

Schedulers Otimistas

- Técnicas pessimistas
 - *overhead* no processamento de transações
 - executam verificações e ações antes de qualquer operação no BD para garantir a serializabilidade (solicitação de bloqueio, teste de TS)
- Técnicas otimistas
 - não realizam nenhuma verificação durante o processamento da transação
 - pressupõem nenhuma ou pouca interferência
 - verificações de violação de serializabilidade feitos somente ao final de cada transação
 - técnica mais conhecida: *Técnica de Validação*

Scheduler Baseado em Validação

- Técnica na qual atualizações de uma transação T_x são feitas sobre cópias locais dos dados
- Quando T_x solicita *commit* é feita a sua **validação**
 - T_x violou a serializabilidade?
 - **SIM**: T_x é abortada e reiniciada posteriormente
 - **NÃO**: atualiza o BD a partir das cópias dos dados e encerra T_x

Técnica de Validação

- Cada transação T_x passa por 3 fases:

1. Leitura

- T_x lê dados de transações *committed* do BD e atualiza dados em cópias locais

2. Validação

- análise da manutenção da serializabilidade de conflito caso as atualizações de T_x sejam efetivadas no BD

3. Escrita

- se fase de Validação for OK, aplica-se as atualizações de T_x no BD e T_x encerra com sucesso; caso contrário, T_x é abortada

Técnica de Validação

- Duas listas de dados são mantidas para T_x
 - lista-READ(T_x) : conjunto de dados que T_x leu
 - lista-WRITE(T_x): conjunto de dados que T_x atualizou
- Três *timesteps* são definidos para T_x
 - *TS-Start*(T_x): início da fase de leitura de T_x
 - *TS-Validation*(T_x): início da fase de validação de T_x
 - *TS-Finish*(T_x): término da fase de escrita de T_x

Funcionamento da Técnica

- Durante a fase de Leitura
 - T_x lê / atualiza dados; lista-READ(T_x) e list-WRITE(T_x) vão sendo alimentadas
- Durante a fase de Validação
 - três condições são testadas entre T_x e toda transação T_y que **já encerrou com sucesso** ou **está sofrendo validação**
 - se **alguma** das condições for **VERDADEIRA** para TODA T_y
 - T_x passa para a **fase de Escrita** e encerra com sucesso
 - caso contrário
 - há interferência entre T_x e T_y
 - T_x é abortada e suas cópias locais são descartadas

Condições para Validação de T_x

- Condição 1

- $TS-Finish(T_y) < TS-Start(T_x)$

- se T_y encerrou suas atualizações antes de T_x iniciar, então T_x não interfere em T_y

- Exemplo

- $H_{V-C_1} = s1 \ r1(A) \ s2 \ r2(B) \ w1(A) \ v1 \ c1 \ w2(A) \ v2$
 $c2 \ sx \ rx(A) \ s3 \ r3(Z) \ wx(A) \ vx \ cx \dots$

$TS-Finish(T1) < TS-Start(T_x) \quad E$

$TS-Finish(T2) < TS-Start(T_x)$

Condições para Validação de T_x

- Condição 2

- $TS-Start(T_x) < TS-Finish(T_y) < TS-Validation(T_x)$
E $lista-READ(T_x) \cap lista-WRITE(T_y) = \emptyset$
 - T_y encerrou durante a execução de T_x e T_x não leu nenhum dado que possa ter sido atualizado por T_y (não há risco de T_y ter interferido nos dados lidos por T_x)

- Exemplo

- $H_{V-C2} = s1 \ r1(C) \ s2 \ r2(B) \ w1(C) \ v1 \ c1 \ sx \ rx(B) \ rx(C) \ w2(A) \ v2 \ c2 \ wx(B) \ s3 \ r3(Z) \ vx \ cx \dots$
 - $T1$ atende condição 1 em relação à T_x
 - $T2$ atende condição 2 em relação à T_x :
 - $lista-READ(T_x) = \{B, C\}$
 - $lista-WRITE(T2) = \{A\}$

Condições para Validação de T_x

- Condição 3

- $TS-Validation(T_y) < TS-Validation(T_x) \wedge$

- $lista-READ(T_x) \cap lista-WRITE(T_y) = \emptyset \wedge$

- $lista-WRITE(T_x) \cap lista-READ(T_y) = \emptyset \wedge$

- $lista-WRITE(T_x) \cap lista-WRITE(T_y) = \emptyset$

- T_y já estava em validação, mas não há operações em conflito entre ela e T_x

- Exemplo

- $H_{VAL-C3} = s1 \ r1(C) \ s2 \ r2(B) \ w1(C) \ v1 \ c1 \ sx \ rx(B) \ s3 \ r3(C) \ rx(C) \ w2(A) \ v2 \ c2 \ w3(Y) \ w3(Z) \ v3 \ wx(B) \ vx \ cx \dots$

- $T1$ atende condição 1 em relação à T_x

- $T2$ atende condição 2 em relação à T_x

- $T3$ atende condição 3 em relação à T_x :

- $lista-READ(T3) = \{C\}$

- $lista-WRITE(T3) = \{Y, Z\}$

- $lista-READ(T_x) = \{B, C\}$

- $lista-WRITE(T_x) = \{B\}$

Scheduler Baseado em Validação

- Vantagens

- Reduz o *overhead* durante a execução de T_x
- Evita aborto em cascata
 - T_x não grava no BD antes de suas atualizações serem validadas em memória
 - se há interferência entre T_x e outra T_y *committed* ou em validação, as atualizações de T_x são descartadas

- Desvantagens

- Dados atualizados por T_x devem se manter em memória até o final de T_x
- Interferências entre T_x e outras transações (isso não é esperado pois a técnica é otimista), são descobertas somente ao final da execução de T_x (na validação) e T_x pode ser reiniciada só após essa validação

Exercícios 6

1. Apresente um escalonamento que não seja serializável por validação para as transações abaixo:

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. Um *scheduler* baseado em validação garante um escalonamento passível de recuperação pelo *recovery*?

Bloqueios e Granularidade

- Grânulo
 - porção do BD
 - atributo, tupla, tabela, bloco, ...
 - níveis de granularidade
 - granularidade fina
 - porção pequena do BD \Rightarrow 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 \Rightarrow 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**
 - T_x atualiza uma tupla; T_y atualiza toda uma tabela
- Algumas questões
 - se T_y quer atualizar toda uma tabela, T_y deve bloquear TODAS as tuplas?
 - se T_x bloqueou uma tupla da tabela T (bloqueio fino) e T_y quer bloquear T (bloqueio grosso), como T_y sabe que T_x 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 T_x está bloqueando algum dado em um grânulo mais fino
 - BD é visto 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

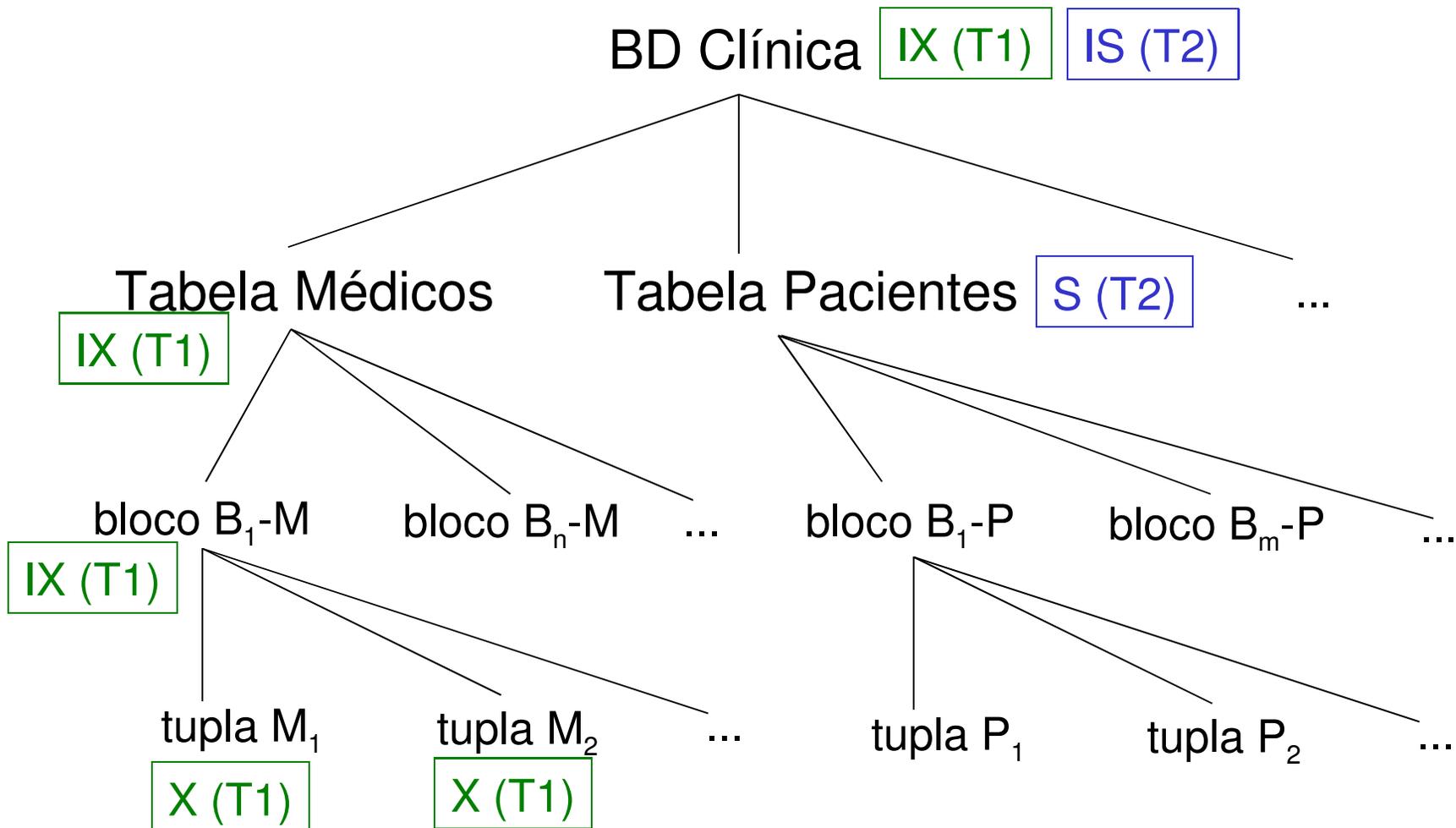


Tabela de Compatibilidade de Bloqueios

	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 T_x no modo S ou IS se o nodo pai de N já estiver bloqueado por T_x no modo IS ou IX
 4. Um nodo N pode ser bloqueado por T_x no modo X, IX ou SIX se o nodo pai de N já estiver bloqueado por T_x no modo IX ou SIX
 5. T_x pode bloquear um nodo se não tiver desbloqueado nenhum nodo (é 2PL!)
 6. T_x pode desbloquear um nodo N se nenhum dos filhos de N estiver bloqueado por T_x

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)

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

Exercícios 7

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.