Capítulo 14: Controle de Concorrência

- Protocolos Baseados em Bloqueio
- Deadlock
- 2PL
- Granularidade Múltipla

Protocolos Baseados em Bloqueio

- Bloqueio é um mecanismo de controle de acesso concorrente a um item de dados
- Dados podem ser bloqueados de 2 modos:

  1. *exclusivo (X)*. Dado pode ser tanto lido como escrito. É feito usando-se a instrução **lock-X**.

  2. *compartilhado (S)*. Dado somente pode ser lido. É feito usando-se a instrução **lock-S**.
- Pedidos de bloqueio (*lock*) são feitos ao gerente de controle de concorrência.
- Transação só pode prosseguir após seu pedido ser atendido.

Protocolos Baseados em Bloqueio (Cont.)

- Matriz de compatibilidade de bloqueio (*Lock-compatibility matrix*)

	S	X
S	true	false
X	false	false

- Transação somente pode conseguir um bloqueio de um item se o mesmo for compatível com os atuais bloqueios já obtidos por outras transações
- Qualquer número de transações pode compartilhar bloqueios sobre um item, mas se qualquer transação possui um bloqueio exclusivo nenhuma outra transação pode obter qualquer bloqueio nele.
- Se um bloqueio não pode ser concedido, a transação solicitante espera até que todos bloqueios incompatíveis tenham sido liberados. Só depois o bloqueio é concedido.

Protocolos Baseados em Bloqueio (Cont.)

- Exemplo de transação fazendo bloqueio:

  T_2: **lock-S**(*A*);
  read (*A*);
  unlock(*A*);
  lock-S(*B*);
  read (*B*);
  unlock(*B*);
  display(*A*+*B*)

- Locking como acima não é suficiente para se garantir a serialização — se *A* e *B* forem atualizadas entre o read de *A* e *B*, a soma mostrada será errada.
- Um **protocolo de bloqueio (*locking protocol*)** é um conjunto de regras seguidas por todas transações enquanto pedindo e liberando bloqueios. Restringe o número de possíveis escalas.

Falhas de Protocolos Baseados em Bloqueio

- Considere o schedule parcial abaixo:

T_3	T_4
lock-X(B)	
read(B)	
$B := B - 50$	
write(B)	
	lock-S(A)
	read(A)
	lock-S(B)
lock-X(A)	

- Nem T_3 ou T_4 podem prosseguir — executar **lock-S**(*B*) causa T_4 esperar por T_3 liberar seu bloqueio em *B*, enquanto executar **lock-X**(*A*) causa T_3 esperar por T_4 liberar seu bloqueio em *A*.

Deadlock

- Sistema está em *deadlock* (nó cego) se há um conjunto de transações, tal que toda transação no conjunto estiver esperando por outra transação nesse conjunto.
- Potencial para *deadlock* existe na maioria dos protocolos de bloqueio. São um mal necessário.
- **Prevenção de *Deadlock,*** há 2 abordagens para garantir que o sistema não entre em estado de *deadlock*:
  - ★ Requerer que cada transação bloqueie todos os itens de dados antes de começar a execução (predeclaração).
  - ★ Fazer que a transação seja refeita, em vez de esperar um bloqueio, sempre que potencialmente possa ocorrer um *deadlock*

Recuperação de *Deadlock*

- Quando *deadlock* é detectado 3 ações devem ser tomadas:
  - Selecionar alguma transação para *rollback* (vitima) e com isso se quebrar o *deadlock*. Deve-se escolher a de menor custo.
  - Determinar até que ponto fazer *rollback*
    - *Rollback* total: abortar a transação toda e então reiniciá-la.
    - *Rollback* parcial: é mais efetivo fazer *rollback* da transação somente o necessário para quebrar o *deadlock*.
  - Tratar inanição (*starvation*) que ocorre se a mesma transação é sempre a escolhida. Incluir o número de vezes que sofreu *rollback* no fator de custos, para garantir que não seja mais escolhida.
- Exemplo de mensagem de erro:
  - Transaction (Process ID 81) was deadlocked on lock resources with another process and has been chosen as the deadlock victim. Rerun the transaction. [SQLSTATE 40001] (Error 1205). The step failed.

Tempo esgotado

- Esquema com base em tempo esgotado (*Timeout-Based Schemes*):
  - ★ transação espera por um bloqueio somente um tempo especificado;
  - ★ se não conseguir, seu tempo está esgotado e ela é revertida e pode ou não ser reiniciada automaticamente;
  - ★ simples de implementar; contudo a inanição é possível;
  - ★ também é difícil determinar um bom valor de *timeout*, se for grande a espera é longa e se for curto há muito *rollback*;
  - ★ alguma coisa entre prevenção de *deadlock*, dado que o mesmo não ocorre, e detecção e recuperação.
- Exemplo de mensagem de erro:
  - ★ Error: SQL0911N The current transaction has been rolled back because of a deadlock or timeout. Reason code "2".

Protocolo *Two-Phase Locking*

- O protocolo de **bloqueio em duas fases** (2PL) garante escalas serializáveis sem conflito.
- Fase 1: *Growing Phase* (crescimento)
  - ★ transação pode obter bloqueios
  - ★ transação não pode liberar bloqueios
- Fase 2: *Shrinking Phase* (encolhimento)
  - ★ transação pode liberar bloqueios
  - ★ transação não pode obter bloqueios

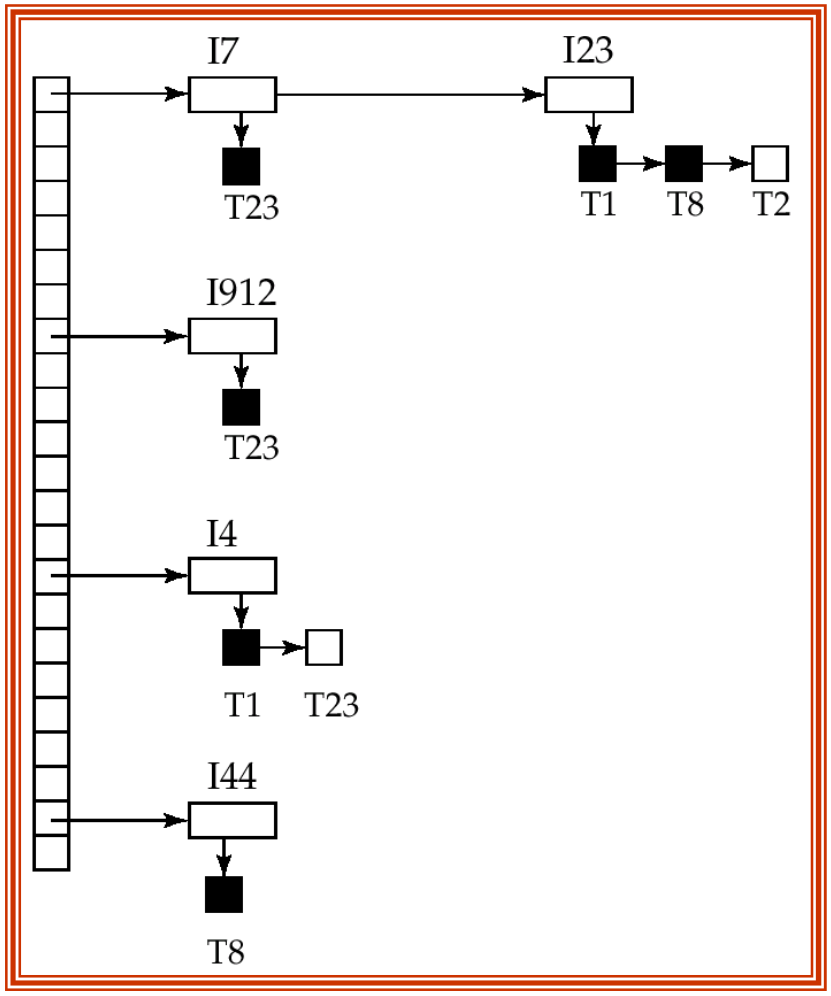
Protocolo *Two-Phase Locking*

- *Two-phase locking* **não garante** ausência de *deadlocks*
- Efeito cascata *(Cascading roll-back)* , uma transação cai e derruba outra e assim vai, também é possível sob 2PL.
- Para se evitar a cascata, há duas opções de protocolo modificado:
  - ⋆ **Bloqueio em duas fases severo (*strict two-phase locking*)**. Nele uma transação deve manter todos os seus bloqueios **exclusivos** até que commit/aborte.
  - ⋆ **Bloqueio em duas fases rigoroso (*rigorous two-phase locking*)** é ainda mais restrito, **todos** bloqueios são mantidos até que a transação commit/aborte. Nesse protocolo transações podem ser serializadas na ordem de sua efetivação (commit).

Implementação de Locking

- Um **Gerente de Bloqueio** (*lock manager*) pode ser implementado como um processo separado para o qual transações enviam pedidos de *lock* e *unlock*
- Ele responde a um pedido de bloqueio enviando uma mensagem de concessão de bloqueio (ou uma mensagem pedindo que a transação faça *roll back*, em caso de *deadlock*)
- A transação requisitante espera até seu pedido ser respondido
- O gerente mantém uma estrutura de dados chamada **tabela de bloqueios *(lock table)*** que grava os bloqueios concedidos e os pedidos pendentes
- A tabela de bloqueio usualmente é implementada em-memória, indexada pelos nomes dos itens de dados bloqueados

Lock Table

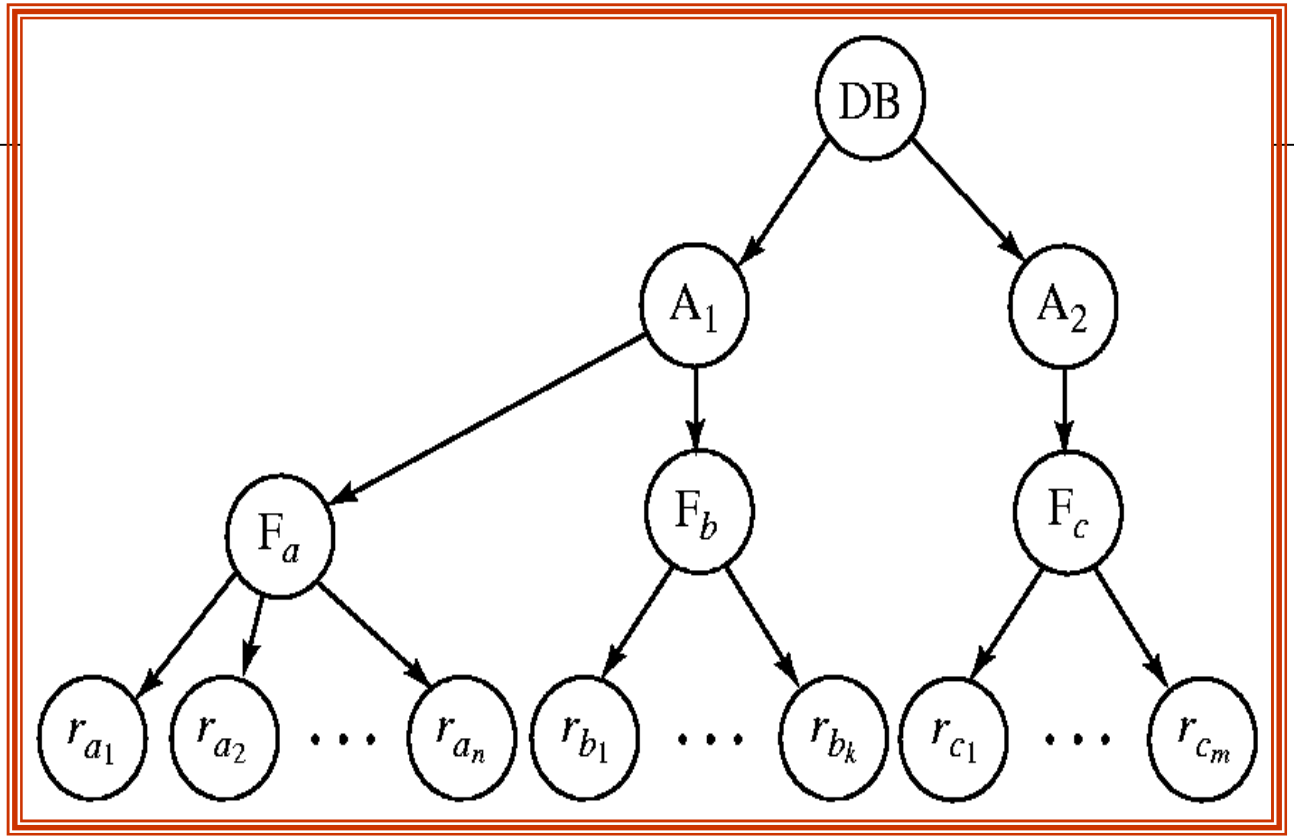


- Retângulos escuros indicam bloqueios concedidos (*granted locks*), os claros indicam pedidos em espera
- Registra também o tipo de bloqueio envolvido
- Novo pedido é adicionado ao final da fila de pedidos para o item de dados, e granted se for compatível com todos bloqueios prévios
- Pedidos de desbloqueio resultam no pedido ser deletado, e pedidos anteriores são checados se podem agora ser granted
- Se a transação abortar, todos seus pedidos, pendentes ou concedidos, são deletados
  - ★ lock manager pode manter uma lista de bloqueios mantidos por cada transação, para implementar isso com mais eficiência

Granularidade Múltipla

- Permite itens de dados serem de vários tamanhos e define a hierarquia de granularidade dos dados, onde menores granularidades são aninhadas dentro das maiores
- Pode ser representado graficamente como uma árvore
- Quando a transação bloqueia um nó na árvore *explicitamente*, ela *implicitamente* bloqueia todos os nós descendentes no mesmo modo.
- Granularidade de bloqueio (nível na árvore onde é feito o locking):
  - ★ *fina granularidade* (nível mais baixo da árvore): alta concorrência, alto *locking overhead*
  - ★ *grossa granularidade* (nível mais alto da árvore): baixo *locking overhead*, baixa concorrência

Exemplo de Hierarquia de Granularidade



O mais alto nível na hierarquia de exemplo é o database inteiro.

Os níveis abaixo são do tipo *area*, *file* e *record* nessa ordem.

Fim de Capítulo