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)

Bloqueios e Granularidade

- Grânulo
  - porção do BD
    - atributo, tupla, tabela, bloco, ...
  - níveis de granularidade
    - granularidade fina
      - porção pequena do BD ⇒ muitos itens de dados
      - maior número de itens de dados a serem bloqueados e controlados pelo *scheduler*
      - maior concorrência
    - granularidade grossa
      - porção grande do BD ⇒ menos itens de dados
      - menor número de itens de dados a serem bloqueados e controlados pelo *scheduler*
      - menor concorrência

Bloqueios e Granularidade

- Na prática, transações podem realizar bloqueios em vários níveis de granularidade
  - *Tx* atualiza uma tupla; *Ty* atualiza toda uma tabela
- Algumas questões
  - se *Ty* quer atualizar toda uma tabela, *Ty* deve bloquear TODAS as tuplas?
  - se *Tx* bloqueou uma tupla da tabela *T* (bloqueio fino) e *Ty* quer bloquear *T* (bloqueio grosso), como *Ty* sabe que *Tx* mantém um bloqueio fino?
- Solução
  - gerenciar bloqueios por níveis de granularidade
    - além do uso de bloqueios S e X, uso de bloqueios de intenção

Bloqueios de Intenção

- Indicam, em grânulos mais grossos, que *Tx* está bloqueando algum dado em um grânulo mais fino
  - vê o BD como uma árvore de grânulos
- Tipos de bloqueios de intenção
  - IS (*Intention-Shared*)
    - indica que um ou mais bloqueios compartilhados serão solicitados em nodos descendentes
  - IX (*Intention-eXclusive*)
    - indica que um ou mais bloqueios exclusivos serão solicitados em nodos descendentes
  - SIX (*Shared-Intention-eXclusive*)
    - bloqueia o nodo corrente no modo compartilhado, porém um ou mais bloqueios exclusivos serão solicitados em nodos descendentes

Exemplo

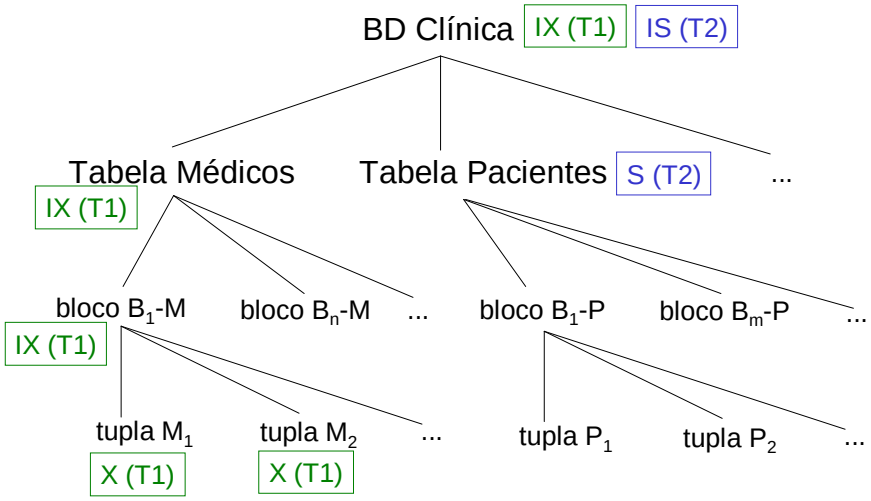

Tabela de Compatibilidade de Bloqueios

	IS	**IX**	**S**	**SIX**	**X**
IS	verdadeiro	verdadeiro	verdadeiro	verdadeiro	falso
IX	verdadeiro	verdadeiro	falso	falso	falso
S	verdadeiro	falso	verdadeiro	falso	falso
SIX	verdadeiro	falso	falso	falso	falso
X	falso	falso	falso	falso	falso

Técnica de Bloqueio de Várias Granularidades

- Protocolo que atende às seguintes regras:
  1. A tabela de compatibilidade de bloqueios deve ser respeitada
  2. A raiz da árvore deve ser bloqueada em primeiro lugar, em qualquer modo
  3. Um nodo *N* pode ser bloqueado por *Tx* no modo S ou IS se o nodo pai de *N* já estiver bloqueado por *Tx* no modo IS ou IX
  4. Um nodo *N* pode ser bloqueado por *Tx* no modo X, IX ou SIX se o nodo pai de *N* já estiver bloqueado por *Tx* no modo IX ou SIX
  5. *Tx* pode bloquear um nodo se não tiver desbloqueado nenhum nodo (é 2PL!)
  6. *Tx* pode desbloquear um nodo *N* se nenhum dos filhos de *N* estiver bloqueado por *Tx*

Técnica de Bloqueio de Várias Granularidades

- Serializabilidade é garantida
  - segue-se um protocolo 2PL
- Obtenção de bloqueios é *top-down*
- Liberação de bloqueios é *bottom-up*
- Vantagens
  - reduz o *overhead* na imposição de bloqueios
  - adequada a qualquer tipo de transação
    - alcance de dados pequeno, médio ou grande
- Desvantagens
  - maior controle e registro de bloqueios
  - não está livre de *deadlock*

Exemplo

- T1: deseja atualizar os dados do médico com CRM 100 (está no bloco B_1-M) e do paciente com CPF 200 (está no bloco B_2-P)
- T2: deseja atualizar os médicos ortopedistas (estão no bloco B_{10}-M)
- T3: deseja ler os dados do médico com CRM 50 (está no bloco B_1-M) e todos os dados de pacientes
- Escalonamento (apenas os bloqueios são mostrados)

$H_{2PL\text{-}VG}$ = lix1(BD) lix1(Médicos) lix2(BD) lis3(BD) lis3(Médicos) lis3(Médicos.BlocoB_1-M) lix1(Médicos.BlocoB_1-M) lx1(Médicos[CRM=100]) lix2(Médicos) lx2(Médicos.BlocoB_{10}-M) ls3(Médicos[CRM=50]) lix1(Pacientes) lix1(Pacientes.BlocoB_2-P) lx1(Pacientes[CPF=200]) u1(Pacientes[CPF=200]) u1(Pacientes.BlocoB_2-P) u1(Pacientes) ls3(Pacientes) u2(Médicos.BlocoB_{10}-M) u2(Médicos) u2(BD) u1(Médicos[CRM=100]) u1(Médicos.BlocoB_1-M) u1(Médicos) u1(BD) u3(Médicos[CRM=50]) u3(Médicos.BlocoB_1-M) u3(Médicos) u3(Pacientes) u3(BD)

Exercício 6

1. Apresente um escalonamento concorrente 2PL de várias granularidades (considerando os níveis: BD-Tabela-Tupla) para as transações abaixo:

   T1: r(Médicos[CRM=100]) w(Médicos[CRM=100])
   w(Pacientes[CPF=101])
   T2: r(Médicos) r(Pacientes[CPF=200])
   w(Pacientes[CPF=200])
   T3: r(Pacientes[CPF=101]) w(Pacientes[CPF=111])
   T4: r(Médicos)
   w(Médicos[especialidade = 'ortopedia'])

   Obs.: o médico com CRM=100 é ortopedista.