Práticas de Banco de Dados

Parte 5 – Controle de Concorrência

Professor Eduardo Xavier

Técnicas de Controle de Concorrência

- Técnicas **Pessimistas**
  - Supõem que **sempre** ocorre interferência entre transações e garantem a serializabilidade enquanto a transação está ativa
  - Técnicas:
    - **Bloqueio** (**locking**)
    - **Timestamp**
- Técnicas **Otimistas**
  - Supõem que **quase nunca** ocorre interferência entre transações e verificam a serializabilidade somente ao final de uma transação
  - Técnica
    - **Validação**

Bloqueios (Locking)

- **Bloqueios**, ou **lockings**, são técnicas utilizada para controlar a execução concorrente de transações.
- São as mais utilizadas pelos SGBDs relacionais modernos.
- Um bloqueio ou lock é uma variável **associada a cada item de dado** que descreve o estado do item em relação às operações que podem ser aplicadas ao mesmo.
- Bloqueios são usados como um meio de **sincronizar acessos concorrentes** aos itens do banco de dados

Bloqueios (Locking)

- Princípio de funcionamento
  - Controle de operações de **leitura** (*read(X)*), **escrita** (*write(X)*) e **postergação** (através de bloqueio) de algumas dessas operações de modo a **evitar conflito**.

- Todo dado possui um status de bloqueio, que pode ser:
  - liberado (*Unlocked* - U)
  - com bloqueio compartilhado (*Shared lock* - S)
  - com bloqueio exclusivo (*eXclusive lock* - X)

Bloqueios (Locking)

- **Bloqueio Compartilhado** (S)
  - Solicitado por uma transação que deseja realizar leitura de dado D várias transações podem manter esse bloqueio sobre D
- **Bloqueio Exclusivo** (X)
  - Solicitado por uma transação que deseja realizar 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>`

Bloqueios (Locking)

- Operações de Bloqueio e Histórico
  - 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) ou lock-S(D) ou read-lock(D)***: solicitação de bloqueio compartilhado sobre D
  - ***lx(D) ou lock-X(D) ou write-lock(D)***: solicitação de bloqueio exclusivo sobre D
  - ***u(D) ou unlock(D)***: libera o bloqueio sobre D

Bloqueios (Locking)

- Exemplo de histórico de bloqueios:
  - Considere a tabela ao lado como duas transações T1 e T2 emitindo comandos ao scheduler do SGBD ao longo do tempo.
  - O histórico desses comandos pode ser representado por:

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

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( )	

Bloqueios (Locking)

- É permitido a uma transação que já tenha um bloqueio no item X converter o bloqueio de um estado para outro.
  - **Upgrade** – mudança para um estado mais rígido – exemplo : uma transação T pode emitir um read_lock(X) e depois um write_lock(X), se T for a única com bloqueio read em X poderá executar o upgrade, caso contrário espera.
  - **Downgrade** – mudança para um estado menos rígido – exemplo : uma transação T que está com bloqueio exclusivo write_lock(X) pode emitir um read_lock(X).

Bloqueios (Locking)

- **Problema**: o simples uso de bloqueis **não garante** escalonamentos serializáveis de transação
- Exemplo

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

Note que tanto as transações 1 e 2 manipulam os dados X e Y. O dado Y é alterado (e "comitado") por T2 antes de T1 terminar de trabalhar com ele, enquanto que o dado X foi lido e atualizado por T2, porém T1 alterou X após isso acontecer (o que pode impactar no resultado de T2).

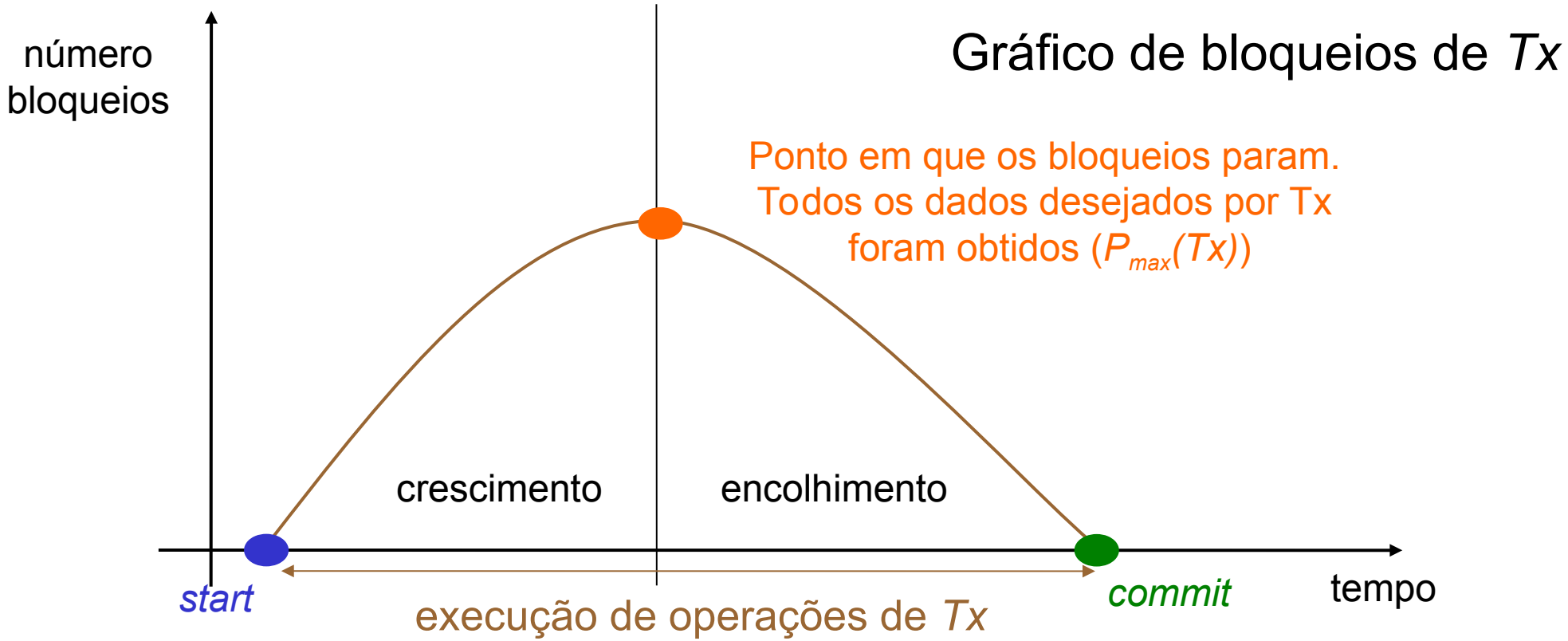
- Necessita-se de uma técnica mais rigorosa de bloqueio para garantir que o escalonamento usado possa ser serial.
- A solução mais utilizada é o **bloqueio de duas fases** (*two-phase locking* – **2PL**)

Bloqueios de Duas Fases

- **2PL / Bloqueio de Duas Fases**
  - Premissa:
    - *"para toda transação Tx, todas as operações de bloqueio de dados feitas por Tx precedem a primeira operação de desbloqueio feita por Tx"*
  - Funcionamento do protocolo de duas fases:
    - Fase de expansão ou crescimento
      - *Tx* pode obter bloqueios, mas não pode liberar nenhum bloqueio
    - Fase de retrocesso ou encolhimento
      - *Tx* pode liberar bloqueios, mas não pode obter nenhum bloqueio

Bloqueios de Duas Fases

- **2PL / Bloqueio de Duas Fases**



Bloqueios de Duas Fases

- **Comparação**

T (sem protocolo)	T' (usando protocolo de duas fases)	
read-lock(y);	read-lock(y);	Primeira fase
read-item(y);	read-item(y);	
unlock(y);	write-lock(x);	
write-lock(x);	unlock(y);	Segunda fase
read-item(x);	read-item(x);	
x=x+y;	x=x+y;	
write-item(x);	write-item(x);	
unlock(x);	unlock(x);	

Em T' → Na primeira fase prende todos os recursos necessários e na segunda fase libera os recursos.

Bloqueios de Duas Fases

- **Importante:**
  - Se todas as transações em um escalonamento seguirem o protocolo de bloqueio em duas fases, o escalonamento é garantidamente serializável.
  - O bloqueio em duas fases limita o volume de concorrência, podendo provocar deadlock (impasse) / starvation (espera infinita).

Bloqueios de Duas Fases

- **Importante:**
  - Se todas as transações em um escalonamento seguirem o protocolo de bloqueio em duas fases, o escalonamento é **garantidamente serializável**.
  - O bloqueio em duas fases limita o volume de concorrência.
    - Isso pode provocar deadlock (impasse) / starvation (espera infinita).
    - Mesmo não provocando deadlocks ou starvation, um dado pode permanecer bloqueado por *Tx* muito tempo até que *Tx* adquira bloqueios em todos os outros dados que deseja, o que pode provocar problemas de performance.

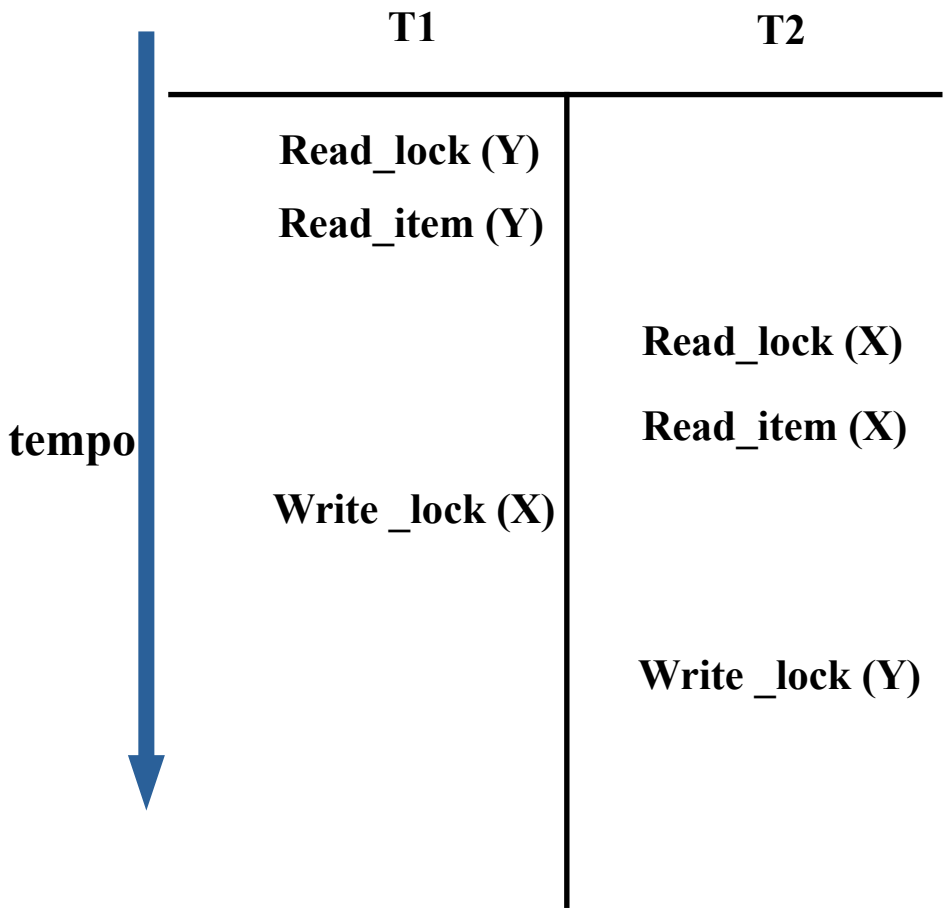
Bloqueios de Duas Fases

- **Importante:**
  - Se todas as transações em um escalonamento seguirem o protocolo de bloqueio em duas fases, o escalonamento é **garantidamente serializável**.
  - O bloqueio em duas fases limita o volume de concorrência.
    - Isso pode provocar **deadlock (impasse)** ou **starvation (espera infinita)**.
    - Mesmo não provocando deadlocks ou starvation, um dado pode permanecer bloqueado por *Tx* muito tempo até que *Tx* adquira bloqueios em todos os outros dados que deseja, o que pode provocar problemas de performance.
  - O **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
    - Não garante escalonamentos adequados à recuperação pelo *recovery*

Deadlocks (Impasses)

- Um **deadlock** ocorre quando cada transação T1 em um conjunto de duas ou mais transações está esperando algum ítem que esteja bloqueado por alguma outra transação T2 no conjunto.

- Cada transação do conjunto está na fila de espera aguardando que o recurso (ítem) seja liberado.



Prevenção de Deadlocks

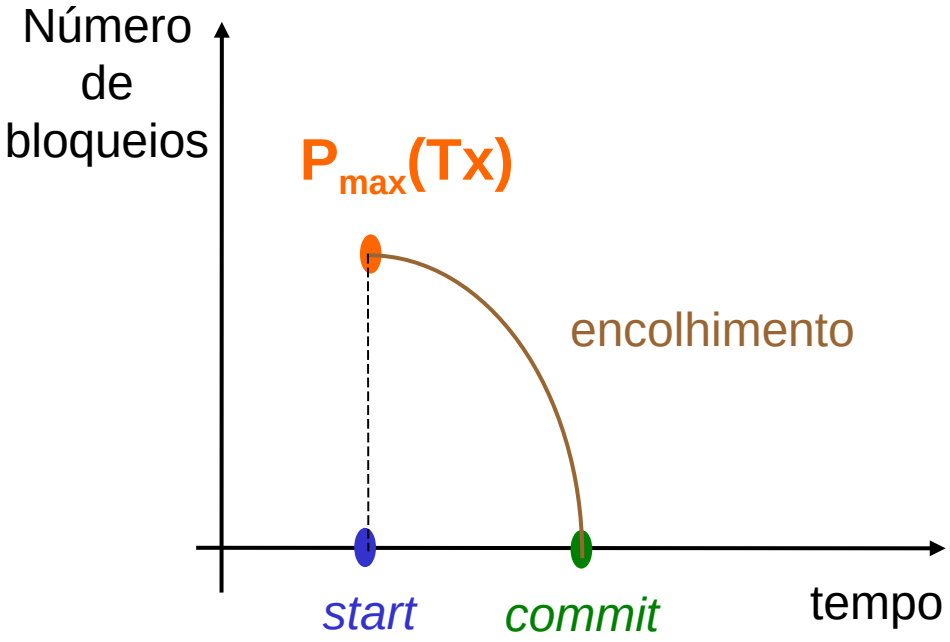
- **Timeouts**
  - Uma transação fica em estado de espera por um determinado tempo estabelecido no sistema. Após atingir o tempo ela é desfeita. Método prático e simples com baixo nível de overhead.
  - O timeout ocorre quando uma transação não pode continuar sua execução durante um intervalo indefinido de tempo, enquanto que outras transações continuam a ser executadas normalmente.

Tratamento de Deadlocks

- **Protocolos de Prevenção**
  - Abordagens pessimistas
    - *Deadlocks* ocorrem com frequência!
    - Impõem um *overhead* no processamento de transações
      - controles adicionais para evitar *deadlock*
    - tipos de protocolos pessimistas
      - técnica de bloqueio 2PL conservador
      - técnicas baseadas em *timestamp* (*wait-die / wound-wait*)
      - técnica de *espera-cautelosa* (*cautious-waiting*)
  - Uso de *timeout*
    - Se tempo de espera de *Tx* > *timeout* então *abort(Tx)*

Tratamento de Deadlocks

- **Protocolo 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!



Tratamento de Deadlocks

- **Timestamp**
  - A ideia é associar um rótulo de tempo a cada transação *Tx* (*TS(Tx)*)
- Técnicas
  - Considere que *Tx* deseja um dado bloqueado por outra transação *Ty*
  - **Técnica 1: *Esperar-ou-morrer* (*wait-die*)**

1) Se TS(Tx) < TS(Ty)

Então Tx é mais velha que Ty

Neste caso Tx aguarda (wait) para prosseguir bloqueando

2) Se TS(Tx) > TS(Ty)

Então Tx é mais nova que Ty

Tx é cancelada (die) e reiniciada com o mesmo timestamp anterior após um período de espera

Tratamento de Deadlocks

- **Timestamp (continuação)**
  - **Técnica 2: *Ferir-ou-esperar* (wound-*wait*)**

1) Se TS(Tx) < TS(Ty)
Então Tx é mais velha que Ty
Tx força rollbak em Ty (wound - Tx "fere" Ty)
Ty é cancelada e reiniciada com o mesmo timestamp anterior após um período de espera
2) Se TS(Tx) > TS(Ty)
Então Tx é mais nova que Ty
Neste caso Tx é forçada a esperar (wait) até que o recurso disputado esteja disponível.

- **Vantagem das duas técnicas:** evitam *starvation* (espera indefinida) de uma transação *Tx (q*uanto mais antiga for *Tx*, maior a sua prioridade)
- **Desvantagem das técnicas:** muitos abortos podem ser provocados, sem nunca ocorrer um deadlock

Tratamento de Deadlocks

- **Espera Cautelosa**

Se *Tx* deseja D e D está bloqueado por *Ty*
Então Se *Ty* não está em alguma fila de espera (WAIT)
então *Tx* espera D ser liberado
senão início
abort(Tx)
start(Tx)
fim

- **Vantagem**
  - Se *Ty* já está em espera, *Tx* é abortada para evitar um possível ciclo de espera
  - É livre de Deadlock
- **Desvantagem**
  - A mesma das técnicas baseadas em *timestamp*

Tratamento de Deadlocks

- **Validação**
  - Em técnicas de controle de concorrência de validação (ou técnicas otimistas), nenhuma verificação é realizada enquanto a transação está sendo executada, diminuindo um custo adicional durante a execução da transação.
  - Geralmente, as técnicas otimistas de controle de concorrência funcionam bem se houver pouca interferência (suposição otimista) entre as transações de um escalonamento.

Tratamento de Deadlocks

- **Validação – Exemplo:**
  - Uma das técnicas otimistas propõe as seguintes regras:
    - Atualizações na transação não são aplicadas diretamente aos itens do banco de dados até que a transação atinja seu final.
    - Todas as atualizações são aplicadas a cópias locais dos itens de dados.
    - Ao final de uma transação, a fase de validação verifica se qualquer uma das atualizações da transação viola a serialização.
    - Se a serialização não for violada, a transação é confirmada e o banco de dados é atualizado a partir das cópias locais; caso contrário, a transação é abortada e posteriormente reiniciada.

Tratamento de Deadlocks

- **Protocolos de detecção**
  - Abordagens otimistas
    - *Deadlocks* não ocorrem com frequência (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 2PL

- **2PL Conservador (ou Estático)**
  - A transação deve solicitar o bloqueio de TODOS os itens que acessa antes de iniciar a execução, ESPERANDO até que todos estejam disponíveis.
  - Evita deadlock, porém Tx pode esperar muito para executar
  - Pouco prático e inviável na maioria dos casos.

- **2PL Estrito (muito usado pelos SGBDs)**
  - Tx só libera seus bloqueios exclusivos após executar commit ou abort
  - Assim, nenhuma outra transação pode ler ou escrever item a menos que T tenha sido efetivada.
  - Vantagem: garante escalonamentos estritos
  - Desvantagem: não está livre de deadlocks

Outras Técnicas de 2PL

- **2PL Rigoroso**
  - Tx só libera TODOS os seus bloqueios (S e X) após executar commit ou abort
  - Vantagem:
    - Menos overhead para Tx, já que não há esforço para liberar bloqueios antes do final.
  - Desvantagem:
    - Não está livre de deadlocks

Outras Técnicas de 2PL

- 2PL – Quadro comparativo

T1 SEM ADOTAR PROTOCOLO 2PL	T1 COM 2PL BÁSICO	T1 COM 2PL CONSERVADOR	T1 COM 2PL ESTRITO	T1 COM 2PL RIGOROSO
READ_LOCK(Y)	READ_LOCK(Y)	READ_LOCK(Y)	READ_LOCK(Y)	READ_LOCK(Y)
READ_ITEM(Y)	READ_ITEM(Y)	WRITE_LOCK(X)	READ_ITEM(Y)	READ_ITEM(Y)
UNLOCK(Y)	WRITE_LOCK(X)	READ_ITEM(Y)	WRITE_LOCK(X)	WRITE_LOCK(X)
WRITE_LOCK(X)	UNLOCK(Y)	UNLOCK(Y)	UNLOCK(Y)	* NÃO DESBLOQUEIA O ELEMENTO DE BLOQUEIO COMPARTILHADO
READ_ITEM(X)	READ_ITEM(X)	READ_ITEM(X)	READ_ITEM(X)	READ_ITEM(X)
X=X+Y	X=X+Y	X=X+Y	X=X+Y	X=X+Y
WRITE_ITEM(X)	WRITE_ITEM(X)	WRITE_ITEM(X)	WRITE_ITEM(X)	WRITE_ITEM(X)
UNLOCK(X)	UNLOCK(X)	UNLOCK(X)	* NÃO DESBLOQUEIA O ELEMENTO DE BLOQUEIO EXCLUSIVO	* NÃO DESBLOQUEIA O ELEMENTO DE BLOQUEIO EXCLUSIVO
COMMIT	COMMIT	COMMIT	COMMIT	COMMIT

Referencias bibliográfica

- Sistemas de banco de dados (Elmasri / Navathe)
  - Capítulo 20