CONTROLE DE CONCORRÊNCIA - PROTOCOLOS BASEADOS EM LOCK

Sérgio Mergen

Versão modificada de Database System Concepts, 5th Ed.

Tópicos

- Mecanismo de Lock
- Protocolo Two-Phase Locking
- Protocolo Strict Two-Phase Locking
- Conversões de Lock
- Aquisição/Liberação Automática de Locks
- Protocolo Rigorous Two-Phase Locking
- Implementação do Locking
- Problemas a tratar com o Protocolo Two-phase Locking
- Protocolos Baseados em Grafo

Mecanismo de Lock

- Um lock é um mecanismo que controla o acesso concorrente a um item de dados

- Itens de dados podem sofrer lock em dois modos:
- 1. *exclusivo (X)*
  - O item de dados pode ser tanto lido quanto escrito
  - O lock é obtido usando a instrução **lock-X**.

- 2. *compartilhado (S)*.
  - O item de dados pode ser apenas lido
  - O lock é obtido usando a instrução **lock-S**.

Mecanismo de Lock

- Os pedidos de lock são feitos para o gerenciador do controle de concorrência.
  - A transação pode prosseguir apenas depois que o pedido for aceito
- Quando a transação efetua um unlock
  - Ela avisa que não mais usará o item de dados

Mecanismo de Lock

- Matriz de compatibilidade de Lock

Lock concedido \ Lock solicitado	S	X
S	Sim	Não
X	Não	Não

- Uma transação pode ter um pedido de lock em um item aceito se
  - O pedido de lock for compatível com os locks já obtidos sobre o item por outras transações

Mecanismo de Lock

- Matriz de compatibilidade de Lock

Lock concedido \ Lock solicitado	S	X
S	Sim	Não
X	Não	Não

- Várias transações podem ter locks compartilhados sobre um item
  - Mas se uma transação possui um lock exclusivo sobre um item
    - Nenhuma outra transação pode obter qualquer tipo de lock sobre o mesmo item

Mecanismo de Lock

- Matriz de compatibilidade de Lock

Lock concedido \ Lock solicitado	S	X
S	Sim	Não
X	Não	Não

- Se um lock não puder ser dado
  - A transação que fez o pedido deve esperar
    - Até que todos os locks incompatíveis das outras transações sejam liberados (através de unlocks)
    - Para então o pedido de lock ser aceito.

Mecanismo de Lock

- Ao lado, exemplo de transações realizando locks e unlocks

- Isso por si só não é suficiente para garantir schedules serializáveis.
  - A gravação de A em T1 leva a soma impressa errada em T2.

T1	T2
lock-X*(A);*	
read (A);	
A := A – 50;	
write *(A);*	
unlock*(A);*	
	lock-S*(A);*
	lock-S*(B);*
	read(A);
	read(B);
	print(A + B);
	unlock*(A);*
	unlock*(B);*
	commit;
lock-X*(B);*	
read *(B);*	
B := B + 50;	
write *(B);*	
unlock*(B);*	
commit;	

Tópicos

- Mecanismo de Lock
- Protocolo Two-Phase Locking
- Protocolo Strict Two-Phase Locking
- Conversões de Lock
- Aquisição/Liberação Automática de Locks
- Protocolo Rigorous Two-Phase Locking
- Implementação do Locking
- Problemas a tratar com o Protocolo Two-phase Locking
- Protocolos Baseados em Grafo

Protocolo Two-Phase Locking

- Um **protocolo de locking** é um conjunto de regras seguidas por todas as transações ao pedir e liberar locks.

- O papel do protocolo é restringir o conjunto de schedules possíveis, visando em primeiro lugar schedules cujas transações sejam
  - Concorrentes
  - Isoladas entre si
  - Recuperáveis

- Um dos protocolos mais conhecidos é o protocolo de lock de duas fases (**two-phase locking**)

Protocolo Two-Phase Locking

- O Two-Phase Locking garante schedules serializáveis em conflito.

- Fase 1: Fase de crescimento
  - Transações podem obter locks
  - Transações não podem liberar locks
- Fase 2: Fase de encolhimento
  - Transações podem liberar locks
  - Transações não pode obter locks

Mecanismo de Lock

- O schedule ao lado não obedece ao *two-phase locking*

- Na transação T1, um **lock** aparece depois de um **unlock**

T1	T2
lock-X*(A);*	
read (A);	
A := A – 50;	
write *(A);*	
unlock*(A);*	
	lock-S*(A);*
	lock-S*(B);*
	read(A);
	read(B);
	print(A + B);
	unlock*(A);*
	unlock*(B);*
	commit;
lock-X*(B);*	
read *(B);*	
B := B + 50;	
write *(B);*	
unlock*(B);*	
commit;	

Protocolo Two-Phase Locking

- O schedule ao lado obedece ao *two-phase locking*
- Em cada transação, todos **locks** vem antes de todos **unlocks**

T1	T2
lock-X*(A)*; **read (A);** **write *(A)*;** **lock-S*(B)*;** **unlock*(A)*;** **read *(B)*;** **unlock*(B)*;** **Commit**	**lock-S*(C)*;** **read (C);** **lock-S*(A)*;** **read *(A)*;** **unlock*(C)*;** **unlock*(A)*;** **Commit**

Protocolo Two-Phase Locking

- É possível provar que as transações podem ser serializadas na mesma ordem que seus **pontos de lock**
- Ou seja, o ponto onde a transação adquiriu o lock final.
- No caso ao lado, o schedule serial seria <T1, T2>

T1	T2
lock-X*(A)*;	
read (A);	
write *(A)*;	
lock-S*(B)*;	
unlock*(A)*;	
	lock-S*(C)*;
	read (C);
read *(B)*;	
	lock-S*(A)*;
unlock*(B)*;	
	read *(A)*;
Commit	**unlock*(C)*;**
	unlock*(A)*;
	Commit

Protocolo Two-Phase Locking

- Usando as regras brutas, o protocolo apresenta alguns problemas
  - Não previne rollbacks em cascata
  - Gera schedules não recuperáveis

- Esses problemas podem ser resolvidos usando uma versão mais restrita do protocolo
  - Conforme veremos mais adiante

Protocolo Two-Phase Locking

- **Exemplo de rollback em cascata**

- No schedule ao lado, ao abortar T1
  - T2 também precisaria abortar

T1	T2
lock-X*(A)*; read (A); write *(A)*; lock-S*(B)*; unlock*(A)*; read *(B)*; unlock(B); abort	 lock-S*(A)*; read *(A)*; abort (cascata)

Protocolo Two-Phase Locking

- **Exemplo de schedule não recuperável**

- No schedule ao lado, ao abortar T1
  - T2 precisaria ser revertida

T1	T2
lock-X*(A)*;	
read (A);	
write *(A)*;	
lock-S*(B)*;	
unlock*(A)*;	
	lock-S*(A)*;
read *(B)*;	
	read *(A)*;
unlock(B);	
	print(A);
	commit
abort	

Tópicos

- Mecanismo de Lock
- Protocolo Two-Phase Locking
- Protocolo Strict Two-Phase Locking
- Conversões de Lock
- Aquisição/Liberação Automática de Locks
- Protocolo Rigorous Two-Phase Locking
- Implementação do Locking
- Problemas a tratar com o Protocolo Two-phase Locking
- Protocolos Baseados em Grafo

Protocolo Strict Two-Phase Locking

- O protocolo two-phase locking puro tem inconvenientes
  - Não previne rollbacks em cascata
  - Gera schedules não recuperáveis

- Para evitar esses problemas, usa-se um protocolo ajustado chamado de **strict two-phase locking**.

- Nesse protocolo, uma transação deve segurar todos os locks exclusivos até a instrução final de commit/abort.

Protocolo Strict Two-Phase Locking

- O protocolo estrito só libera os locks exclusivos após
  - commit ou
  - abort
- O protocolo é livre de roll-back em cascata
  - Consequentemente, também é recuperável

T1	T2
lock-X*(A)*; **read (A);** **write *(A)*;** **lock-S*(B)*;** **read *(B)*;** **unlock(B);** **commit/abort** **unlock*(A)*;**	**lock-S*(A)*;** **read *(A)*;** **print(A)** **unlock(A);** **commit**

Tópicos

- Mecanismo de Lock
- Protocolo Two-Phase Locking
- Protocolo Strict Two-Phase Locking
- Conversões de Lock
- Aquisição/Liberação Automática de Locks
- Protocolo Rigorous Two-Phase Locking
- Implementação do Locking
- Problemas a tratar com o Protocolo Two-phase Locking
- Protocolos Baseados em Grafo

Conversões de Lock

- No schedule ao lado, T1 teve que pedir um lock exclusivo em A
- Isso levou T2 a ter que esperar que A fosse liberado
- No entanto, T1 demorou a gravar em A
  - E fazer uso efetivo do lock exclusivo

T1	T2
lock-X*(A)*; **read (A);** **lock-S*(B)*;** **read *(B)*;** **lock-S*(C)*;** **read *(C)*;** **write (A)** **unlock(B);** **unlock(C);** **commit** **unlock(A);**	**lock-S*(A)*;** **read *(A)*;** **unlock(A);** **commit**

Conversões de Lock

- Pode-se aumentar a concorrência usando conversões de lock
  - T1 primeiro pede lock compartilhado em A
  - Quando for necessário gravar, T1 pede lock exclusivo

T1	T2
lock-S*(A)*; **read (A);** **lock-S*(B)*;** **read *(B)*;** **lock-S*(C)*;** **read *(C)*;** **lock-X*(A)*;** **write (A)** **unlock(B);** **unlock(C);** **commit** **unlock(A);**	 **lock-S*(A)*;** **read *(A)*;** **unlock(A);** **commit**

Conversões de Lock

- Modificação do Two-phase locking com conversões de lock:
  - Primeira fase:
    - Pode adquirir um lock-S sobre um item
    - Pode adquirir um lock-X sobre um item
    - Pode converter um lock-S em um lock lock-X (upgrade)
  - Segunda fase:
    - Pode liberar um lock-S
    - Pode liberar um lock-X

Tópicos

- Mecanismo de Lock
- Protocolo Two-Phase Locking
- Protocolo Strict Two-Phase Locking
- Conversões de Lock
- Aquisição/Liberação Automática de Locks
- Protocolo Rigorous Two-Phase Locking
- Implementação do Locking
- Problemas a tratar com o Protocolo Two-phase Locking
- Protocolos Baseados em Grafo

Protocolo Two-Phase Locking

- O protocolo visto assegura serialização.
  - Mas depende de um programador para inserir as instruções de lock.

- Como evitar que a transação explicitamente peça locks e unlocks?
  - Aquisição automática de locks
  - Liberação automática de locks

Aquisição Automática de Locks

- Uma transação T_i envia as instruções de read/write sem explicitar as chamadas para lock.
- A operação de **read**(*D*) é processada da seguinte maneira:

se T_i tem um lock em *D* **então**
read(*D*)
senão begin
se necessário
aguardar até que nenhuma outra transação tenha **lock-X** em *D*
dar a T_i um **lock-S** sobre *D*;
read(*D*)
end

Aquisição Automática de Locks

- **write**(*D*) é processada da seguinte maneira:

```
se Ti tem um lock-X em D  então
    write(D)
senão begin
    se necessário
        esperar a liberação de todos os locks em D,
    se Ti tem um lock-S em D então
        upgrade lock em D  para lock-X
    senão
        dar a Ti um lock-X em D
    write(D)
end;
```

Liberação automática de locks

- Os locks (compartilhados e exclusivos) são liberados automaticamente quando a transação
  - Abortar ou
  - Commitar

- Essa estratégia também é conhecida como **Rigorous two-phase locking**
  - Segura todos os locks até o fim
  - em vez de segurar somente os locks exclusivos

Tópicos

- Mecanismo de Lock
- Protocolo Two-Phase Locking
- Protocolo Strict Two-Phase Locking
- Conversões de Lock
- Aquisição/Liberação Automática de Locks
- Protocolo Rigorous Two-Phase Locking
- Implementação do Locking
- Problemas a tratar com o Protocolo Two-phase Locking
- Protocolos Baseados em Grafo

Protocolo Rigorous Two-Phase Locking

- No protocolo estrito, uma transação deve segurar todos os locks exclusivos até a instrução final de commit/abort.

- Existe um protocolo ainda mais rigoroso (alias, é chamado de rigoroso)
  - Nessa versão, uma transação deve segurar todos os locks (exclusivos e compartilhados) até a instrução final de commit/abort

- SGBDs cujo controle de concorrência seja baseado no two phase locking usam o modo rigoroso

Protocolo Rigorous Two-Phase Locking

- Todos unlocks aparecem somente no final

T1	T2
read (A) / lock-S*(A)*	**read(C) / lock-S*(C)***
write *(A)* / lock-X*(A)*	
read *(B)* / lock-S*(B)*	
commit / unlock(A,B)	**read *(A)* / lock-S*(A)*; commit / unlock(A,C)**

Protocolo Rigorous Two-Phase Locking

- Os locks e unlocks são pedidos automaticamente

T1	T2
read (A) / lock-S*(A)*	
	read(C) / lock-S*(C)*
write *(A)* / lock-X*(A)*	
read *(B)* / lock-S*(B)*	
commit / unlock(A,B)	
	read *(A)* / lock-S*(A)*; commit / unlock(A,C)

Protocolo Rigorous Two-Phase Locking

- No protocolo rigoroso, as transações podem ser serializáveis na ordem em que comitarem.
- No exemplo ao lado, a ordem seria
  - <T1, T2>

T1	T2
read (A) / lock-S*(A)*	**read(C) / lock-S*(C)***
write *(A)* / lock-X*(A)*	
read *(B)* / lock-S*(B)*	
commit / unlock(A,B)	**read *(A)* / lock-S*(A)*; commit / unlock(A,C)**

Tópicos

- Mecanismo de Lock
- Protocolo Two-Phase Locking
- Protocolo Strict Two-Phase Locking
- Conversões de Lock
- Aquisição/Liberação Automática de Locks
- Protocolo Rigorous Two-Phase Locking
- Implementação do Locking
- Problemas a tratar com o Protocolo Two-phase Locking
- Protocolos Baseados em Grafo

Implementação do Locking

- Um **gerenciador de lock** pode ser implementado como um mecanismo separado para onde as transações enviam pedidos de
  - lock e
  - unlock

- O gerenciador responde a um pedido de lock
  - Enviando uma mensagem de liberação
  - Ou uma mensagem solicitando abort
    - Caso seja identificado algum problema (ex. um deadlock)

- A transação solicitante aguarda por uma resposta

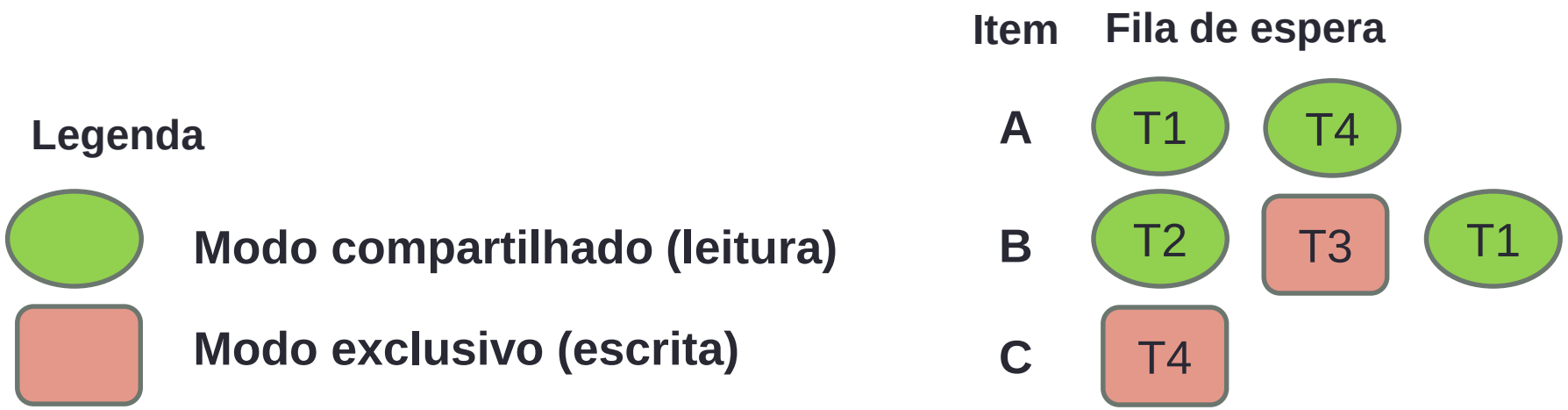
Implementação do Locking

- O gerenciador mantém uma estrutura de dados chamada de **tabela de lock** para registrar pedidos de lock aceitos e pendentes

- A tabela é normalmente implementada como uma hash table em memória
  - A chave é o nome do item de dados sofrendo lock
  - O valor é uma fila de espera que contém os pedidos

Implementação do Locking

- No exemplo abaixo
  - O item de dados **A** está sendo usado por **T1** e **T4** em modo **compartilhado**
  - O item de dados **B** está sendo usado por **T2** em modo **compartilhado**
    - **T3** está **esperando** na fila pela liberação
    - **T1** também está **esperando**, atrás de **T3**
  - O item de dados **C** está sendo usado por **T4** em modo exclusivo



Implementação do Locking

- Para compreender como o gerenciador de locks usa a tabela de locks, os próximos slides simulam transações solicitando locks

- Na simulação, uma transação de cada vez tenta realizar uma operação de forma intercalada
  - Ex. T1, T2, T3, T1, T2, T3, T1, T2, T3, T1, T2, T3, ...

- O objetivo da intercalação é simular a concorrência na disputa das transações pelo uso do processador

Exemplo

Inicialmente nenhum item está alocado

Schedule

T1	T2	T3

Transações

T1	T2	T3
R(A)	R(B)	W(B)
R(B)	R(C)	R(A)

Hashtable

Item **Fila de espera**

A

B

C

Exemplo

T1 obtém lock de leitura em A e efetua a operação

Schedule

T1	T2	T3
R(A)		

Transações

T1	T2	T3
R(A)	R(B)	W(B)
R(B)	R(C)	R(A)

Hashtable

Item	Fila de espera
A	T1
B	
C	

Exemplo

T2 obtém lock de leitura em B e efetua a operação

Schedule

T1	T2	T3
R(A)		
	R(B)	

Transações

T1	T2	T3
R(A)	R(B)	W(B)
R(B)	R(C)	R(A)

Hashtable

Item	Fila de espera
A	T1
B	T2
C	

Exemplo

T3 solicita lock de escrita em B e espera na fila

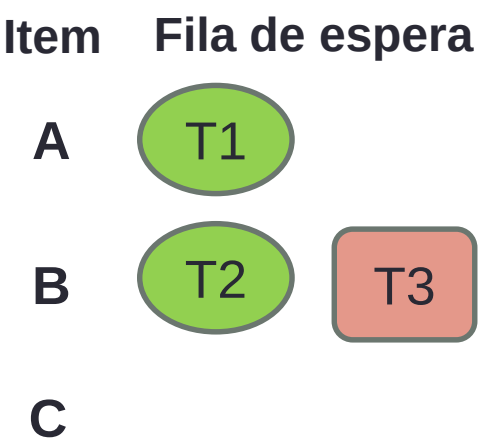
Schedule

T1	T2	T3
R(A)		
	R(B)	

Transações

T1	T2	T3
R(A)	R(B)	W(B)
R(B)	R(C)	R(A)

Hashtable



Exemplo

T1 solicita lock de leitura em B e espera na fila

Schedule

T1	T2	T3
R(A)		
	R(B)	

Transações

T1	T2	T3
R(A)	R(B)	W(B)
R(B)	R(C)	R(A)

Hashtable

Item	Fila de espera
A	T1
B	T2 → T3 → T1
C	

Exemplo

T2 obtém lock de leitura em C e efetua a operação

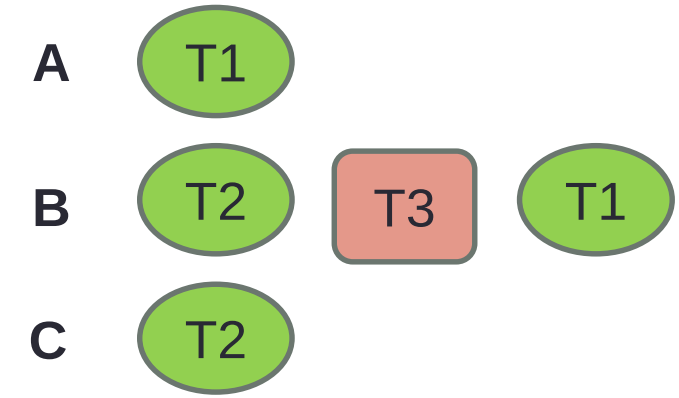
Schedule

T1	T2	T3
R(A)		
	R(B)	
	R(C)	

Transações

T1	T2	T3
R(A)	R(B)	W(B)
R(B)	R(C)	R(A)

Hashtable



Exemplo

T3 aguarda liberação do item de dados B

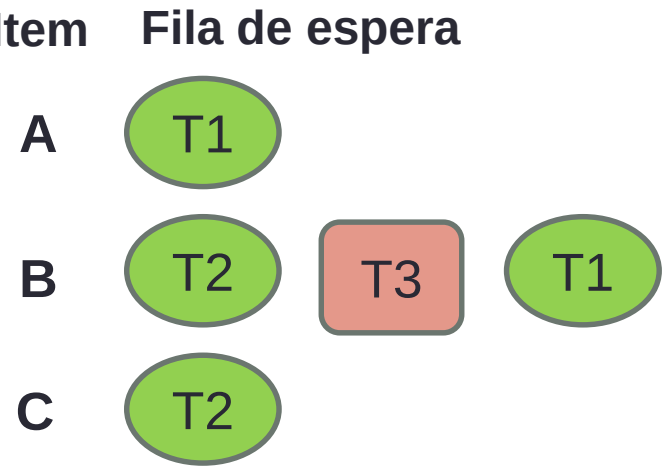
Schedule

T1	T2	T3
R(A)		
	R(B)	
	R(C)	

Transações

T1	T2	T3
R(A)	R(B)	W(B)
R(B)	R(C)	R(A)

Hashtable



Exemplo

T1 aguarda liberação do item de dados B

Schedule

T1	T2	T3
R(A)		
	R(B)	
	R(C)	

Transações

T1	T2	T3
R(A)	R(B)	W(B)
R(B)	R(C)	R(A)

Hashtable

Item	Fila de espera
A	T1
B	T2 T3 T1
C	T2

Exemplo

T2 pode comitar

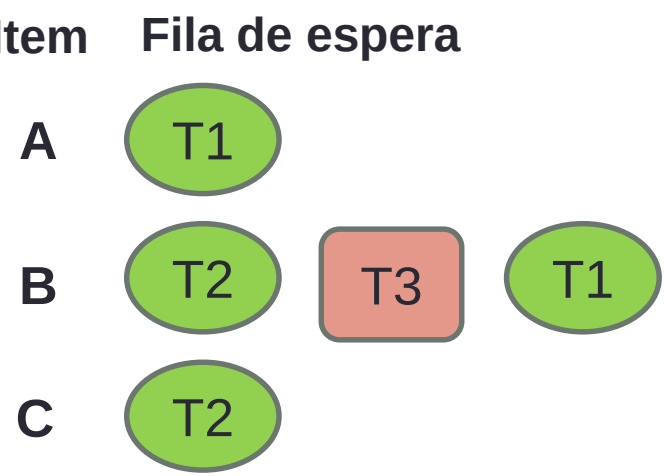
Schedule

T1	T2	T3
R(A)		
	R(B)	
	R(C)	
	Commit	

Transações

T1	T2	T3
R(A)	R(B)	W(B)
R(B)	R(C)	R(A)

Hashtable



Exemplo

Os itens bloqueados por T2 são liberados

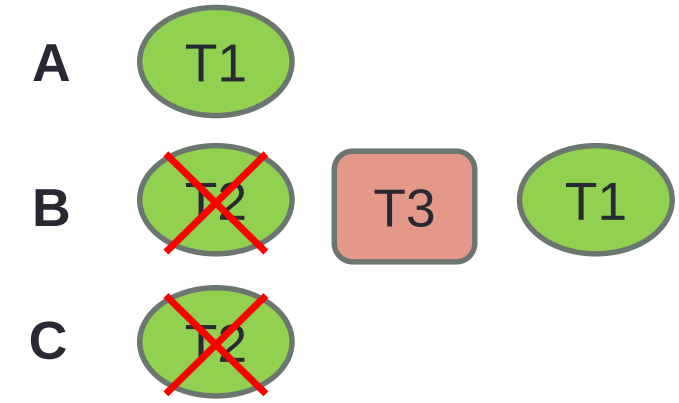
Schedule

T1	T2	T3
R(A)		
	R(B)	
	R(C)	
	Commit	

Transações

T1	T2	T3
R(A)	R(B)	W(B)
R(B)	R(C)	R(A)

Hashtable



Exemplo

Agora T3 pode obter lock de escrita em B e efetuar operação

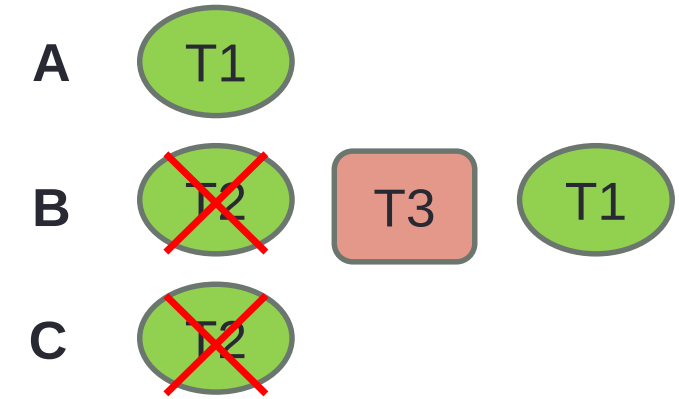
Schedule

T1	T2	T3
R(A)		
	R(B)	
	R(C)	
	Commit	
		W(B)

Transações

T1	T2	T3
R(A)	R(B)	W(B)
R(B)	R(C)	R(A)

Hashtable



Exemplo

T1 aguarda liberação do item de dados B

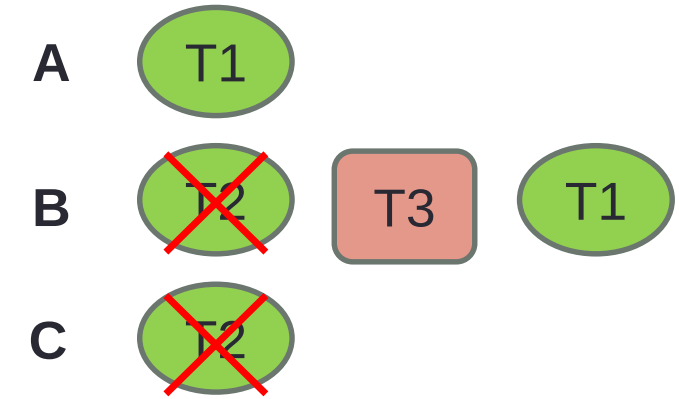
Schedule

T1	T2	T3
R(A)		
	R(B)	
	R(C)	
	Commit	
		W(B)

Transações

T1	T2	T3
R(A)	R(B)	W(B)
R(B)	R(C)	R(A)

Hashtable



Exemplo

T3 obtém lock de leitura em A e efetua a operação

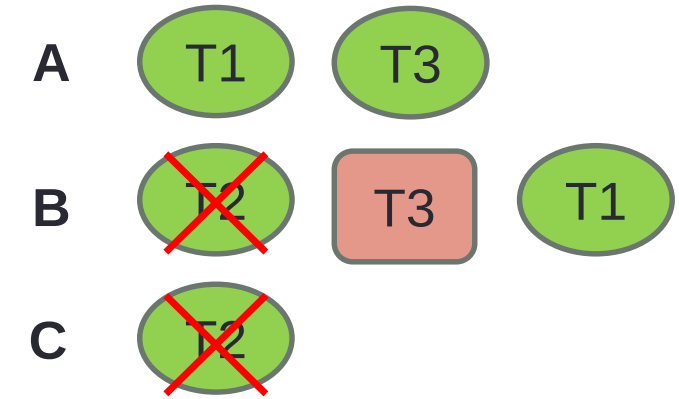
Schedule

T1	T2	T3
R(A)		
	R(B)	
	R(C)	
	Commit	
		W(B)
		R(A)

Transações

T1	T2	T3
R(A)	R(B)	W(B)
R(B)	R(C)	R(A)

Hashtable



Exemplo

T1 aguarda liberação do item de dados B

Schedule

T1	T2	T3
R(A)		
	R(B)	
	R(C)	
	Commit	
		W(B)
		R(A)

Transações

T1	T2	T3
R(A)	R(B)	W(B)
R(B)	R(C)	R(A)

Hashtable

Item	Fila de espera
A	T1 T3
B	~~T2~~ T3 T1
C	~~T2~~

Exemplo

T3 pode comitar

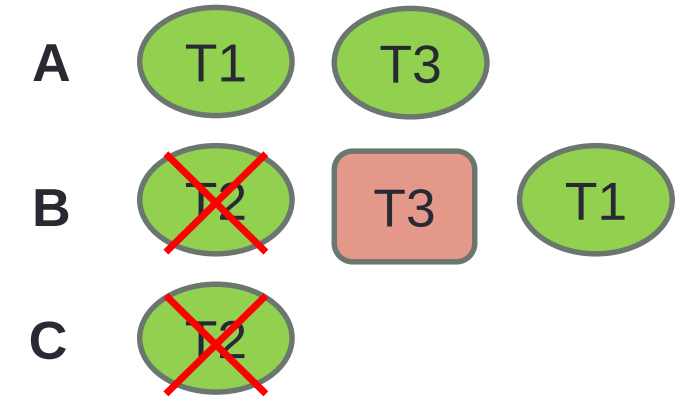
Schedule

T1	T2	T3
R(A)		
	R(B)	
	R(C)	
	Commit	
		W(B)
		R(A)
		Commit

Transações

T1	T2	T3
R(A)	R(B)	W(B)
R(B)	R(C)	R(A)

Hashtable



Exemplo

Os itens bloqueados por T3 são liberados

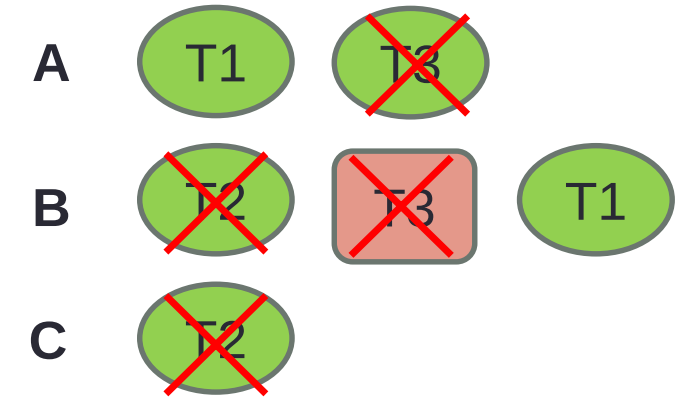
Schedule

T1	T2	T3
R(A)		
	R(B)	
	R(C)	
	Commit	
		W(B)
		R(A)
		Commit

Transações

T1	T2	T3
R(A)	R(B)	W(B)
R(B)	R(C)	R(A)

Hashtable



Exemplo

Finalmente T1 obtém lock de leitura em B e efetua a operação

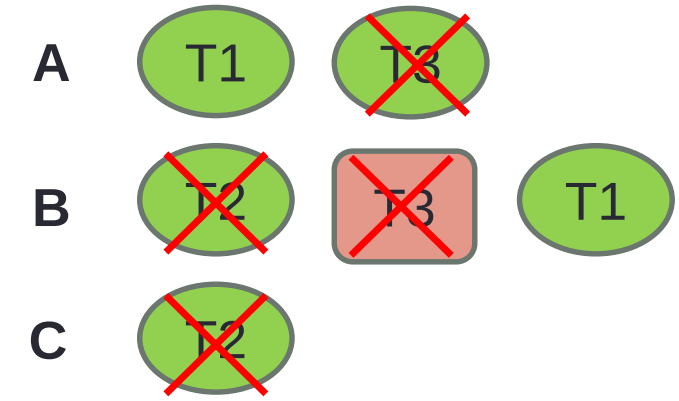
Schedule

T1	T2	T3
R(A)		
	R(B)	
	R(C)	
	Commit	
		W(B)
		R(A)
		Commit
R(B)		

Transações

T1	T2	T3
R(A)	R(B)	W(B)
R(B)	R(C)	R(A)

Hashtable



Exemplo

T1 pode comitar

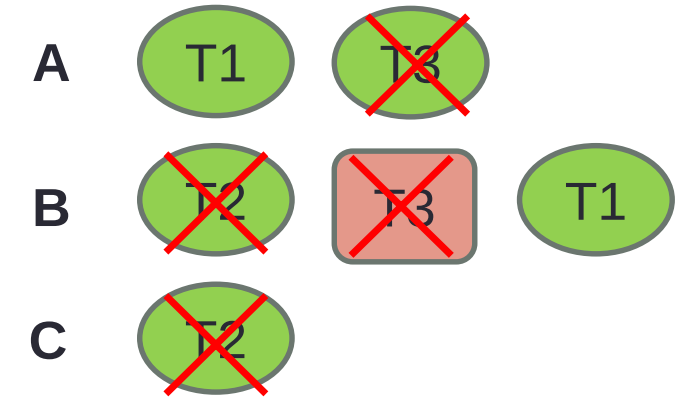
Schedule

T1	T2	T3
R(A)		
	R(B)	
	R(C)	
	Commit	
		W(B)
		R(A)
		Commit
R(B)		
Commit		

Transações

T1	T2	T3
R(A)	R(B)	W(B)
R(B)	R(C)	R(A)

Hashtable



Exemplo

Os itens bloqueados por T1 são liberados

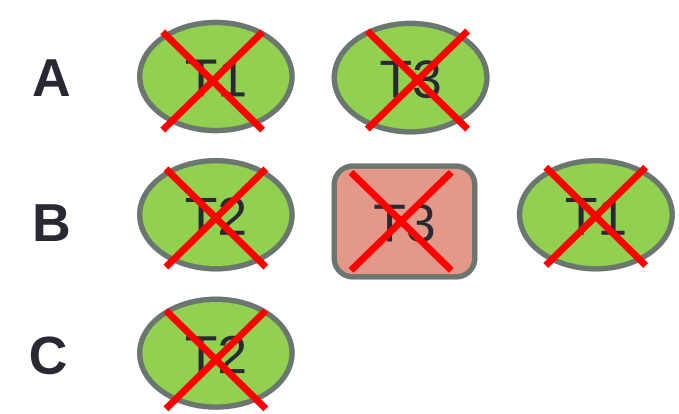
Schedule

T1	T2	T3
R(A)		
	R(B)	
	R(C)	
	Commit	
		W(B)
		R(A)
		Commit
R(B)		
Commit		

Transações

T1	T2	T3
R(A)	R(B)	W(B)
R(B)	R(C)	R(A)

Hashtable



Código Fonte

- O SGBD criado para a disciplina oferece um ambiente para simular a execução de transações concorrentes
- Classes que compõe o ambiente
  - **Transaction**: uma transação
  - **Instruction**: uma instrução de uma transação
  - **Item**: um item de dados (um registro de uma tabela)
  - **Lock**: uma transação na fila de espera para acessar um item
  - **LockTable**: a tabela de bloqueios de todos os itens
  - **Manager**: o gerenciador de acesso concorrente
  - **SimulatedIteractions**: simula um ambiente de concorrência composto por múltiplas transações

Classe LockTable

- O gerenciador de acesso concorrente usa uma tabela de locks (LockTable) para decidir se uma transação pode obter acesso a um item de dados

Classe LockTable

```
public class LockTable {
    private Hashtable<String, Item> itens = new Hashtable<>();

    …
}
```

```
public class Item {

    public Table table;
    public long primaryKey;
    public LockTable lockTable;
    ArrayList<Lock> locks = new ArrayList<>();

    …
}
```

```
public class Lock {
    Transaction transaction;
    int mode;
     …
}
```

Classe LockTable

- Principais funções da LockTable
  - **queueTransaction:** adiciona uma transação na fila de espera de um item
  - **removeTransaction**: remove uma transação da tabela de bloqueios

Classe LockTable

- A função **queueTransaction** adiciona a transação que possui a instrução instruction na fila de espera para acessar o item de dados referenciado pela instrução
- O retorno é a transação que precisa ser abortada caso haja algum problema
  - Retorna null se nenhuma transação precisa ser abortada

```
public Transaction queueTransaction(Instruction instruction) {
        Item item = getItem(instruction);
        return item.addToQueue(instruction.getTransaction(), instruction);
    }
```

Classe LockTable

- A função **queue**
  possui a instruçã
  acessar o item d
- O retorno é a tra
  ao enfileiramento
  - Retorna null se nenhuma transação precisa ser abortada

```
public Transaction addToQueue(Transaction t, Instruction i) {
        if (!alreadyInQueue(t, i.getMode())) {
            Lock l = new Lock(t, i.getMode());
            locks.add(l);
            i.setItem(this);
}
return null;
 }
```

```
public Transaction queueTransaction(Instruction instruction) {
        Item item = getItem(instruction);
        return item.addToQueue(instruction.getTransaction(), instruction);
    }
```

Classe LockTable

- A função **removeTransaction** remove a transação da tabela de bloqueios
- A transação é removida de todos os itens

```
public void removeTransaction(Transaction t){
    Iterator<Item> it = itens.values().iterator();
        while (it.hasNext()){
            Item item = it.next();
            item.removeTransaction(t);
        }
}
```

Classe LockTable

- A função **removeTransaction** remove a transação da tabela de bloqueios
- A transação é removida de todos os itens

```
public void removeTransaction(Transaction t) {
        for (int i = locks.size() - 1; i >= 0; i--) {
            Lock lock = locks.get(i);
            if (lock.transaction.equals(t)) {
                locks.remove(i);
            }
        }
    }
```

```
public void removeTransaction(Transaction t){
    Iterator<Item> it = itens.values().iterator();
        while (it.hasNext()){
            Item item = it.next();
            item.removeTransaction(t);
        }
}
```

Tópicos

- Mecanismo de Lock
- Protocolo Two-Phase Locking
- Protocolo Strict Two-Phase Locking
- Conversões de Lock
- Aquisição/Liberação Automática de Locks
- Protocolo Rigorous Two-Phase Locking
- Implementação do Locking
- Problemas a tratar com o Protocolo Two-phase Locking
- Protocolos Baseados em Grafo

Problemas a tratar com o Protocolo Two-phase Locking

- Mesmo usando o protocolo rigoroso, os problemas abaixo podem ocorrer
  - Deadlocks
  - Starvation

Two-Phase Locking e Deadlocks

- No schedule ao lado, nenhuma das transações pode prosseguir
- T1 aguarda um recurso alocado por T2
- T2 aguarda um recurso alocado por T1
- Essa situação é chamada de **deadlock**

T1	T2
lock-X*(A)*;	
write (A)	
	lock-X*(B)*;
	write (B)
lock-X*(B)*; (travou)	
	lock-X*(A)*; (travou)

Two-Phase Locking e Deadlocks

- Caso um deadlock ocorra:
  - uma das duas transações deve ser abortada e revertida
  - e ter seus locks liberados.

T1	T2
lock-X*(A)*; **write (A)**	
	lock-X*(B)*; **write (B)**
lock-X*(B)*; (travou)	
	lock-X*(A)*; (travou)

Two-Phase Locking e Starvation

- **Starvation** ocorre quando uma transação demora muito para receber acesso a um item
  - Porque a prioridade é dada para outras transações que querem acessar o mesmo item
  - Enquanto as outras transações usam o item, a transação preterida “passa fome” (starve)

Problemas Adicionais com o Protocolo Two-phase Locking

- Deadlock
  - O potencial para deadlocks existe na maioria dos protocolos.
  - O custo para evitá-los é muito alto
  - Por isso, são considerados um mal necessário.
- Starvation
  - Os gerenciadores de controle de concorrência devem ser projetados de forma a prevenir starvation.
  - Como isso poderia ser feito?

Tópicos

- Mecanismo de Lock
- Protocolo Two-Phase Locking
- Protocolo Strict Two-Phase Locking
- Conversões de Lock
- Aquisição/Liberação Automática de Locks
- Protocolo Rigorous Two-Phase Locking
- Implementação do Locking
- Problemas a tratar com o Protocolo Two-phase Locking
- Protocolos Baseados em Grafo

Protocolo Two-Phase Locking

- Existem protocolos melhores que o de duas fases
  - Em algumas características
- Entretanto, eles dependem de informações extras

- Por exemplo, os **protocolos baseados em grafo**
  - Nesses protocolos, a informação extra seria a ordem de acesso aos itens (representada como um grafo)

Protocolos Baseados em Grafo

- Impõe uma ordem parcial no conjunto **D** = {d_1, d_2 ,..., d_h} de todos itens de dados.
  - se $d_i \rightarrow d_j$ então todas transações acessando tanto d_i como d_j devem acessar d_i antes de acessar d_j.
  - Implica que o conjunto **D** pode agora ser visto como um grafo acíclico direcionado, chamado de *grafo do banco*.

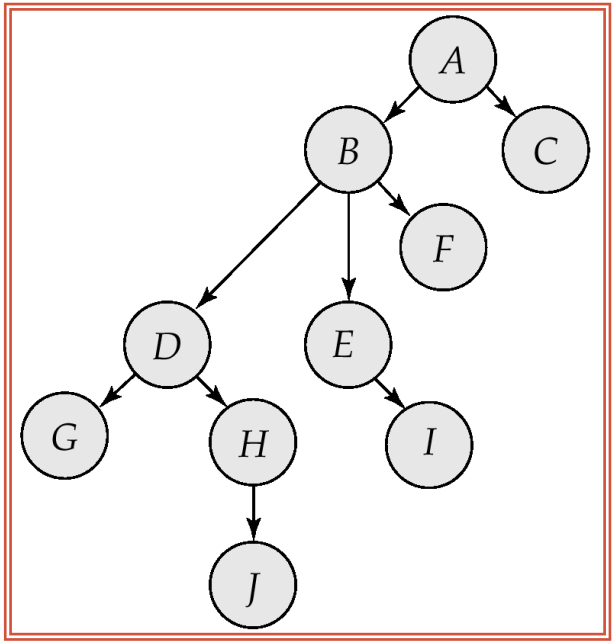
- O *protocolo de árvore* é um tipo simples de protocolo de grafo.

Protocolo de Árvore

1. Apenas locks exclusivos são permitidos.

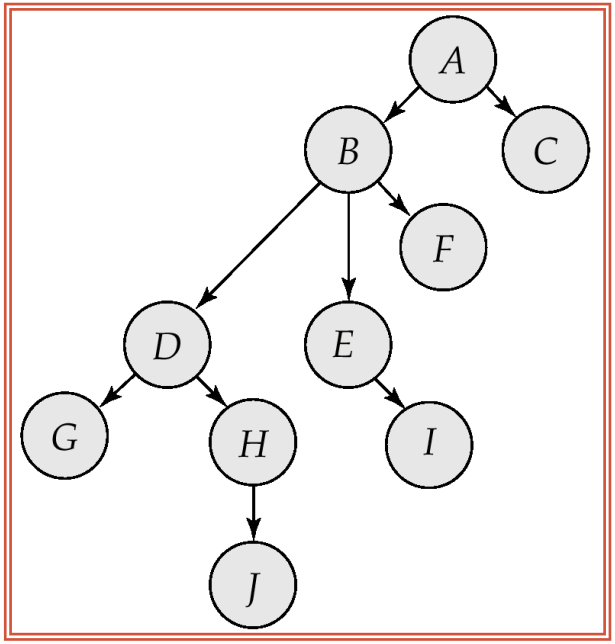
2. O primeiro lock de T_i pode ser sobre qualquer item.

3. Depois, um item Q pode sofrer lock por T_i apenas se o pai de Q tiver sofrido lock por T_i.



Protocolo de Árvore

- Ex. A transação precisa usar primeiro D e depois I durante o processamento
  - Os bloqueios devem ocorrer sobre os itens B, D, E, I (nessa ordem)



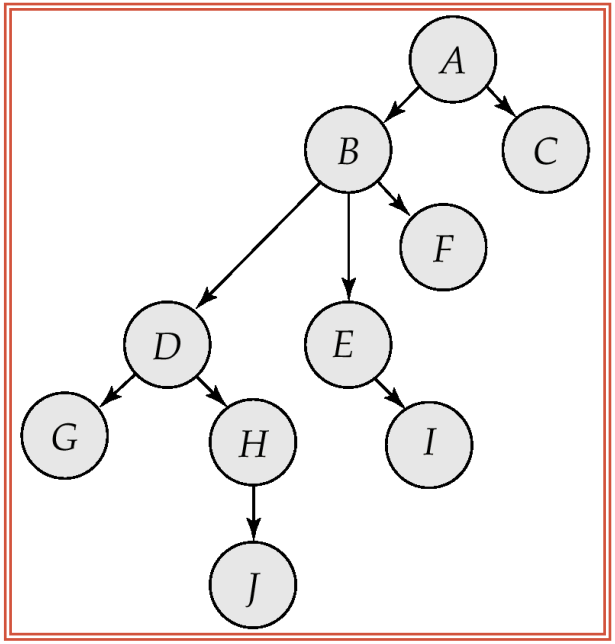
Protocolo de Árvore

- Vantagens
  - O protocolo de árvore garante serialização de conflito
  - Previne deadlocks.
    - Não recorre a rollbacks

- Desvantagens (mostradas nos próximos slides)
  - Reduz a concorrência devido ao número grande de bloqueios
  - Dependem de informações que normalmente não se conhece

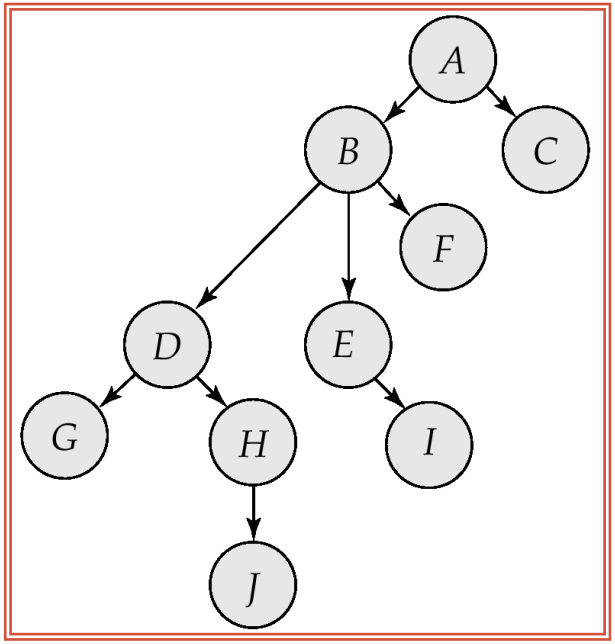
Protocolo de Árvore

- Ex. A transação precisa usar primeiro D e depois I durante o processamento
  - Os bloqueios devem ocorrer sobre os itens B, D, E, I (nessa ordem)
- Problemas
  - A transação deve bloquear itens que não irá usar (B, E)
    - Isso reduz a concorrência



Protocolo de Árvore

- Ex. A transação precisa usar primeiro D e depois I durante o processamento
  - Os bloqueios devem ocorrer sobre os itens B, D, E, I (nessa ordem)



- Problemas
  - Para saber qual o item a bloquear primeiro (B)
    - o gerenciador precisa conhecer todos os itens que a transação irá precisar
    - essa informação normalmente não está disponível

Atividade Individual

- Com base nas transações do próximo slide
  - Monte um Schedule de execução que obedeça ao protocolo Two-Phase Locking.
  - Mostre a tabela de locks usada para controlar a aquisição e liberação de acesso aos recursos requisitados pelas transações.

- Considere ciclos de execuções, onde em cada ciclo todas transações tem o direito de executar uma instrução.
  - Dentro de cada ciclo a ordem de execução é determinada pelo número da transação.

Atividade Individual

- Transações

T1:	T2:	T3:	T4:	T5:
read(A); **write(B).**	**read(D);** **read(B);** **write(C);** **read(H).**	**write(D);** **read(E);** **read(B).**	**read(F);** **read(G);** **read(A).**	**write(B);** **write(F);** **read(G).**