Se lock(X) = 1, então X não pode ser acessado por uma operação que requisita o item
  - Se lock(X) = 0, então X pode ser acessado

Tipos de bloqueios

Bloqueio binário

- Para acessar um item X, primeiro a transação precisa emitir uma operação **lock_item(X)**
  - Se lock(X) = 1, então a transação é forçada a esperar
  - Se lock(X) = 0, então a transação **bloqueia** o item (status do bloqueio passa a valer 1) e tem permissão para acessá-lo
- Ao terminar de usar um item X, a transação emite uma operação **unlock_item(X)**
  - A transação **desbloqueia** o item (status do bloqueio passa a valer 0), de modo que X possa ser acessado por outras transações

Impõe a **exclusão mútua no acesso ao item de dados.**

Bloqueio binário

Implementação de **lock_item**

```
1: função lock_item(X)
2:     se lock(X) = 0 então
3:         lock(X) ← 1
4:     senão
5:         Esperar até que lock(X) = 0 e
6:           o gerenciador de bloqueio "desperte" a transação
7:         Vá para a linha 2
8:     fim se
9: fim função
```

Essa operação deve ser implementada como uma unidade indivisível. Nenhuma intercalação deve ser permitida durante a execução da operação.

Bloqueio binário

Implementação de **unlock_item**

```
1: função unlock_item(X)
2:     lock(X) ← 0
3:     se alguma transação estiver esperando então
4:         "acorde" uma das transações em espera
5:     fim se
6: fim função
```

Essa operação deve ser implementada como uma unidade indivisível. Nenhuma intercalação deve ser permitida durante a execução da operação.

Tipos de bloqueio

Bloqueios de leitura/gravação (ou compartilhados/exclusivos)

- Podemos permitir que várias transações acessem um mesmo item X **se todas acessarem X apenas para leitura**
- Mas se uma transação tiver que gravar em X, ela precisa ter acesso exclusivo a X
- Esse controle pode ser feito por meio de um bloqueio em X (lock(x)) com três estados possíveis:
  - **bloqueado para leitura** – que permite que **uma ou mais** transações acessem o item
  - **bloqueado para gravação** – que permite que **apenas uma** transação acesse o item
  - desbloqueado
- Agora, são necessárias três operações de bloqueio: **read_lock(X)**, **write_lock(X)** e **unlock(X)**

Bloqueio de leitura/gravação

Implementação de **read_lock**

```
 1: função read_lock(X)
 2:     se lock(X) = "desbloqueado" então
 3:         lock(X) ← "bloqueado para leitura"
 4:         num_de_leituras(X) ← 1
 5:     senão
 6:         se lock(X) = "bloqueado para leitura" então
 7:             num_de_leituras(X) ← num_de_leituras(X) + 1
 8:         senão
 9:             Esperar até que lock(X) = "desbloqueado" e
10:                 o gerenciador de bloqueio "desperte" a transação
11:             Vá para a linha 2
12:         fim se
13:     fim se
14: fim função
```

Bloqueio de leitura/gravação

Implementação de **write_lock**

```
1: função write_lock(X)
2:     se lock(X) = "desbloqueado" então
3:         lock(X) ← "bloqueado para escrita"
4:     senão
5:         Esperar até que lock(X) = "desbloqueado" e
6:           o gerenciador de bloqueio "desperte" a transação
7:         Vá para a linha 2
8:     fim se
9: fim função
```

Essa operação deve ser implementada como uma unidade indivisível. Nenhuma intercalação deve ser permitida durante a execução da operação.

Bloqueio de leitura/gravação

Implementação de **unlock**

```
1:  função unlock(X)
2:      se lock(X) = "bloqueado para escrita" então
3:          lock(X) ← "desbloqueado"
4:          "acorde" uma transação em espera, se houver
5:      senão
6:          se lock(X) = "bloqueado para leitura" então
7:              num_de_leituras(X) ← num_de_leituras(X) - 1
8:              se num_de_leituras(X) = 0 então
9:                  lock(X) ← "desbloqueado"
10:                 "acorde" uma transação em espera, se houver
11:             fim se
12:         fim se
13:     fim se
14: fim função
```

Bloqueios e seriação de escalonamentos

O uso de bloqueios binários ou de leitura/gravação não garantem por si só a seriação dos escalonamentos.

Exemplo

- Nas transações abaixo, as regras de bloqueio foram seguidas, mas os itens Y em T_1 e X em T_2 foram desbloqueados muito cedo. Isso permite que um escalonamento não seriável (como o mostrado no slide seguinte) ocorra

(a)

T_1	T_2
read_lock(Y);	read_lock(X);
read_item(Y);	read_item(X);
unlock(Y);	unlock(X);
write_lock(X);	write_lock(Y);
read_item(X);	read_item(Y);
$X := X + Y$;	$Y := X + Y$;
write_item(X);	write_item(Y);
unlock(X);	unlock(Y);

(b) Valores iniciais: X= 20, Y=30

Schedule serial resultante T_1 seguido por T_2: X=50, Y=80

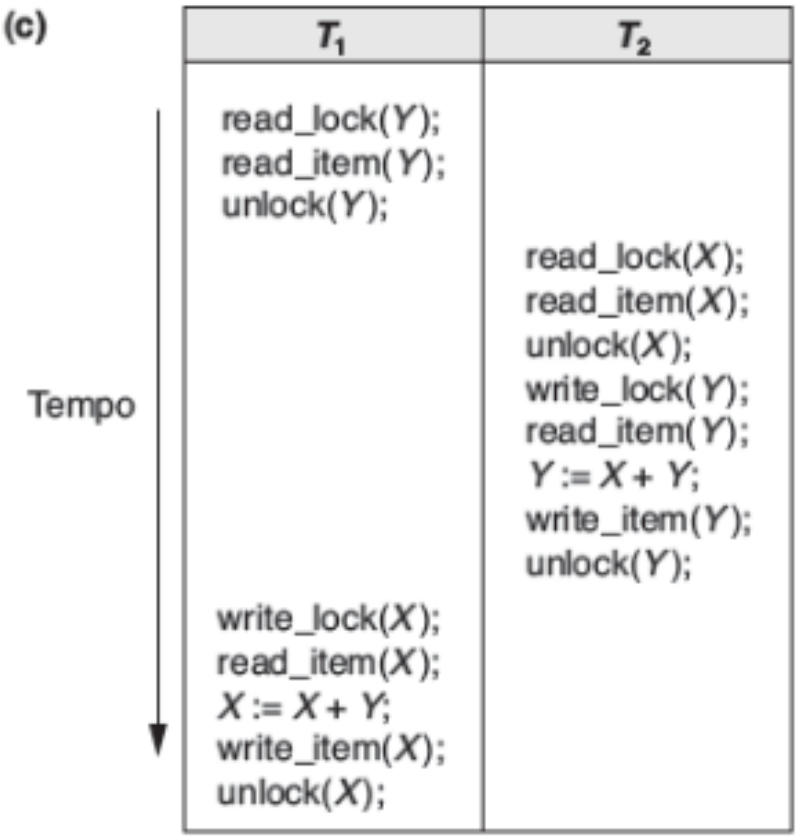
Schedule serial resultante T_2 seguido por T_1: X=70, Y=50

Bloqueios e seriação de escalonamentos

Continuação do exemplo

- Exemplo de escalonamento não seriável das transações do slide anterior

(c)



Resultado de schedule S:
X=50, Y=50
(não serializável)

Garantindo a seriação pelo bloqueio bifásico

Protocolo de bloqueio bifásico

- Uma transação segue o **protocolo de bloqueio bifásico se todas as operações de bloqueio** (read_lock and write_lock) **precedem a primeria de desbloqueio** (unlock) na transação
- Duas fases na transação:
  1. **fase de expansão** ou **crescimento** – quando novos bloqueios podem ser adquiridos, mas nenhum pode ser liberado
  2. **fase de encolhimento** – quando os bloqueios existentes podem ser liberados, mas nenhum novo bloqueio pode ser adquirido

Protocolo de bloqueio bifásico

Exemplo de transações que não seguem o protocolo

T_1	T_2

read_lock(Y);
read_item(Y);
unlock(Y);
write_lock(X);
read_item(X);
X := X + Y;
write_item(X);
unlock(X);
	read_lock(X);
	read_item(X);
	unlock(X);
	write_lock(Y);
	read_item(Y);
	Y := X + Y;
	write_item(Y);
	unlock(Y);

- Em T_1, a operação write_lock(X) segue a operação unlock(Y)
- Em T_2, a operação write_lock(Y) segue a operação unlock(X)

Protocolo de bloqueio bifásico

Modificando as transações para que sigam o protocolo

T_1	T_2

read_lock(Y);
read_item(Y);
unlock(Y);
write_lock(X);
read_item(X);
$X := X + Y$;
write_item(X);
unlock(X);
	read_lock(X);
	read_item(X);
	unlock(X);
	write_lock(Y);
	read_item(Y);
	$Y := X + Y$;
	write_item(Y);
	unlock(Y);

$\Rightarrow$

T_1'	T_2'

read_lock(Y);
read_item(Y);
write_lock(X);
unlock(Y)
read_item(X);
$X := X + Y$;
write_item(X);
unlock(X);
	read_lock(X);
	read_item(X);
	write_lock(Y);
	unlock(X)
	read_item(Y);
	$Y := X + Y$;
	write_item(Y);
	unlock(Y);

Protocolo de bloqueio bifásico

- Se cada transação em um escalonamento seguir o protocolo de bloqueio bifásico, **o escalonamento é garantidamente seriável**
  - Não é necessário testar a seriação do escalonamento!
  - As regras impostas pelo protocolo também impõem a seriação

Problemas desse procolo:

- Ele pode limitar a quantidade de concorrência passível de ocorrer em um escalonamento
  - Por causa do protocolo, nem sempre a transação pode liberar um item logo depois de usá-lo e às vezes ela precisa bloquear um item antes que precise dele, impedindo o acesso de outras transações
- Embora garanta a seriação, ele não permite todos os escalonamentos seriáveis possíveis – alguns serão proibidos pelo protocolo

Variações do bloqueio bifásico

- Bloqueio bifásico básico – o que acabamos de ver
- **Bloqueio bifásico conservador**, também chamado de **estático**
- **Bloqueio bifásico estrito**
- **Bloqueio bifásico rigoroso**

Variações do bloqueio bifásico

Bloqueio bifásico conservador

- Requer que uma transação **bloqueie todos os itens que ela acessa antes de começar sua execução**, pré-declarando seus conjuntos de leitura e de gravação
  - Se algum dos itens pré-declarados pela transação não puder ser bloqueado, nenhum dos itens será. A transação esperará até que todos os itens estejam disponíveis.
- É livre de *deadlock* (espera cíclica)
- É difícil de ser usado na prática, pela necessidade de pré-declarar os itens acessados (nem sempre isso é possível)

Variações do bloqueio bifásico

Bloqueio bifásico estrito

- Variação mais popular de bloqueio bifásico
- Garante escalonamentos estritos, que são de fácil recuperação
- Requer que uma transação T **não libere nenhum de seus bloqueios de gravação até depois de confirmar ou abortar**
  - Nenhuma outra transação pode acessar um item gravado por T até que T termine
- Não é livre de *deadlock* (impasse)

O próprio subsistema de controle de concorrência do SGBD pode se responsabilizar por gerar as solicitações read_lock e write_lock.

Variações do bloqueio bifásico

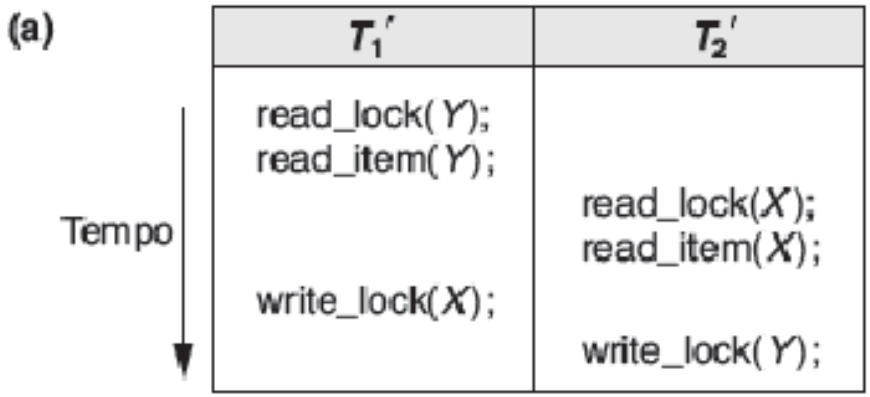
Bloqueio bifásico rigoroso

- Variação mais rigorosa do bloqueio bifásico estrito
- Também garante escalonamentos estritos
- Requer que uma transação T **não libere nenhum de seus bloqueios (de gravação ou de leitura) até depois de confirmar ou abortar**
  - Nenhuma outra transação pode acessar um item gravado por T até que T termine
- Não é livre de *deadlock* (impasse)

Problemas (adicionais) do uso de bloqueios

Deadlock (impasse)

- Ocorre quando cada transação T em um conjunto de 2 ou mais transações está esperando por algum item que está bloqueado por alguma outra transação T' no conjunto.
- Maneiras de se lidar com ele: **prevenção**, **detecção** e ***timeouts***
- Exemplo em um escalonamento parcial



Exemplos de protocolos de prevenção de *deadlocks* não-práticos

- Protocolo 1, o mesmo usado no bloqueio conservador – cada transação deve bloquear, com antecedência, todos os itens que precisar
- Protocolo 2 – ordenar todos os itens no BD e garantir que uma transação que precisa de vários itens os bloqueará de acordo com essa ordem
  - Além de limitar a concorrência, como o anterior, esse protocolo requer que o programador ou o sistema conheça a ordem dos itens

Exemplo 1 de protocolo de prevenção de *deadlocks* prático

Usando **rótulo de tempo** (*timestamp*) da transação

- O **rótulo de tempo** $TS(T)$ de uma transação T é um identificador exclusivo atribuído à T com base no momento em que ela foi iniciada
- Se $TS(T_1) < TS(T_2)$, então T_1 é mais antiga que T_2 (ou seja, começou antes)
- Dois esquemas diferentes usam o rótulo de tempo para decidir qual transação deve ser abortada em um caso de impasse:
  - Esperar-morrer (*wait-die*)
  - Ferir-esperar (*wound-waid*)

Exemplo 1 de protocolo de prevenção de *deadlocks* prático

Esperar-morrer

Suponha que a transação T_i tente bloquear um item X mas não consiga, porque ele já está bloqueado para a transação T_j.

- No esquema **esperar-morrer**, se $TS(T_i) < TS(T_j)$ então T_i tem permissão para esperar; caso contrário, aborta T_i e o reinicia mais tarde *com o mesmo rótulo de tempo*.
- Ou seja, uma transação mais antiga tem permissão para esperar por uma transação mais nova, enquanto uma transação mais nova que solicita um item mantido por uma mais antiga é abortada e reiniciada.

Exemplo 1 de protocolo de prevenção de *deadlocks* prático

Ferir-esperar

Suponha que a transação T_i tente bloquear um item X mas não consiga, porque ele já está bloqueado para a transação T_j.

- No esquema **ferir-esperar**, se $TS(T_i) < TS(T_j)$ então aborta T_j e o reinicia mais tarde *com o mesmo rótulo de tempo*; caso contrário, T_i tem permissão para esperar.
- Ou seja, uma transação mais nova tem permissão para esperar por uma mais antiga, enquanto uma transação mais antiga que solicita um item mantido por uma mais nova apodera-se da mais nova ao abortá-la.

Os dois esquemas acabam abortando a transação mais nova (em execução a menos tempo), supondo que isso desperdiçará menos processamento.

Exemplo 1 de protocolo de prevenção de *deadlocks* prático

Usando **rótulo de tempo** (*timestamp*) da transação

- **Esperar-morrer** é livre de de *deadlocks* porque nela uma transação somente espera por outra transação mais nova que ela $\rightarrow$ não há formação de ciclos!
- De modo análogo, **ferir-esperar** é livre de de *deadlocks* porque nela uma transação somente espera por outra transação mais antiga
- Mas em ambos esquemas, transações podem ser abortadas e reiniciadas sem necessidade

Exemplo 2 de protocolo de prevenção de *deadlocks* prático

Sem uso de **rótulo de tempo** da transação

- **Sem espera** – se uma transação for incapaz de obter um bloqueio, ela é imediatamente abortada e reiniciada depois de um tempo
  - transações podem ser abortadas e reiniciadas sem necessidade
- **Espera cuidadosa** – Suponha que T_i tente bloquear (sem sucesso) um item X que já está bloqueado por T_j. As regras da espera cuidadosa são:
  - Se T_j não estiver bloqueada (esperando por outro item bloqueado), então T_i está bloqueada e tem permissão para esperar.
  - Senão, aborte T_i.

A espera cuidadosa é livre de *deadlocks* porque nenhum transação espera por outra que está bloqueada.

Detecção de *deadlock*

- Na detecção, o sistema verifica realmente se o *deadlock* existe
- Essa solução só é viável quando sabe-se que há pouca interferência entre as transações. No caso contrário, é mais vantajoso usar um esquema de prevenção.
- Para a detecção, o sistema constrói e mantém um **grafo de espera**:
  - Cada nó do grafo representa uma transação em execução
  - Há uma aresta de T_i para T_j no grafo sempre que a transação T_i estiver esperando para bloquear um item que atualmente está bloqueado por T_j
  - Há um estado de *deadlock* se e somente se o grafo de espera tiver um ciclo

Detecção de *deadlock*

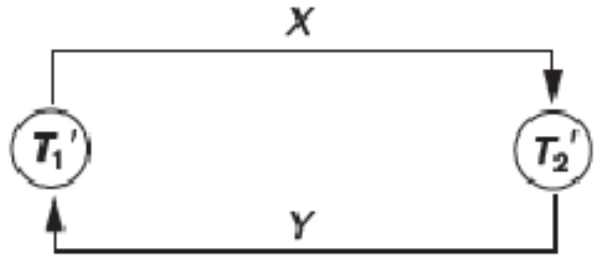
Exemplo de grafo de espera

(a)

T_1'	T_2'
read_lock(Y); read_item(Y);	
	read_lock(X); read_item(X);
write_lock(X);	
	write_lock(Y);

Tempo

(b)



Detecção de *deadlock*

Quando o sistema deve procurar um *deadlock*?

Possibilidades:

- A cada vez que uma aresta for adicionada ao grafo de espera (pode causar muito *overhead*!)
- Algum critério envolvendo o número de transações em execução
- Algum critério envolvendo o tempo de espera das transações para bloquear itens

Detecção de *deadlock*

Quando o *deadlock* é detectado, o que o sistema deve fazer?

- Selecionar "vítimas" – transações envolvidas no impasse que serão abortadas
- O algoritmo de seleção deve tentar selecionar as transações mais novas, pois essas não fizeram ainda muitas modificações e por isso têm menor custo de recuperação

Timeouts como tratamento para *deadlocks*

- Usar *timeouts* é um esquema prático (simples e de baixo custo) para tratar *deadlocks*
- Se uma transação esperar por um tempo maior que o limite (*timeout*) definido, o sistema pressupõe que a transação pode entrar em *deadlock* e a aborta
  - Note que a decisão independe do fato do *deadlock* existir de verdade ou não

Problemas (adicionais) do uso de bloqueios

Inanição (*starvation*)

- Ocorre quando uma transação não pode prosseguir por um período indefinido de tempo, enquanto outras continuam normalmente.
- Causa 1: esquema de espera para itens bloqueados injusto
- Possíveis soluções:
  - uso de fila (primeiro a chegar, primeiro a sair) – as transações são habilitadas para bloquear um item de acordo com ordem da chegada das solicitações de acesso
  - uso de prioridade – a prioridade de uma transação ser habilitada para bloquear o item aumenta proporcionalmente ao tempo de espera dela pelo item

Problemas (adicionais) do uso de bloqueios

Inanição (*starvation*)

- Causa 2: transação é repetidamente abortada (por conta da estratégia de escolha da "vítima")
- Possível solução:
  - atribuir prioridades maiores às transações que tiveram esse problema

Outras técnicas de controle de concorrência

Controle de concorrência multiversão

- Essa técnica mantém os valores antigos de um item de dados quando esse é atualizado
  - cada nova gravação gera uma nova **versão** do item
- Quando uma transação requer acesso a um item, uma versão dele apropriada é escolhida para manter a seriação do escalonamento em execução (se possível)
- Assim, algumas operações de leitura que poderiam ser rejeitadas em outras técnicas, nessa podem ser aceitas ao ler uma versão mais antiga do item

Outras técnicas de controle de concorrência

Controle de concorrência otimista

- Nessa técnica, diferentemente do que ocorre no bloqueio bifásico, nenhuma verificação é feita enquanto a transação está executando
- As atualizações feitas pela transação não são aplicadas diretamente aos itens do BD até que a transação termine
  - As atualizações são aplicadas em cópias locais dos itens mantidas para a transação
- Ao final da transação, uma fase de validação verifica se as atualizações violaram a seriação
  - Em caso de violação, a transação é abortada e reiniciada posteriormente
  - No caso contrário, a transação é confirmada e o BD é atualizado

Otimismo: assumir que há pouca interferência entre as transações!

Outras técnicas de controle de concorrência

Controle de concorrência baseado na ordenação de rótulos de tempo (*timestamps*)

- Essa técnica se baseia na ideia de ordenar a execução das transações conforme um escalonamento serial equivalente
  - não usa bloqueios, portanto é livre de *deadlocks*
  - transações são ordenadas de acordo com o seu rótulo de tempo
  - garante que, para cada item acessado pelas operações em conflito no escalonamento, a ordem em que o item é acessado não viola a ordem dos rótulos de tempo das transações
  - o rótulo de tempo é um identificador exclusivo para a transação -> pode ser entendido como o seu horário de início

Referências Bibliográficas

- *Sistemas de Bancos de Dados* (6ª edição), Elmasri e Navathe. Pearson, 2010. – Capítulo 22