Técnicas de Controle de Concorrência

- Pessimistas
  - supõem que sempre ocorre interferência entre transações e garantem a serializabilidade enquanto a transação está ativa
  - técnicas
    - bloqueio (*locking*)
    - *timestamp*
- Otimistas
  - supõem que quase nunca ocorre interferência entre transações e verificam a serializabilidade somente ao final de uma transação
  - técnica
    - validação

Técnicas Baseadas em Bloqueio

- Técnicas mais utilizadas pelos SGBDs
- Princípio de funcionamento
  - controle de operações *read(X)* e *write(X)* e postergação (através de bloqueio) da execução de algumas dessas operações de modo a evitar conflito
- Todo dado possui um status de bloqueio
  - liberado (*Unlocked* - U)
  - com bloqueio compartilhado (*Shared lock* - S)
  - com bloqueio exclusivo (*eXclusive lock* - X)

Modos de Bloqueio

- Bloqueio Compartilhado (S)
  - solicitado por uma transação que deseja realizar leitura de um dado D
    - várias transações podem manter esse bloqueio sobre D
- Bloqueio Exclusivo (X)
  - solicitado por uma transação que deseja realizar atualização ou leitura+atualização de um dado D
    - uma única transação pode manter esse bloqueio sobre D
- Matriz de Compatibilidade de Bloqueios

	S	X
S	verdadeiro	falso
X	falso	falso

- Informações de bloqueio são mantidas no DD
  `<ID-dado,status-bloqueio,ID-transação>`

Operações de Bloqueio na História

- O *Scheduler* gerencia bloqueios através da invocação automática de operações de bloqueio conforme a operação que a transação deseja realizar em um dado
- Operações
  - *ls(D)*: solicitação de bloqueio compartilhado sobre D
  - *lx(D)*: solicitação de bloqueio exclusivo sobre D
  - *u(D)*: libera o bloqueio atual sobre D

Exemplo de História com Bloqueios

T1	T2
lock-S(Y)	
read(Y)	
unlock(Y)	
	lock-S(X)
	lock-X(Y)
	read(X)
	read(Y)
	unlock(X)
	write(Y)
	unlock(Y)
	commit( )
lock-X(X)	
read(X)	
write(X)	
unlock(X)	
commit( )	

H = ls1(Y) r1(Y) u1(Y) ls2(X) lx2(Y)
r2(X) r2(Y) u2(X) w2(Y) u2(Y) c2
lx1(X) r1(X) w1(X) u1(X) c1

Implementação das Operações

- Solicitação de bloqueio compartilhado

```
lock-S(D, Tx)                                   → status de bloqueio de D
início
        se lock(D) = 'U' então
        início
                insere Tx na lista-READ(D);     → lista de transações com bloqueio compartilhado sobre D
                lock(D) ← 'S';
        fim
        senão se lock(D) = 'S' então insere Tx na lista-READ(D)
                senão /* lock(D) = 'X' */ insere (Tx, 'S') na fila-WAIT(D);
fim                                             → fila de transações aguardando a liberação de um bloqueio conflitante sobre D
```

Obs.: supor que os métodos de inclusão/exclusão de elementos nas EDs automaticamente alocam/desalocam a ED caso ela não exista/se torne vazia

Exercício 2

1. Propor algoritmos de alto nível para as operações:
   a) lock-X(D, Tx)
   b) unlock(D, Tx) (considere que essa operação também pode retirar transações da fila-WAIT e solicitar novos bloqueios)
2. O algoritmo lock-S(D, Tx) apresentado anteriormente pode gerar *starvation* (espera indefinida de Tx, se Tx solicitou lock-X(D, Tx) e lista-READ(D) nunca fica vazia!). Modifique os algoritmos das operações de bloqueio (aqueles que forem necessários) de modo a evitar *starvation*

Uso de Bloqueios "S" e "X"

- Não garantem escalonamentos serializáveis
- Exemplo

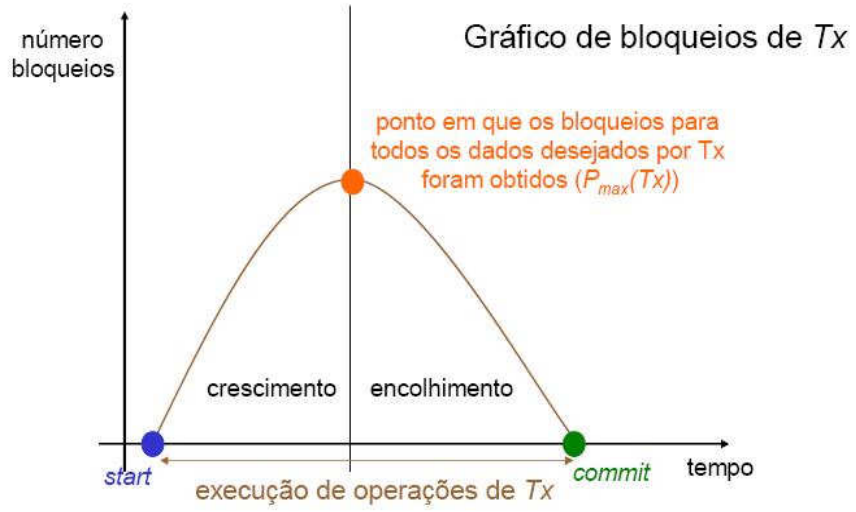
  $H_{N\text{-}SR}$= ls1(Y) r1(Y) u1(Y) ls2(X) r2(X) u2(X) lx2(Y) r2(Y)
  lx1(X) r1(X) w1(X) w2(Y) u2(Y) c2 u1(X) c1

- Necessita-se de uma técnica mais rigorosa de bloqueio para garantir a serializabilidade
  – técnica mais utilizada
    - bloqueio de duas fases (*two-phase locking* – 2PL)

Bloqueio de 2 Fases – 2PL

- Premissa
  - *"para toda transação Tx, todas as operações de bloqueio de dados feitas por Tx precedem a primeira operação de desbloqueio feita por Tx"*
- Protocolo de duas fases
  1. Fase de expansão ou crescimento
     - *Tx* pode obter bloqueios, mas não pode liberar nenhum bloqueio
  2. Fase de retrocesso ou encolhimento
     - *Tx* pode liberar bloqueios, mas não pode obter nenhum bloqueio

Scheduler 2 PL – Funcionamento

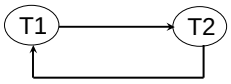


Scheduler 2PL - Exemplo

- T1: r(Y) w(Y) w(Z)
- T2: r(X) r(Y) w(Y) r(Z) w(Z)
- Contra-Exemplo

não é 2PL!

$H_{N\text{-}2PL}$ = lx1(Y) r1(Y) ls2(X) r2(X) u2(X) w1(Y) u1(Y) lx2(Y) r2(Y) w2(Y) u2(Y) lx2(Z) r2(Z) w2(Z) c2 lx1(Z) w1(Z) u1(Z) c1



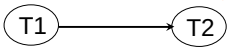
não garantiu SR!

- Exemplo

$P_{max}(T1)$

H_{2PL} = ls2(X) r2(X) lx1(Y) r1(Y) lx1(Z) w1(Y) u1(Y) lx2(Y) r2(Y) w1(Z) u1(Z) c1 w2(Y) lx2(Z) u2(X) u2(Y) w2(Z) u2(Z) c2

é SR!



$P_{max}(T2)$

Scheduler 2PL - Crítica

- Vantagem
  - técnica que sempre garante escalonamentos SR sem a necessidade de se construir um grafo de dependência para teste!
    - se *Tx* alcança P_{max}, *Tx* não sofre interferência de outra transação *Ty*, pois se *Ty* deseja um dado de *Tx* em uma operação que poderia gerar conflito com *Tx*, *Ty* tem que esperar (teremos aresta $Tx \rightarrow Ty$, mas não teremos aresta $Ty \rightarrow Tx$!)
    - depois que *Tx* liberar os seus dados, não precisará mais deles, ou seja, *Tx* não interferirá nas operações feitas futuramente nestes dados por *Ty* (também evita aresta $Ty \rightarrow Tx$!)

Scheduler 2PL - Crítica

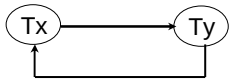
- Desvantagens
  - limita a concorrência
    - um dado pode permanecer bloqueado por *Tx* muito tempo até que *Tx* adquira bloqueios em todos os outros dados que deseja
  - 2PL básico (técnica apresentada anteriormente) não garante escalonamentos
    - livres de *deadlock*
      - *Tx* espera pela liberação de um dado bloqueado por *Ty* de forma conflitante e vice-versa
    - adequados à recuperação pelo *recovery*

Exercício 3

1. Apresente um início de escalonamento concorrente 2PL básico que recaia em uma situação de *deadlock*
2. Apresente um escalonamento concorrente 2PL básico que não seja recuperável
   - lembrete: um escalonamento é recuperável se *Tx* nunca executa *commit* antes de *Ty*, caso *Tx* tenha lido dados atualizados por *Ty*

Deadlock (Impasse) de Transações

- Ocorrência de *deadlock*
  - *Ty* está na Fila-WAIT(D1) de um dado D1 bloqueado por *Tx*
  - *Tx* está na Fila-WAIT(D2) de um dado D2 bloqueado por *Ty*
- Pode ser descoberto através de um grafo de espera de transações
  - se o grafo é cíclico existe *deadlock*!

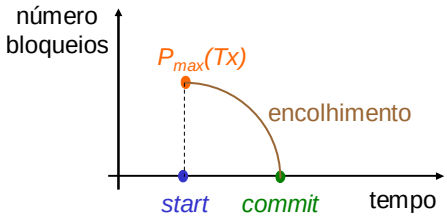


Tratamento de *Deadlock*

- Protocolos de Prevenção
  - abordagens pessimistas
    - *deadlocks* ocorrem com freqüência!
    - impõem um *overhead* no processamento de transações
      - controles adicionais para evitar *deadlock*
    - tipos de protolocos pessimistas
      - técnica de bloqueio 2PL conservador
      - técnicas baseadas em *timestamp* (*wait-die* e *wound-wait*)
      - técnica de *espera-cautelosa* (*cautious-waiting*)
  - uso de *timeout*
    - se tempo de espera de *Tx* > *timeout* $\Rightarrow$ *abort(Tx)*

Scheduler 2PL Conservador

- *Tx* deve bloquear todos os dados que deseja antes de iniciar qualquer operação
  - caso não seja possível bloquear todos os dados, nenhum bloqueio é feito e *Tx* entra em espera para tentar novamente



  - vantagem
    - uma vez adquiridos todos os seus bloqueios, *Tx* não entra em *deadlock* durante a sua execução
  - desvantagem
    - espera pela aquisição de todos os bloqueios!

Técnicas Baseadas em *Timestamp*

- *Timestamp*
  - rótulo de tempo associado à *Tx* (*TS(Tx)*)

tempo de start de Tx

- Técnicas
  - consideram que *Tx* deseja um dado bloqueado por outra transação *Ty*
  - Técnica 1: *esperar-ou-morrer* (*wait-die*)
    - se *TS(Tx)* < *TS(Ty)* então *Tx* espera
      senão início
        abort(Tx)
        start(Tx) com o mesmo TS
      fim

Técnicas Baseadas em *Timestamp*

- Técnicas (cont.)
  - Técnica 2: *ferir-ou-esperar* (*wound-wait*)
    - se *TS(Tx)* < *TS(Ty)* então
      início
        abort(Ty)
        start(Ty) com o mesmo TS
      fim
      senão *Tx* espera
  - vantagem das técnicas
    - evitam *starvation* (espera indefinida) de uma *Tx*
      - quanto mais antiga for *Tx*, maior a sua prioridade
  - desvantagem das técnicas
    - muitos abortos podem ser provocados, sem nunca ocorrer um *deadlock*

Técnica *Cautious-Waiting*

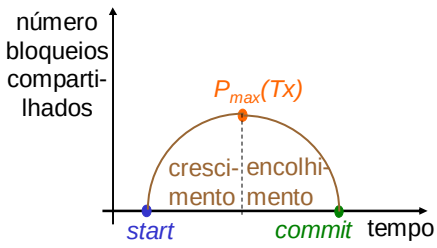
- Princípio de Funcionamento
  - se *Tx* deseja D e D está bloqueado por *Ty*
    então
      se *Ty* não está em alguma Fila-WAIT
      então *Tx* espera
      senão início
              abort(Tx)
              start(Tx)
          fim
- Vantagem
  - se *Ty* já está em espera, *Tx* é abortada para evitar um possível ciclo de espera
- Desvantagem
  - a mesma das técnicas baseadas em *timestamp*

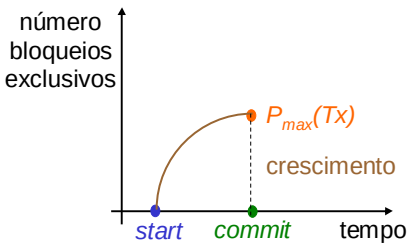
Tratamento de *Deadlock*

- Protocolos de Detecção
  - abordagens otimistas
    - *deadlocks* não ocorrem com freqüência!
      - são tratados quando ocorrem
    - mantém-se um grafo de espera de transações
    - se há *deadlock*, seleciona-se uma transação vítima *Tx* através de um ou mais critérios
      - quanto tempo *Tx* está em processamento
      - quantos itens de dado *Tx* já leu/escreveu
      - quantos itens de dado *Tx* ainda precisa ler/escrever
      - quantas outras transações serão afetadas pelo *abort*(*Tx*)

Outras Técnicas de Bloqueio 2PL

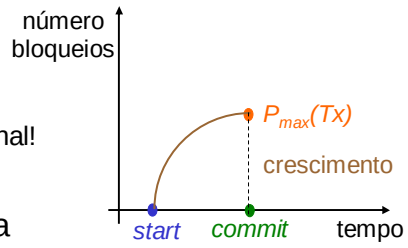
- *Scheduler* 2PL Conservador ou Estático
  - evita *deadlock*, porém *Tx* pode esperar muito para executar
- *Scheduler* 2PL Estrito (muito usado pelos SGBDs)
  - *Tx* só libera seus bloqueios exclusivos após executar *commit* ou *abort*





Outras Técnicas de Bloqueio 2PL

- *Scheduler* 2PL Estrito
  - vantagem: garante escalonamentos estritos
  - desvantagem: não está livre de *deadlocks*
- *Scheduler* 2PL (Estrito) Rigoroso
  - *Tx* só libera seus bloqueios após executar *commit* ou *abort*
  - vantagem
    - menos *overhead* para *Tx*
      - *Tx* libera tudo apenas no final!
  - desvantagem
    - limita mais a concorrência



Exercícios 4

1. Apresente exemplos de escalonamentos concorrentes 2PL conservador, 2PL estrito e 2PL rigoroso para as seguintes transações:

   T1: r(Y) w(Y) w(Z)

   T2: r(X) r(T) w(T)

   T3: r(Z) w(Z)

   T4: r(X) w(X)

2. Apresente uma situação de *deadlock* em um escalonamento concorrente 2PL estrito

Scheduler Baseado em *Timestamp*

- Técnica na qual toda transação *Tx* possui uma marca *timestamp* (*TS(Tx)*)
- Princípio de funcionamento (TS-Básico)
  - *"no acesso a um item de dado D por operações conflitantes, a ordem desse acesso deve ser equivalente à ordem de TS das transações envolvidas"*
    - garante escalonamentos serializáveis através da ordenação de operações conflitantes de acordo com os TSs das transações envolvidas
  - cada item de dado X possui um registro TS (*R-TS(X)*)

`<ID-dado, TS-Read, TS-Write>`

TS da transação mais recente que leu o dado

TS da transação mais recente que atualizou o dado

Técnica TS-Básico - Exemplo

- T1: r(Y) w(Y) w(Z) → TS(T1) = 1
- T2: r(X) r(Y) w(Y) r(Z) w(Z) → TS(T2) = 2
- Registros iniciais de TS de X, Y e Z:
  - `<X,0,0>; <Y,0,0>; <Z,0,0>`
- Exemplo de escalonamento serializável por TS

$H_{TS\text{-}B}$ = r2(X) r1(Y) w1(Y) r2(Y) w1(Z) c1 w2(Y) r2(Z) w2(Z) c2

<Z,0,1> (...) ⇒ <Z,2,2>

<Y,2,1> (...) ⇒ <Y,2,2>

<Z,0,0> (TS(T1) >= {TS-Read(Z),TS-Write(Z)} OK!) ⇒ <Z,0,1>

<Y,1,1> (TS(T2) >= TS-Write(Y) OK!) ⇒ <Y,2,1>

<Y,1,0> (TS(T1) >= {TS-Read(Y),TS-Write(Y)} OK!) ⇒ <Y,1,1>

<Y,0,0> (TS(T1) >= TS-Write(Y) OK!) ⇒ <Y,1,0>

<X,0,0> (TS(T2) >= TS-Write(X) OK!) ⇒ <X,2,0>

Algoritmo TS-Básico

```
TS-Básico(Tx, dado, operação)
início
  se operação = 'READ' então
    se TS(Tx) < R-TS(dado).TS-Write então
    início     abort(Tx);
               restart(Tx) com novo TS;
    fim
    senão início executar read(dado);
                 se R-TS(dado).TS-Read < TS(Tx) então
   /* mantém TS-Read sempre atualizado com a transação mais nova que leu,
      para garantir sempre um controle correto da ordenação */
                   R-TS(dado).TS-Read ← TS(Tx);
           fim
  senão início /* operação = 'WRITE' */
          se TS(Tx) < R-TS(dado).TS-Read OU
             TS(Tx) < R-TS(dado).TS-Write então
         início      abort(Tx);
                     restart(Tx) com novo TS;
          fim
          senão início       executar write(dado);
                             R-TS(dado).TS-Write ← TS(Tx);
                  fim
        fim
fim
```

Técnica TS-Básico

- Vantagens
  - técnica simples para garantia de serializabilidade (não requer bloqueios)
  - não há *deadlock* (não há espera)
- Desvantagens
  - gera muitos abortos de transações
    - passíveis de ocorrência quando há conflito
  - pode gerar abortos em cascata
    - não gera escalonamentos adequados ao *recovery*
- Para minimizar essas desvantagens
  - técnica de *timestamp* estrito (TS-Estrito)

Técnica TS-Estrito

- Gera escalonamentos serializáveis e estritos
  - passíveis de *recovery* em caso de falha
- Funcionamento
  - baseado no TS-básico com a seguinte diferença
    - "se *Tx* deseja *read(D)* ou *write(D)* e *TS(Tx) > R-TS(D).TS-Write*, então *Tx* espera pelo *commit* ou pelo *abort* da transação *Ty* cujo *R-TS(D).TS-Write = TS(Ty)*"
    - exige *fila-WAIT(D)*
    - não há risco de *deadlock*
      - nunca há ciclo pois somente transações mais novas esperam pelo *commit/abort* de transações mais antigas
    - *overhead* no processamento devido à espera

Técnica TS-Estrito - Exemplo

- T1: r(X) w(X) w(Z) → TS(T1) = 1
- T2: r(X) w(X) w(Y) → TS(T2) = 2
- Exemplo de escalonamento TS-Estrito

H_{TS-E} = r1(X) w1(X) r2(X) w1(Z) c1 r2(X) w2(X) w2(Y) c2

T2 espera por T1, pois *TS(T2) > R-TS(X).TS-write* (*r2(X)* não é executado e T2 é colocada na Fila-WAIT(X))

T1 já *committou*! T2 pode executar agora *r2(X)* (tira-se T2 da fila-WAIT(X))

Exercícios 5

1. Considerando a técnica TS-Básico, verifique se alguma transação abaixo é desfeita e em que ponto
   a) H1 = r1(a) r2(a) r3(a) c1 c2 c3
   b) H2 = r1(a) w2(a) r1(a) c1 c2
   c) H3 = r1(a) r1(b) r2(a) r2(b) w2(a) w2(b) c1 c2
   d) H4 = r1(a) r1(b) r2(a) w2(a) w1(b) c1 c2
   e) H5 = r2(a) w2(a) w1(a) r2(a) c1 c2
   f) H6 = r2(a) w2(a) r1(b) r1(c) w1(c) w2(b) c1 c2
2. Apresente o algoritmo *TS-Estrito(Tx, dado, operação).* Há algo a considerar nos algoritmos *Commit(Tx)* e *Abort(Tx)*?
3. Apresente um exemplo e um contra-exemplo de um escalonamento TS-Estrito para as seguintes transações:
   T1: r(Y) w(Y) w(Z)
   T2: r(X) r(T) w(T)
   T3: r(Z) w(Z)
   T4: r(X) w(X)