MC536

Sumário

- Objetivos
- Método baseado em trancas
- Método baseado em timestamps
- Protocolos baseados em múltiplas versões
- Certificação
- Questão de Granularidade

Sumário

- Objetivos
- Método baseado em trancas
- Método baseado em timestamps
- Protocolos baseados em múltiplas versões
- Certificação
- Questão de Granularidade

Objetivos

- Escalonar operações de forma a produzir históricos corretos (históricos serializáveis)
- Garantir isolamento de transações conflitantes
- Preservar consistência de banco de dados
- Resolver conflitos *read-write* e *write-write*

- Opções dos mecanismos de controle de concorrência após receber uma operação:
  - executá-la imediatamente
  - atrasá-la bloqueando transação
  - rejeitá-la abortando transação

Sumário

- Objetivos
- Método baseado em trancas
- Método baseado em timestamps
- Protocolos baseados em múltiplas versões
- Certificação
- Questão de Granularidade

Trancas (locks)

- associa um *lock* a cada item
- transação
  - requisita *lock* antes de acessar um item
  - libera *lock* após utilização
- Verifica compatibilidade entre locks já existentes e lock requisitado:
  - concede *lock* se compatível com locks já existentes
  - bloqueia transação se incompatível

Trancas – operações

- Lock: operação que garante
  - permissão para ler ou
  - permissão para escrever um item de dado para uma transação
  - Exemplo: lock_item (X)
    - Item de dado X é "trancado" pela transação que solicitou tranca
- Unlock: operação que remove estas permissões de um item de dado
  - Exemplo: unlock_item(X)
    - Item de dado X fica disponível para outras transações

Trancas - operações

- Lock upgrade:
  - Tranca de leitura para tranca de escrita

```
if Ti has a read-lock (X) and Tj has no read-lock (X) (i ≠ j) then
        convert read-lock (X) to write-lock (X)
else
        force Ti to wait until Tj unlocks X
```

- Lock downgrade:
  - Tranca de escrita para tranca de leitura

```
Ti has a write-lock (X) (*no transaction can have any lock on X*)
convert write-lock (X) to read-lock (X)
```

Trancas- modos

- compartilhado (leitura)
  - read_lock (X)
  - Mais de uma transação pode requerer uma tranca compartilhada para ler um item X, mas nenhuma tranca de escrita pode ser requerida por nenhuma outra transação
- exclusivo (escrita)
  - write_lock (X)
  - Somente uma tranca de escrita de um item X pode existir e nehuma tranca compartilhada pode ser requerida por outra transação

	Read	Write
Read	Y	N
Write	N	N

Trancas – gerenciador de trancas

- gerencia trancas em itens de dados
- usa tabela de trancas para armazenar identificadores de transações que trancam itens, informação sobre o item trancado, modo de trancamento e um ponteiro para o próximo item trancado

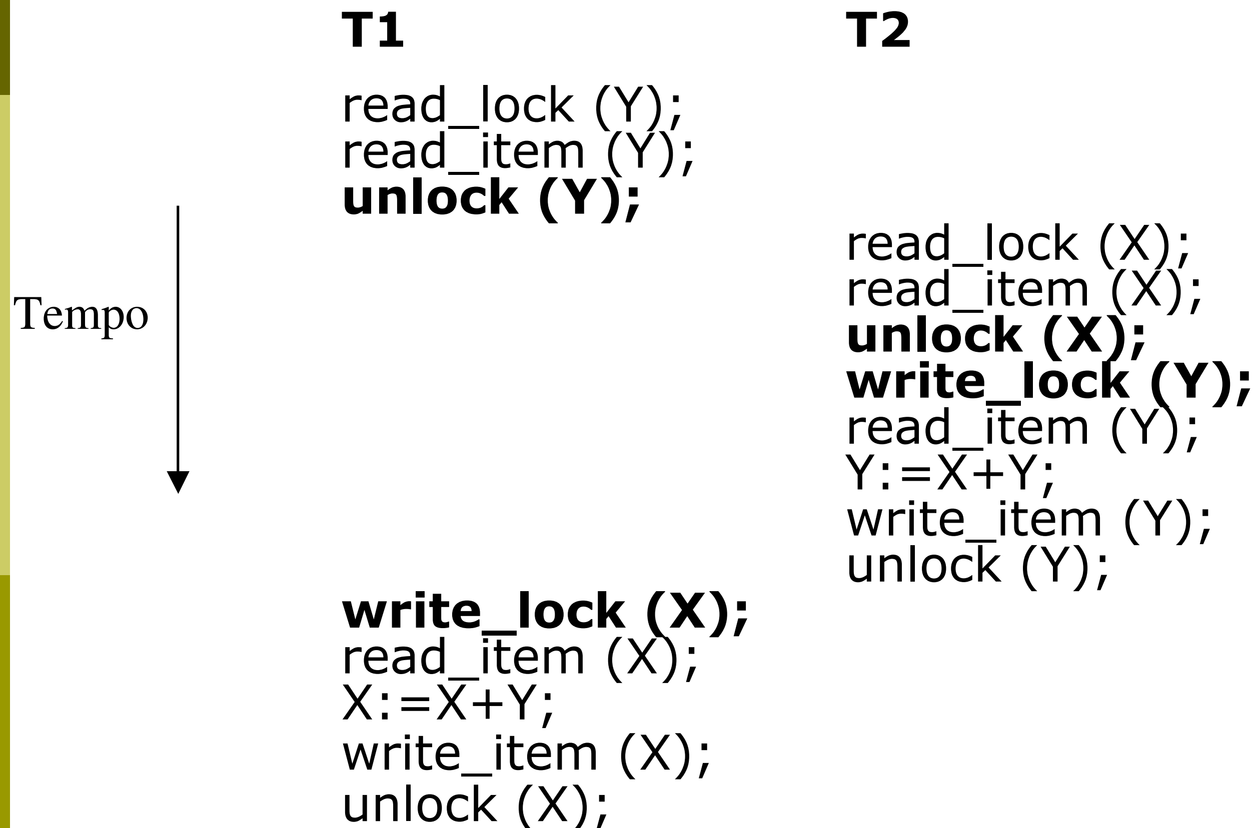
Transaction ID	Data item id	lock mode	Ptr to next data item
T1	X1	Read	Next

Trancas – garantia de serialização

- transações devem ser *bem formadas*
  - Transações devem trancar o dado antes de lê-lo ou escrevê-lo
  - Não devem trancar um item de dado já trancado e não deve destrancar itens livres

- transações devem ter *duas fases*
  - Crescimento: obtenção de tranca
    - Uma transação obtém trancas (de leitura ou escrita) sobre itens de dados
  - Encolhimento: liberação de trancas
    - Uma transação libera trancas sobre itens de dados
  - Requisito: para uma transação, estas duas fases devem ser mutuamente exclusivas, isto é, durante a fase de obtenção de trancas, nenhuma tranca deve ser liberada; durante a liberação, nenhuma tranca deve ser obtida.

Exemplo de uso de trancas



Não-serializável
não é 2 fases

Exemplo de uso de trancas

T1	**T2**
read_lock (Y);	
read_item (Y);	
	read_lock (X);
	read_item (Y);
write_lock (X);	
(espera por X)	write_lock (Y);
	(espera por Y)

Respeitam 2PL, mas estão em deadlock

Algoritmos de tranca Two-Phase Locking (2PL)

- Básico
  - Tranca itens de dados incrementalmente; pode causar deadlock
- Conservativo
  - Tranca todos os itens que a transação precisa antes de a transação iniciar; previne deadlock
- Restrito
  - básico; destrancas são feitas quando a transação termina (efetiva, aborta); algoritmo mais utilizado

Prevenção de deadlock

- timeout
  - Dificuldade: determinar intervalo
- pré-declaração de recursos (2PL conservativo)
  - Desvantagem: diminui concorrência
- pré-ordenação de recursos
  - Desvantagem: diminui concorrência

Prevenção de Deadlocks

- priorização de transações: transações recebem timestamp conforme a ordem em que são criadas.
  - wait-die: Uma transação mais velha pode esperar por uma transação mais nova

    Ti requisita lock

    se Ti mais velha que Tj

    então Ti espera

    senão Ti é cancelada
  - wound-wait

    Ti requisita lock

    se Ti mais nova

    então Ti espera

    senão Tj é cancelada

Detecção de deadlock

- O escalonador mantém um grafo de espera (*wait-for-graph*) para detectar ciclos
  - Quando uma transação é bloqueada, ela é incluída no grafo
  - Se um ciclo existe, então uma transação (vítima) é selecionada para ser cancelada
- Quando procurar por ciclos?
  - Critérios: número de transações executando, tempo em que um conjunto de transações está esperando, a cada intervalo de tempo, quando uma aresta ou certo número de arestas é inserido
- Como selecionar a vítima?
  - Critérios: evitar matar a que está executando há mais tempo; ou executou muitas atualizações; escolher a que está envolvida em mais deadlocks

Problemas do método de locks

- Liberação de trancas só ocorre no final da transação para evitar cancelamentos em cascata
- Ocorrência de deadlocks
- Inadequado para aplicações de longa duração
  - espera
  - aumento do número de *deadlocks*

Sumário

- Objetivos
- Método baseado em trancas
- Método baseado em timestamps
- Protocolos baseados em múltiplas versões
- Certificação
- Questão de Granularidade

Ordenação por timestamps

- Requisitos
  - cada transação tem um timestamp
    - Variável inteira monotonicamente crescente que indica a idade de uma transação
  - cada operação deve ser acompanhada do TS da transação
  - cada item de dado tem associado:
    - TS de leitura: maior timestamp entre todas os timestamps de transações que leram X
    - TS de escrita
- Histórico resultante equivale ao serial ordenado por ordem de TS

Ordenação por timestamps

- Algoritmo Básico
  - TS é o timestamp da operação
  - leitura

    se TS < WTS rejeita

    senão executa (e atualiza RTS, RTS=max{TS,RTS})
  - escrita

    se TS < RTS ou TS < WTS rejeita

    senão executa (e atualiza WTS, WTS=TS)

Ordenação por timestamps

- Algoritmo Estendido (isolamento)
  - pré-escrita

    se TS < RTS ou TS < WTS rejeita

    senão executa
  - leitura

    se TS < WTS rejeita

    senão se TS > min(PTS) atrasa

    senão executa
  - escrita

    se TS > min(PTS) atrasa

    senão executa

Sumário

- Objetivos
- Método baseado em trancas
- Método baseado em timestamps
- Protocolos baseados em múltiplas versões
- Certificação
- Questão de Granularidade

Protocolos baseados em múltiplas versões

- Esta abordagem mantém várias versões para um mesmo item de dado
- Operações de leitura não são rejeitadas

- Efeito colateral:
  - mais espaço de armazenamento para se manter as múltiplas versões
  - necessidade de mecanismos de “coleta de lixo” (versões não usadas)

Protocolos baseados em múltiplas versões

- Assuma que X1, X2, ..., Xn são versões de item de dado X

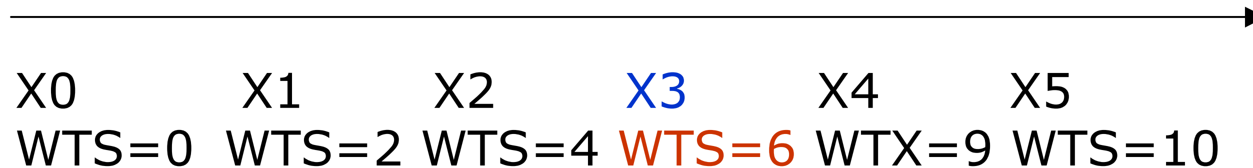
- Para cada Xi há um timestamp de leitura (read_TS) e um timestamp de escrita (write_TS)

  - read_TS(Xi): o timestamp de leitura de Xi é o maior de todos os timestamps de transações que tenham conseguido ler a versão Xi

  - write_TS(Xi): o timestamp de Xi é o timestamp da transação que escreveu a versão Xi

Protocolos baseados em múltiplas versões

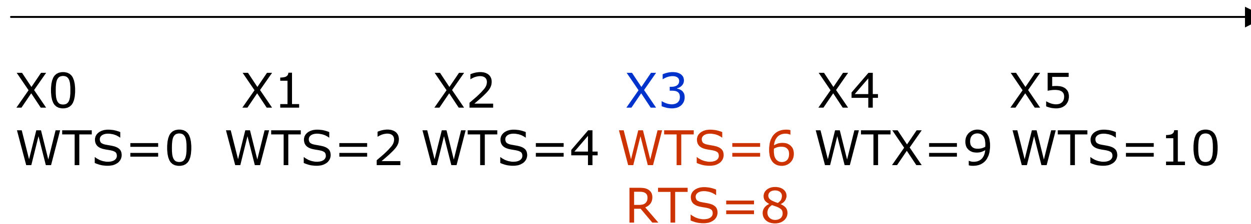
- transação T requisita read_item(X) com TS(T)=7



- Versão selecionada – X3

Protocolos baseados em múltiplas versões

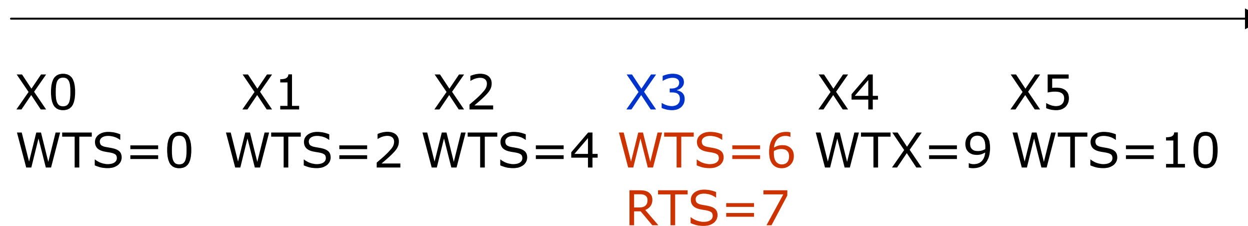
- transação T requisita write_item(X) com TS(T)=7



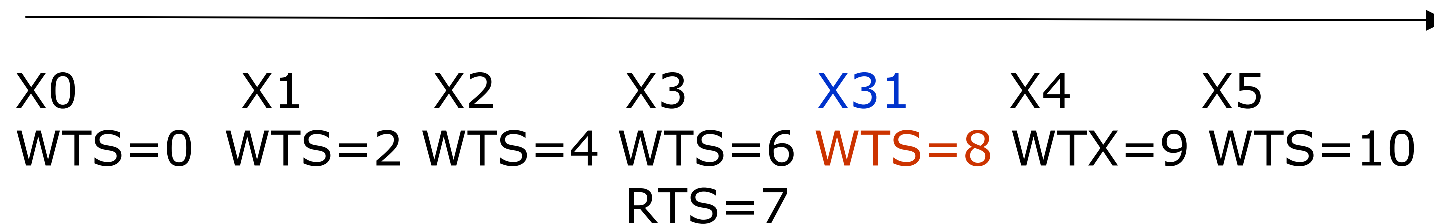
- Operação rejeitada

Protocolos baseados em múltiplas versões

- transação T requisita write_item(X) com TS(T)=8



- Nova versão é criada



Protocolos baseados em múltiplas versões

- transação T requisita write_item(X)
  - Se uma versão i de X tem o maior valor de WTS(Xi) comparado com todas as versões de X que são também menores ou iguais a TS(T), e RTS(Xi) > TS(T), então
    - aborte T;
  - caso contrário,
    - crie uma nova versão Xi
    - RTS(X) = WTS(Xj) = TS(T)
- transação T requisita read_item(X)
  - encontre a versão i de X que tenha o maior valor de WTS(Xi) de todas as versões de X que é também menor ou igual do que TS(T), então
    - retorne o valor de Xi de T
    - RTS(Xi) = max(TS(T), RTS(Xi))

Sumário

- Objetivos
- Método baseado em trancas
- Método baseado em timestamps
- Protocolos baseados em múltiplas versões
- Certificação
- Questão de Granularidade

Certificação – Protocolos Otimistas

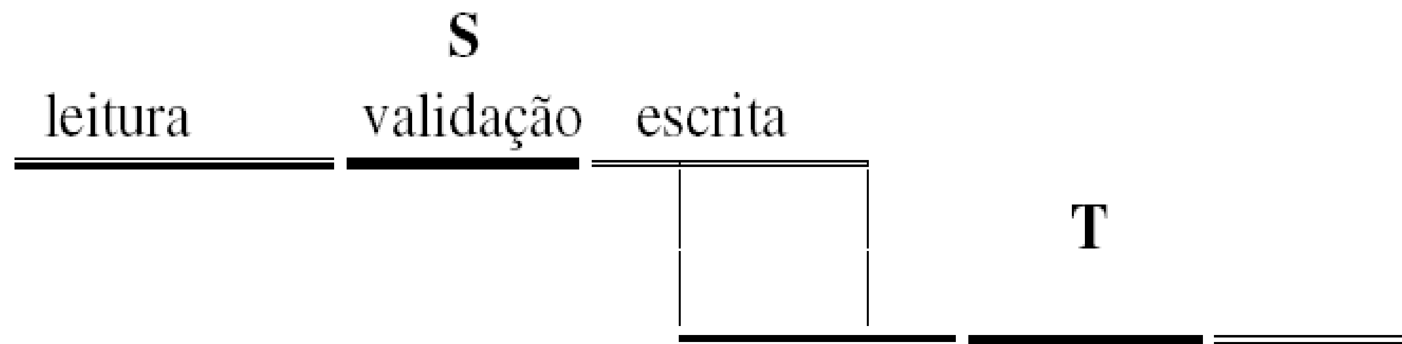
- Características
  - sincronização no final da transação
  - otimista
- Fases da Transação
  - fase de execução
    - atribuir timestamp à transação
    - determinar conjuntos de escrita e leitura da transação
  - fase de validação
  - fase de atualização

Validação – caso 1

- considerando-se S conjunto de transações, uma transação T será válida se:
  - transações em S completam fase de escrita antes de T iniciar fase de leitura

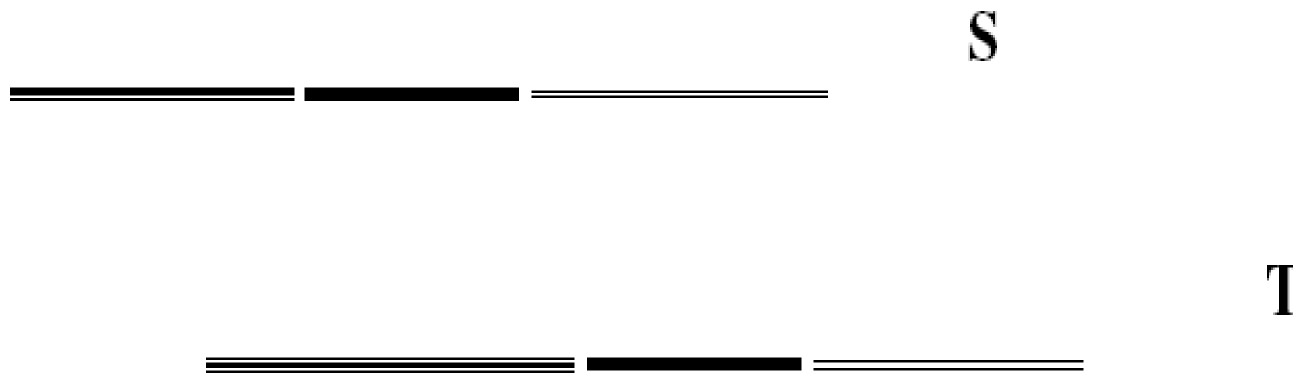
Validação – caso 2

- transações em S terminam fase de escrita antes de T iniciar fase de escrita e *não existe intersecção entre conjuntos de escrita de transações em S e conjunto de leitura de T*



Validação – caso 3

- transações em S terminam fase de leitura antes de T terminar fase de leitura e *não existe intersecção entre conjuntos de escrita de transações em S e conjunto de leitura e escrita de T*



Regras para validação

- *Considerar validação de Tj contra transações Ti concorrentes*

Ti	Tj	Regra
R	W	1. Ti não deve ler dados escritos por Tj
W	R	2. Tj não deve ler dados escritos por Ti
W	W	3. Ti não deve escrever sobre dados escritos por Tj e vice-versa

- *Considerar fases de validação e atualização atômicas*

Tipos de Validação

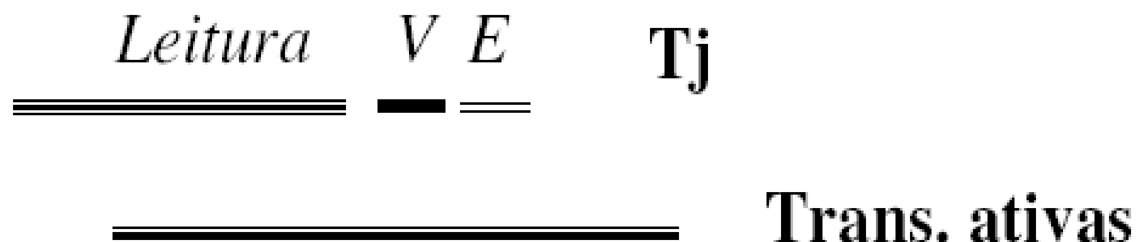
- *Backward*
  - testa Tj contra transações que já passaram por validação
- *Forward*
  - testa Tj contra transações ativas

Validação Backward

- regra 3 satisfeita: Ti não escreve sobre dados escritos por Tj
- regra 1 satisfeita: Ti não lê dados escritos por Tj
- regra 2: Tj não deve ler dados escritos por Ti
  - testar conjunto de escrita de Ti contra conjunto de leitura de Tj
- resolução de conflitos: abortar Tj

Validação Forward

- regra 3 satisfeita: Ti não escreve sobre dados escritos por Tj
- regra 2 satisfeita: Tj não lê dados escritos por Ti
- regra 1
  - testar conjunto de escrita de Tj contra conjuntos de leitura de transações ativas
- resolução de conflitos
  - abortar Tj
  - abortar transações ativas
  - atrasar Tj



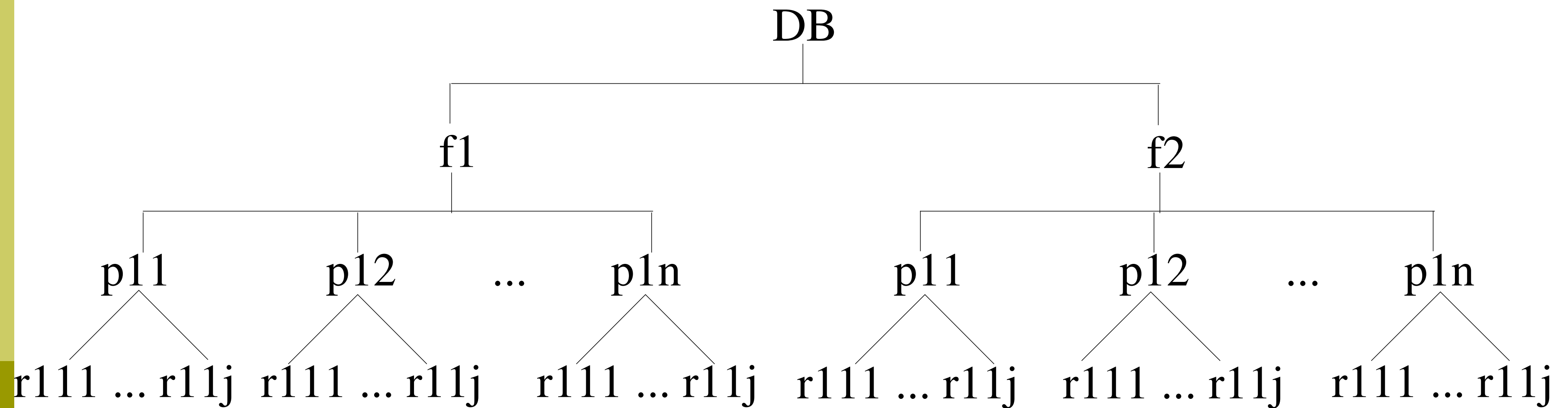
Sumário

- Objetivos
- Método baseado em trancas
- Método baseado em timestamps
- Protocolos baseados em múltiplas versões
- Certificação
- Questão de Granularidade

Questão de granularidade

- Influencia o desempenho dos mecanismos de controle de concorrência
- Quanto maior o item de dado, menor o grau de concorrência
- Exemplos de granularidade de itens:
  - Atributo de uma tupla
  - Uma tupla (registro)
  - Um bloco
  - Um arquivo
  - O BD

Questão de granularidade



Questão de granularidade

- Outros tipos de trancas são necessárias:
  - Intention-shared (IS): indica que uma tranca compartilhada vai ser requisitada para nós descendentes
  - Intention-exclusive (IX): indica que uma tranca exclusiva vai ser requisitada para nós descendentes
  - Shared-intention-exclusive (SIX): indica que o nó corrente está trancado em modo compartilhado, mas uma tranca exclusiva vai ser requisitada para alguns nós descendentes

Questão de granularidade

- Matriz de compatibilidade

	IS	IX	S	SIX	X
IS	yes	yes	yes	yes	no
IX	yes	yes	no	no	no
S	yes	no	yes	no	no
SIX	yes	no	no	no	no
X	no	no	no	no	no

Questão de granularidade

- A raiz da árvore deve ser trancada primeiro, em qualquer modo
- Um nó N pode ser trancado por uma transação T nos modos S ou IX somente se o nó pai já estiver trancado por T nos modos IS ou IX
- Um nó N pode ser trancado por T nos modos X, IX, ou SIX somente se o pai de N já estiver trancado por T no modo IX ou SIX
- T pode trancar um nó somente se ele não tenha destrancado nenhum nó (garante 2PL)
- T pode destrancar um nó N, somente se nenhum dos filhos de N estão trancados por T

Exemplo de execução

T1	T2	T3
IX(db)		
IX(f1)		
	IX(db)	
		IS(db)
		IS(f1)
		IS(p11)
IX(p11)		
X(r111)		
	IX(f1)	
	X(p12)	
		S(r11j)
IX(f2)		
IX(p21)		
IX(r211)		
Unlock (r211)		
Unlock (p21)		
Unlock (f2)		
		S(f2)

Exemplo de execução (cont.)

T1	T2	T3
	unlock(p12)	
	unlock(f1)	
	unlock(db)	
unlock(r111)		
unlock(p11)		
unlock(f1)		
unlock(db)		
		unlock (r111j)
		unlock (p11)
		unlock (f1)
		unlock(f2)
		unlock(db)