Escalonamento
&
Controle de Concorrência

Banco de Dados II

Elmasri – Capítulos 21 e 22

Ramakrishnan – Capítulos 16 e 17

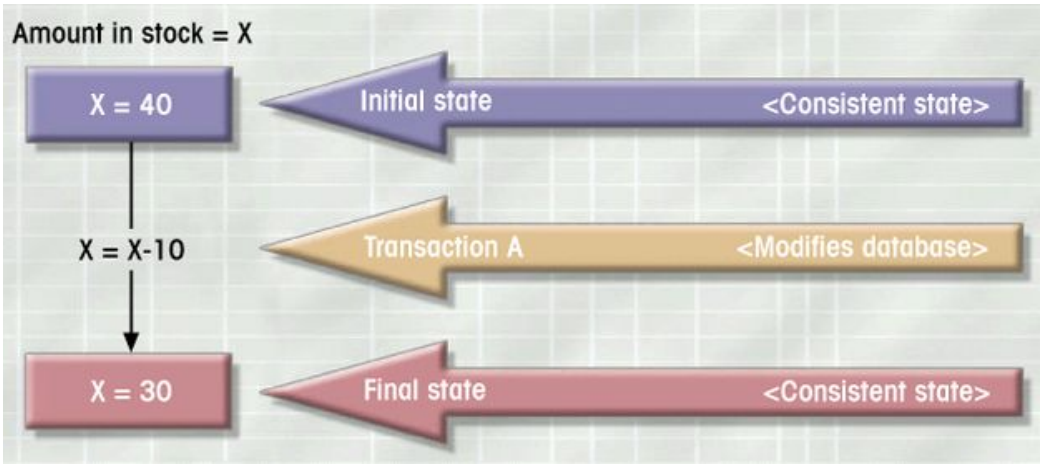
Silberchatz – Capítulos 15 e 16

Complete Book – Chapter 18

Denio Duarte

Introdução

- Revisitando transações



- Conjunto de passos executados pelo SGBD que realiza uma tarefa do usuário
- Ou executa por completo ou não executa
- Estados intermediários não são aceitáveis

Introdução

- Transações
  - Atômicas
  - Consistentes
  - Isoladas
  - Duradouras
- Uma transação termina com um *Commit* (sucesso) ou um *Rollback* (fracasso)
  - Ambos podem ser implícitos ou explícitos

Introdução

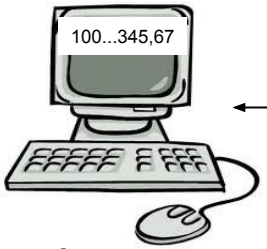
Banco antes das atualizações

João

Dados no disco

Maria

Passo 1 - SGBD lê dados do disco para a RAM de João

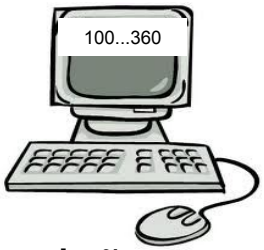


João

Dados no disco

Maria

Passo 2 – João altera os dados na RAM



João

Dados no disco

Maria

Introdução

Passo 3 - SGBD atualiza os dados com a atualização do João

Passo 4 - SGBD lê dados do disco para a RAM de Maria

Passo 5 - Maria altera os dados na RAM

Introdução

Passo 6 - SGBD atualiza os dados com a atualização da Maria

- João leu as tuplas com valores de 100…345,67 do disco e devolveu como 100... 360
- Maria leu 100... 360 e devolveu com 100 … 300

Banco consistente antes e depois

Introdução

Concorrência entre requisições

Passo 1 - SGBD lê dados do disco para a RAM de João

Passo 2 - SGBD lê dados do disco para a RAM de Maria

Introdução

Passo 3 – João altera os dados na RAM

Passo 4 – Maria altera os dados na RAM

Passo 5 - SGBD atualiza os dados com a atualização do João

Introdução

Passo 6 - SGBD atualiza os dados com a atualização da Maria

- João leu as tuplas com valores de 100 … 345,67
- Maria leu 100 … 345,67 (era para ter lido 100 … 360)
- João gravou 100 … 360
- Maria gravou 100 … 300

A atualização do João foi perdida!

Introdução

- Como resolver o problema?
  - Atualizar os dados sequencialmente no fim do expediente?
- Controle de concorrência

Introdução

- Controle de concorrência assegura o *isolamento* das transações
- Garantem a serialização dos escalonamentos das transações
  - Uso de protocolos (conjunto de regras)
- Permite a execução de várias transações em um ambiente multiusuário
- Resolve os problemas: atualizações perdidas, dados não *comitados* e consultas inconsistentes

Introdução

- Duas técnicas para garantir a concorrência
  - Bloqueios
  - Timestamps (marcas de tempo)

Introdução

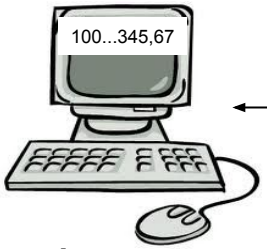
Bloqueio

João

Dados no disco

Maria

Passo 1 - SGBD lê dados do disco para a RAM de João e bloqueia

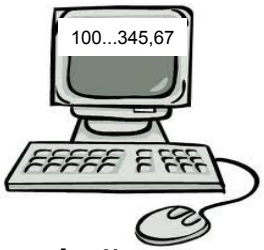


João

Dados no disco

Maria

Passo 2 - SGBD lê dados do disco para a RAM de Maria mas falha



João

Dados no disco

Maria

Introdução

Passo 3 – João altera os dados, o pedido da Maria falha

Passo 4 - SGBD atualiza os dados do João, pedido da Maria falha

Passo 5 - SGBD libera os dados e Maria consegue ler

Introdução

Passo 6 – Maria atualiza os dados SGBD atualiza no disco

Passo 7 – SGBD libera os bloqueios

Banco está atualizado e consistente

Escalonamento

- Escalonamento de transações (conflitos)
  - Duas transações T_i e T_j com duas operações I_i e I_j podem ter suas operações intercaladas:
    - Se I_i e I_j são leituras, a ordem não é importante.
    - Se I_i e I_j são operações sobre itens diferentes, a ordem não é importante.
    - Se I_i=r(Q) e I_j=w(Q) e i<j, T_i não lê o valor escrito por T_j, assim a ordem é importante (conflito RW)
    - Se I_i=w(Q) e I_j=r(Q) e i<j, T_j lê o valor de T_i, assim a ordem é importante (conflito WR)
    - Se I_i=w(Q) e I_j=w(Q), a ordem é importante pois uma transação pode sobrescrever o item da outra (conflito WW)

Escalonamento

- Escalonamentos
  - Intuitivamente, um conflito entre I_i e I_j provoca uma ordem temporal entre elas
  - Duas operações em um escalonamento são consideradas entrando em conflito se:
    - Pertencem a diferentes transações
    - Acessam o mesmo item Q
    - Pelo menos umas das operações é um Write
    - Assim temos os acrônimos:
      - RW (pode causar leitura não repetível)
      - WR (pode causar leitura suja)
      - WW (pode causar atualização perdida)

Escalonamento

- Escalonamentos
  - O escalonador deve cooperar com o recuperador.
  - Categorias
    - Recuperáveis x não recuperáveis
    - Permite aborto em cascata x evitam aborto em cascata
    - Estritos x não estritos

Escalonamento

- Escalonamentos
  - Não recuperáveis
    - Um escalonamento **S** pode ser não recuperável se uma transação T_2 lê um item de outra transação T_1, e T_2 *commita* antes de T_1
      - Uma transação T_2 lê um item gravado por T_1, T_2 confirma e em seguida T_1 aborta.

T1	T2
read(X)	
X = X – 20	
write(X)	
	read(X)
	X = X + 10
	write(X)
	commit
abort	

Escalonamento

- Escalonamentos
  - Recuperáveis
    - Um escalonamento **S** é recuperável se nenhuma transação T_2 em **S** é confirmada até que todas as transações T_1 que gravaram algum item que T_2 lê sejam confirmadas

T1	T2
read(X)	
X = X – 20	
write(X)	
	read(X)
	X = X + 10
	write(X)
commit	
	commit

Escalonamento

- Escalonamentos
  - Recuperáveis
    - Pode gerar aborto em cascata
      - S': $r_1(x);w_1(x);r_2(x);r_1(y);w_2(x);w_1(y);a_1;c_2$
    - Evitar aborto em cascata
      - Uma T em **S** só pode ler dados que tenham sido atualizados por outras transações que já concluíram
      - S': $r_1(x);r_2(y);w_1(x);w_2(y);c_1;r_2(x);w_2(x);c_2$

Escalonamento

- Escalonamentos
  - Estrito
    - Um escalonamento é dito estrito se uma transação T não pode ler nem escrever um item X até que a última transação T' que escreveu X tenha sido validada ou abortada.
      - Todo escalonamento estrito é recuperável e evita aborto em cascata
    - Escalonamento estrito:
      - S': $r_1(X)$; $r_2(Y)$; $w_1(X)$; $w_2(Y)$; c_1; $r_2(X)$;$w_2(X)$; c_2;

Escalonamento

- Escalonamentos
  - Teoria da serializibilidade
    - Garantia de escalonamentos não-seriais válidos
    - Premissa
      - “um escalonamento não-serial de um conjunto de transações deve produzir resultado equivalente a alguma execução serial destas transações”

Escalonamento

- Escalonamentos
  - Teoria da serializibilidade

Serial

T1	T2
read(X)	
X = X – 20	
write(X)	
read(Y)	
Y = Y + 20	
write(Y)	
	read(X)
	X = X + 10
	write(X)

não-serial serializável

T1	T2
read(X)	
X = X – 20	
write(X)	
	read(X)
	X = X + 10
	write(X)
read(Y)	
Y = Y + 20	
write(Y)	

Escalonamento

- Teoria da serializibilidade (equivalência de schedules)
  - Escalonamento serializável equivalente no resultado
    - Visto apenas para conhecimento
  - Escalonamento serializável equivalente em conflito
    - A ordem de duas operações em conflito quaisquer for a mesma nos dois escalonamentos
    - Transformando um schedule serial **S** em um não serial **S'** com **S'** respeitando as operações em conflitos, **S'** é dito serializável equivalente em conflito

Escalonamento

- Teoria da serializibilidade (equivalência de schedules)
  - Escalonamento serializável equivalente em conflito

S

T1	T2
read(X)	
X = X – 20	
write(X)	
read(Y)	
Y = Y + 20	
write(Y)	
	read(X)
	X = X + 10
	write(X)

S'

T1	T2
read(X)	
X = X – 20	
write(X)	
	read(X)
	X = X + 10
	write(X)
read(Y)	
Y = Y + 20	
write(Y)	

Escalonamento

- Equivalência em conflito
  - Construção do grafo de precedência

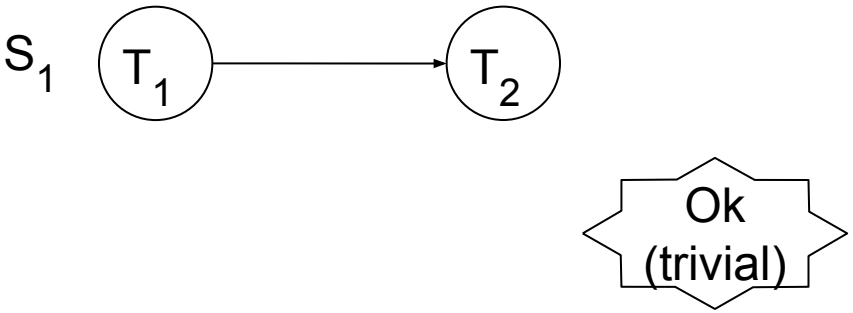
1. Para cada transação T_i em S crie um nó rotulado T_i no grafo G
2. Para cada operação $r_j(X)$ de T_j após $w_i(x)$ de T_i, crie um aresta $T_i \rightarrow T_j$
3. Para cada operação $w_j(X)$ de T_j após $r_i(x)$ de T_i, crie uma aresta $T_i \rightarrow T_j$
4. Para cada operação $w_j(X)$ de T_j após $w_i(x)$ de T_i, crie uma aresta $T_i \rightarrow T_j$

  - Se **G** não tiver ciclos, **S** é serializável equivalente em conflito

Escalonamento

- Equivalência em conflito

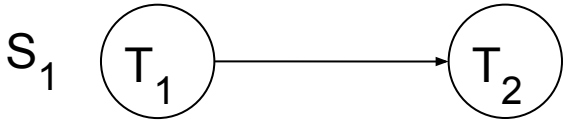
S_1: $r_1(X);w_1(X-20);r_1(Y);w_1(Y-20);r_2(X);w_2(X+10)$
S_2: $r_2(X);w_2(X+10);r_1(X);w_1(X-20);r_1(Y);w_1(Y-20)$
S_3: $r_1(X);r_2(X);w_1(X-20);r_1(Y);w_2(X+10);w_1(Y-20)$
S_4: $r_1(X);w_1(X-20);r_2(X);w_2(X+10);r_1(Y);w_1(Y-20)$

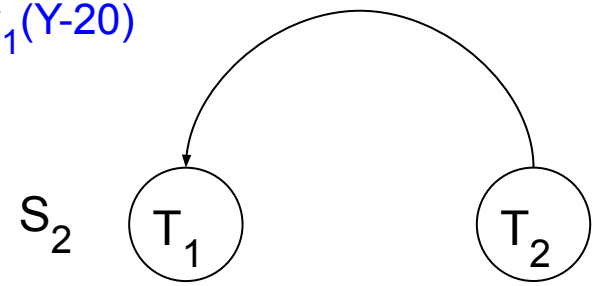


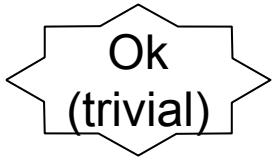
Escalonamento

- Equivalência em conflito

S_1: $r_1(X);w_1(X-20);r_1(Y);w_1(Y-20);r_2(X);w_2(X+10)$
S_2: $r_2(X);w_2(X+10);r_1(X);w_1(X-20);r_1(Y);w_1(Y-20)$
S_3: $r_1(X);r_2(X);w_1(X-20);r_1(Y);w_2(X+10);w_1(Y-20)$
S_4: $r_1(X);w_1(X-20);r_2(X);w_2(X+10);r_1(Y);w_1(Y-20)$



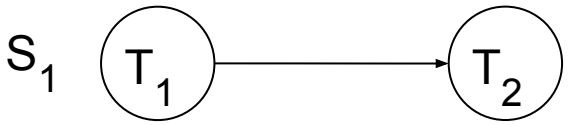


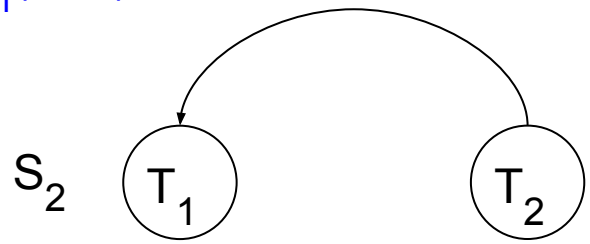


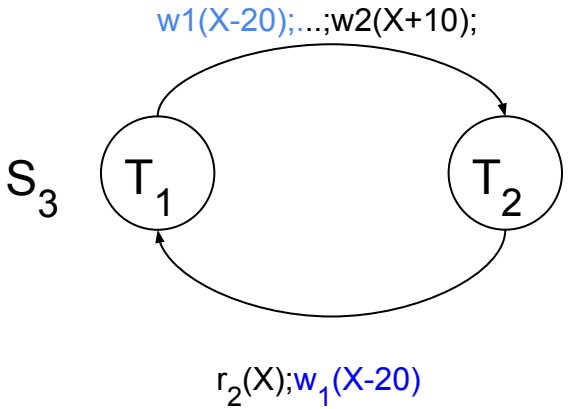
Escalonamento

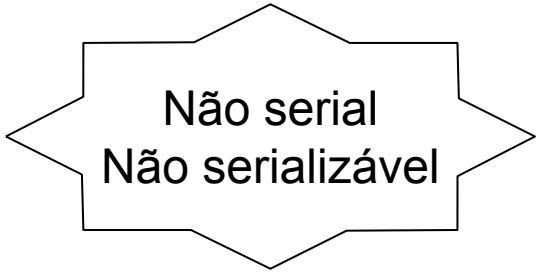
- Equivalência em conflito

S_1: $r_1(X);w_1(X-20);r_1(Y);w_1(Y-20);r_2(X);w_2(X+10)$
S_2: $r_2(X);w_2(X+10);r_1(X);w_1(X-20);r_1(Y);w_1(Y-20)$
S_3: $r_1(X);r_2(X);w_1(X-20);r_1(Y);w_2(X+10);w_1(Y-20)$
S_4: $r_1(X);w_1(X-20);r_2(X);w_2(X+10);r_1(Y);w_1(Y-20)$





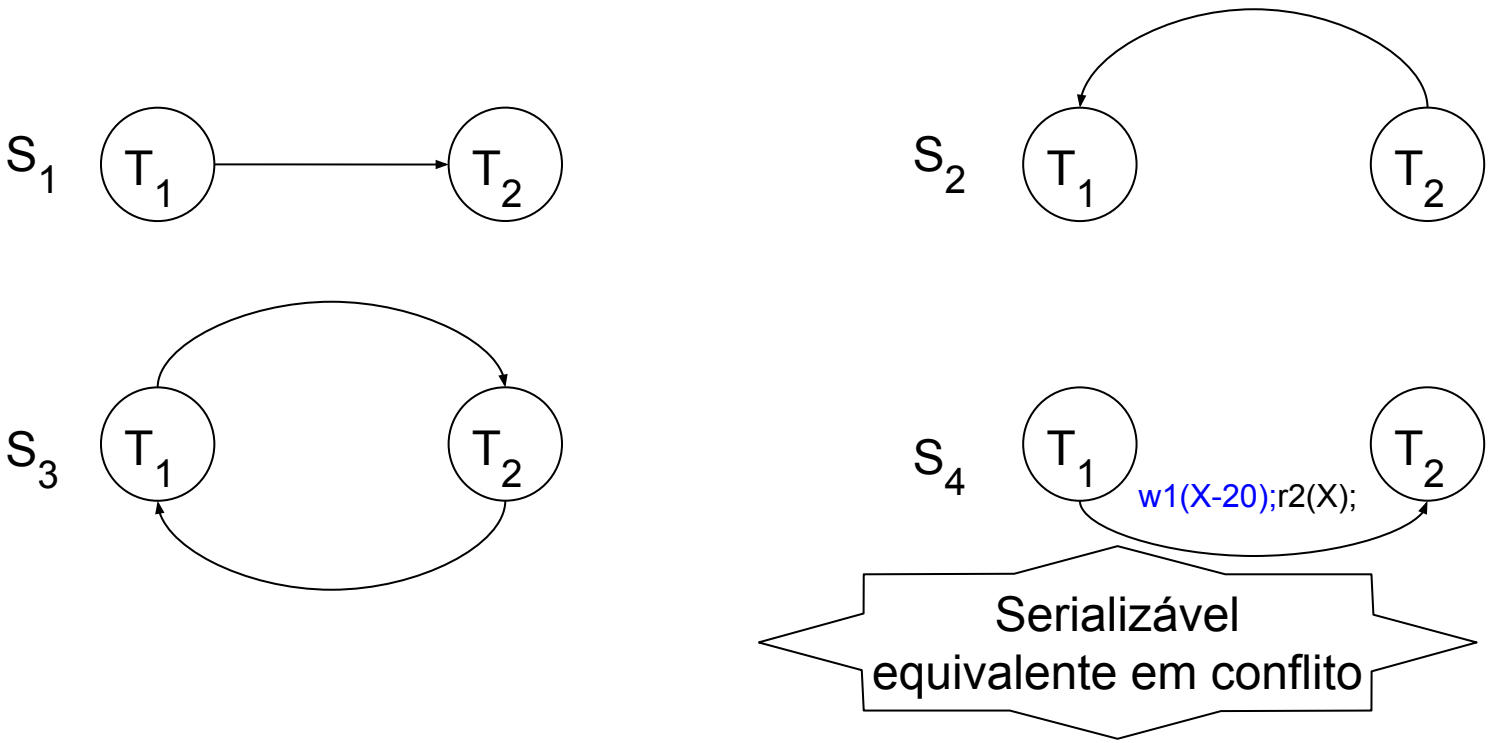




Escalonamento

- Equivalência em conflito

S_1: $r_1(X);w_1(X-20);r_1(Y);w_1(Y-20);r_2(X);w_2(X+10)$
S_2: $r_2(X);w_2(X+10);r_1(X);w_1(X-20);r_1(Y);w_1(Y-20)$
S_3: $r_1(X);r_2(X);w_1(X-20);r_1(Y);w_2(X+10);w_1(Y-20)$
S_4: $r_1(X);w_1(X-20);r_2(X);w_2(X+10);r_1(Y);w_1(Y-20)$



Escalonamento

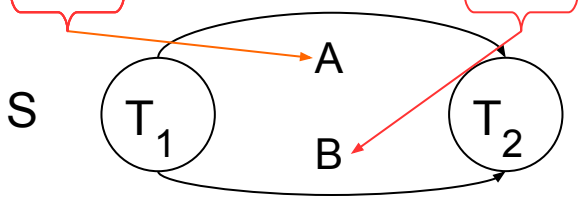
- Equivalência em conflito

  S: $r_1(A);w_1(A);r_2(A);w_2(A);r_1(B);w_1(B);r_2(B);w_2(B)$

Escalonamento

- Equivalência em conflito

S: $r_1(A);w_1(A);r_2(A);w_2(A);r_1(B);w_1(B);r_2(B);w_2(B)$



Outra forma de encontrar a equivalência é trocar as operações não conflitantes

$r_1(A);w_1(A);r_2(A);w_2(A);r_1(B);w_1(B);r_2(B);w_2(B)$

$r_1(A);w_1(A);r_2(A);r_1(B);w_2(A);w_1(B);r_2(B);w_2(B)$ (1)

$r_1(A);w_1(A);r_1(B);r_2(A);w_2(A);w_1(B);r_2(B);w_2(B)$ (2)

$r_1(A);w_1(A);r_1(B);r_2(A);w_2(A);w_1(B);r_2(B);w_2(B)$ (3)

$r_1(A);w_1(A);r_1(B);r_2(A);w_1(B);w_2(A);r_2(B);w_2(B)$ (4)

$r_1(A);w_1(A);r_1(B);w_1(B);r_2(A);w_2(A);r_2(B);w_2(B)$

Escalonamento

- Equivalência em conflito

  S: $r_2(A);r_1(B);w_2(A);r_2(B);r_3(A);w_1(B);w_3(A);w_2(B)$

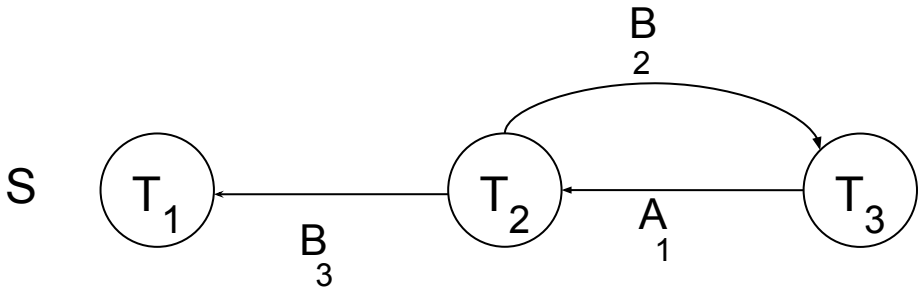
  Grafo de precedência:

Escalonamento

- Equivalência em conflito

S: $r_2(A);r_1(B);w_2(A);r_2(B);r_3(A);w_1(B);w_3(A);w_2(B)$

Grafo de precedência:



Escalonamento

- Equivalência em conflito

  S: $r_2(A);r_1(B);w_2(A);r_3(A);w_1(B);w_3(A);r_2(B);w_2(B)$

  Grafo de precedência:

Escalonamento

- Teoria da serializibilidade (equivalência de schedules)
  - Escalonamento serializável equivalente em visão
    - Equivalência menos restritiva que a em conflito
    - Considere dois escalonamentos **S** e **S'** com o mesmo conjunto de transações. **S** e **S'** são ditos equivalentes em visão se as seguintes condições forem satisfeitas:
      - Para cada item Q, se T_i fizer a leitura inicial de Q em **S**, então T_i deve ler o valor inicial de Q em **S'**
      - Para cada item Q, se T_i fizer $r_i(Q)$ em **S**, o Q foi produzido por T_j (se houver), então T_i em S' deve ler o valor produzido por T_j
      - Para cada item Q, a transação que executa w(Q) final em **S** deve executar o w(Q) final em **S'**

Escalonamento

- Equivalência em visão

S_1: $r_1(A);w_1(A-50);r_1(B);w_1(B+50);r_2(A);w_2(A*0.9);r_2(B);w_2(B*1.1);$
S_2: $r_2(A);w_2(A*0.9);r_2(B);w_2(B*1.1);r_1(A);w_1(A-50);r_1(B);w_1(B+50);$
S_3: $r_1(A);w_1(A-50);r_2(A);w_2(A*0.9);r_1(B);w_1(B+50);r_2(B);w_2(B*1.1);$
S_4: $r_2(A);w_2(A*0.9);r_1(A);w_1(A-50);r_2(B);w_2(B*1.1);r_1(B);w_1(B+50);$

Escalonamento

- Equivalência em visão

S_1: $r_1(A);w_1(A-50);r_1(B);w_1(B+50);r_2(A);w_2(A*0.9);r_2(B);w_2(B*1.1);$
S_2: $r_2(A);w_2(A*0.9);r_2(B);w_2(B*1.1);r_1(A);w_1(A-50);r_1(B);w_1(B+50);$
S_3: $r_1(A);w_1(A-50);r_2(A);w_2(A*0.9);r_1(B);w_1(B+50);r_2(B);w_2(B*1.1);$
S_4: $r_2(A);w_2(A*0.9);r_1(A);w_1(A-50);r_2(B);w_2(B*1.1);r_1(B);w_1(B+50);$

  - S_1 e S_2 não são equivalentes em visão (trivial)
  - S_1 e S_3 são
  - S_2 e S_4 são

Escalonamento

- Equivalente em visão
  - O conceito equivalente em visão leva ao conceito de serializável equivalente em visão
    - Um escalonamento **S'** é serializável equivalente em visão se **S'** for equivalente em visão em relação a algum escalonamento serial **S**
    - A ideia por trás desta abordagem:
      - As operações de leitura das transações têm a mesma visão do dado em ambos os escalonamentos **S** e **S'**
      - A gravação final de um item seja a mesma nos dois escalonamentos

Escalonamento

- O escalonador deve:
  - Propor escalonamentos não seriais serializáveis
  - Abordagens:
    - Escalonamento serializável equivalente em conflito
      - Propor um escalonamento não serial que seja equivalente em conflito a um escalonamento serial
    - Escalonamento serializável equivalente em visão
      - Propor em escalonamento não serial que seja equivalente em visão a um escalonamento serial

Escalonamento

- Escalonamentos
  - Os problemas de escalonamento são solucionados através de bloqueios
    - Binários
    - Compartilhados
    - Duas fases (2PL)

Bloqueios

- Bloqueio (lock) é uma das principais técnicas para controle de concorrência
  - Variável associado a um item de dados que indica quais as operações possíveis sobre tal dado
- Bloqueios binários
  - Possui dois estados: 0 (livre) e 1 (bloqueado)
  - Simples, mas muito restritivo
  - Operações
    - lock_item(X), se Lock(X)=1 transação espera até Lock(X)=0, bloqueando o X.
    - unlock_item(X), faz com que Lock(X) seja 0.

Bloqueio Binário

- Algoritmos

lock_item(X)

```
B: if Lock(X)=0
   Then Lock(X) ← 1
   Else
     Wait while Lock(X)==1
     Go to B
```

unlock_item(X)

```
Lock(X)<-0
 If Transaction Waiting
Then Restart Transaction
```

Necessário apenas um tupla para cada bloqueio `<item,LOCK, transação>` mais uma fila com as transações que esperam

Bloqueio Binário

- Neste esquema cada transação T deve obedecer
  - T deve executar um lock_item(X) antes de qualquer read(X) ou write(X)
  - T deve executar um unlock_item(X) depois de todos read(X) e/ou write(X)
  - T não executará um lock_item(X) se já estiver com o bloqueio de X
  - T não executará um unlock_item(X) caso não tenha feito um lock_item(X)

Bloqueio Compartilhado

- Bloqueios compartilhados/exclusivos
  - Parte do princípio que para uma leitura o bloqueio não precisa ser exclusivo
  - Três modos: read_lock(X), write_lock(X) e unlock(X)
  - A variável LOCK tem três estados: read_locked, write_locked e unlock.
  - Se X está read_locked está com bloqueio compartilhado (share-locked)
  - Tupla que controla `<item,LOCK,qtleituras, transações bloqueio>`

Bloqueio Compartilhado

- Nesta abordagem uma transação T deve:
  - Garantir que read_lock(X) ou write_lock(X) antes de qualquer read(X)
  - Garantir um write_lock(X) antes de qualquer write(X)
  - Garantir unlock(X) depois de todas as operações write(X) ou read(X) pode ser relaxada
  - Não executar um read/write_lock(X) se ela já possuir um bloqueio de leitura/escrita pode ser relaxada
  - Executar um unlock(X) apenas se X tiver sido bloqueado

Bloqueio Compartilhado

- Conversão de bloqueio
  - Uma transação pode bloquear um item X para leitura e depois *promover* o bloqueio para escrita
  - Uma transação pode bloquear um item X para escrita e depois *rebaixar* o bloqueio para leitura
- A tabela de bloqueio deve possuir identificadores para todas as situações implementadas
- Bloqueios não garantem por si só a serialidade das concorrências

Bloqueios

T_1	T_2
read_lock(Y)	
read(Y)	
unlock(Y)	
	read_lock(X)
	read(X)
	unlock(X)
	write_lock(Y)
	read(Y)
	Y=X+Y
	write(Y)
	unlock(Y)
write_lock(X)	
read(X)	
X=X+Y	
write(X)	
unlock(X)	

Dados X=20 e Y=30, a execução serial T_1 -> T_2 resulta X=50 e Y=80
T_2->T_1 resulta X=70 e Y=50

Bloqueios

T_1	T_2
read_lock(Y)	
read(Y)	
unlock(Y)	
	read_lock(X)
	read(X)
	unlock(X)
	write_lock(Y)
	read(Y)
	Y=X+Y
	write(Y)
	unlock(Y)
write_lock(X)	
read(X)	
X=X+Y	
write(X)	
unlock(X)	

Dados X=20 e Y=30, a execução serial T_1 -> T_2 resulta X=50 e Y=80
T_2->T_1 resulta X=70 e Y=50

O plano (escalonamento) ao lado gera X=50 e Y=50 (não serializável, mesmo com os protocolos de bloqueio)

Solução?

Bloqueio de 2 Fases (2PL)

- Garante a serialização dos escalonamentos das transações (2 Phase Lock)
- Funcionamento
  - Todos os read_lock(X) e write_lock(X) precedem o primeiro unlock(X), ou seja, após a primeira liberação não existem mais bloqueios
    - Fase de expansão
  - Após o primeiro unlock(X) não podem mais existir bloqueios
    - Fase de encolhimento
  - Promoção de bloqueios é na fase de expansão
  - Rebaixamento é feito na fase de encolhimento

2PL

- Quais transações obedecem 2PL?

T_1	T_2	T_3	T_4
Read_lock(Y) Read(Y) Unlock(Y) Write_lock(X) Read(X) X:=X+Y Write(X) Unlock(X)	Read_lock(X) Read(X) Unlock(X) Write_lock(Y) Read(Y) Y:=X + Y Write(Y) Unlock(Y)	Read_lock(Y) Read(Y) Write_lock(X) Unlock(Y) Read(X) X:=X+Y Write(X) Unlock(X)	Read_lock(X); Write_lock(Y); Read(X); Read(Y); Y:=X + Y; Write(Y); Unlock(X); Unlock(Y);

Bloqueio de 2 Fases (2PL)

T_1	T_2
read_lock(Y)	**read_lock**(X)
read(Y)	read(X)
write_lock(X)	**write_lock**(Y)
unlock(Y)	**unlock**(X)
read(X)	read(Y)
X=X+Y	Y=X+Y
write(X)	write(Y)
unlock(X)	**unlock**(Y)

As duas transações obedecem o protocolo 2PL

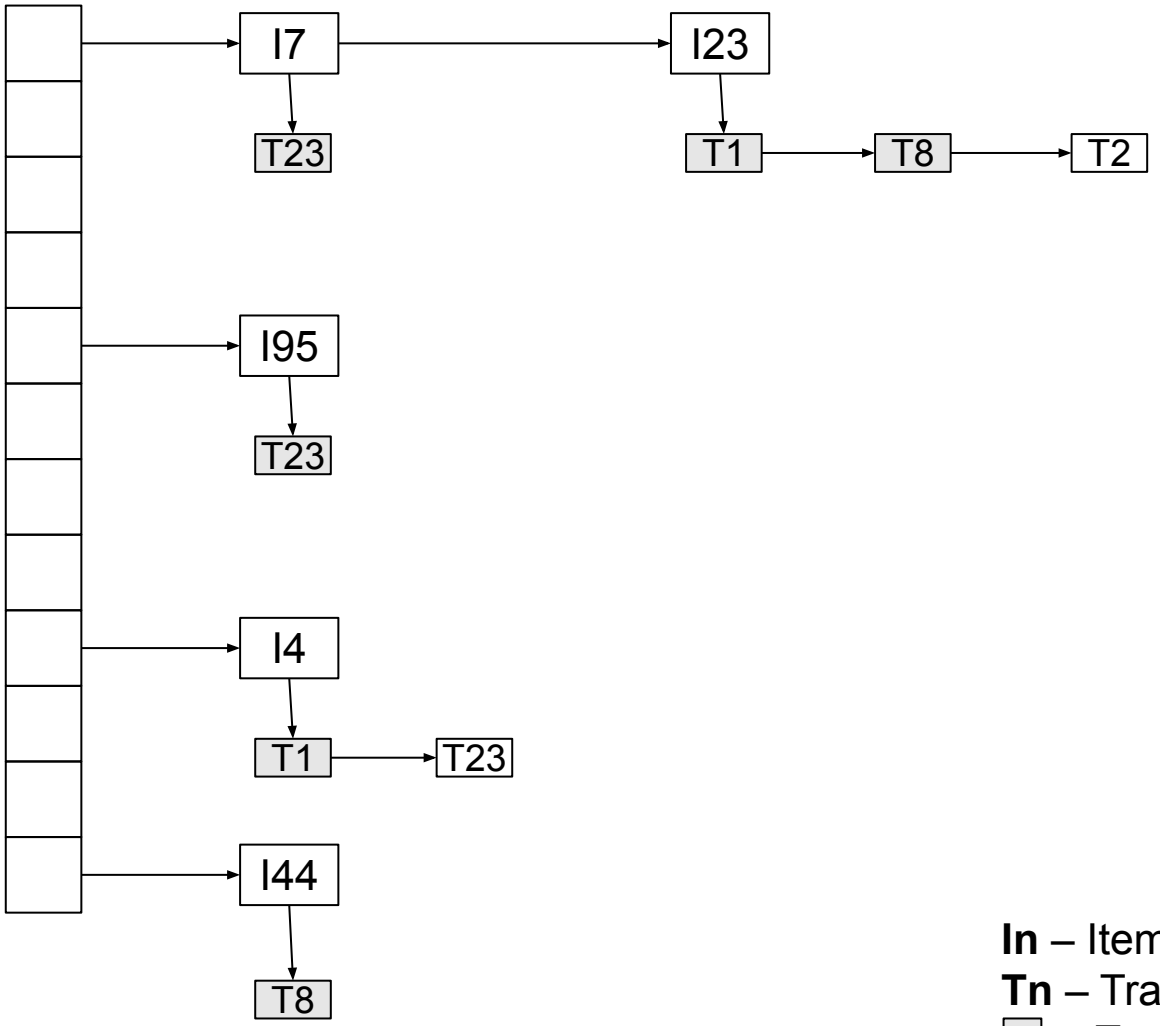
Um plano de execução com transações que obedecem o protocolo 2PL sempre será serializável.

Variações 2PL

- Existem várias implementações do 2PL
  - Básico: visto anteriormente
  - Conservador (ou estático): todos os itens devem ser bloqueados antes da transação iniciar
  - Estrito: a transação só libera os write_lock(X) após ser efetivada (commit) ou abortada (rollback)
  - Rigoroso: similar ao estrito mas libera todos os bloqueios apenas após a transação ser efetivada ou abortada

Discussões

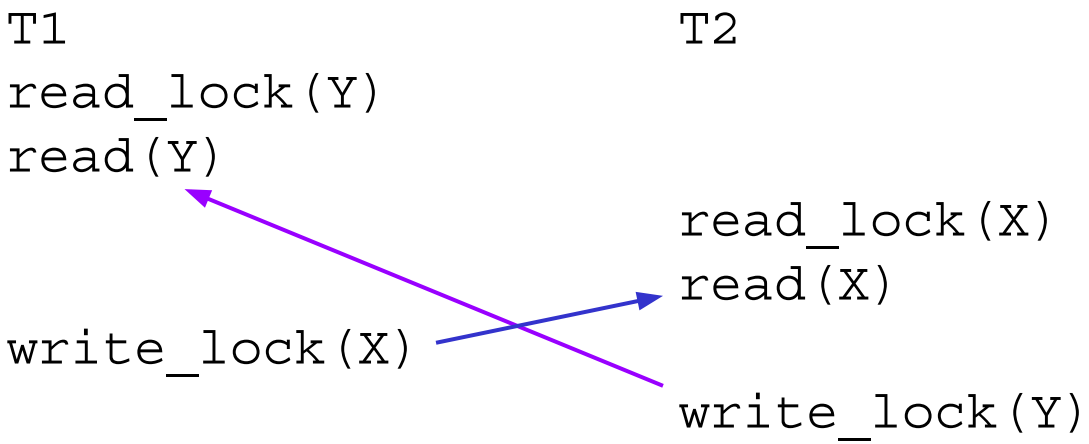
Tabela Bloqueios



In – Item n
Tn – Transação n
Transação conseguiu lock
Transação espera lock

Impasse (Deadlock)

- Impasse (deadlock) ocorre quando transações bloqueiam um item **X** e esperam por um item **Y** bloqueado por outros transações que estão esperando por **X**



Impasse (Deadlock)

- Protocolos de prevenção
  - 2PL conservador é deadlock free.
    - Limita a concorrência
  - Ordenado: tentar por uma ordenação em todos os itens e os locks só podem ocorrer segundo esta ordem
    - Também limita a concorrência
    - Exige que o programador/sistema conheça a ordem
  - Timestamp (TS) (marca de tempo)
    - Identificador único assinalado a cada transação
    - Ordenados segundo a ordem em que as transações começaram
    - Principal vantagem: não usa bloqueios logo *deadlock* é impossível

Impasse (Deadlock)

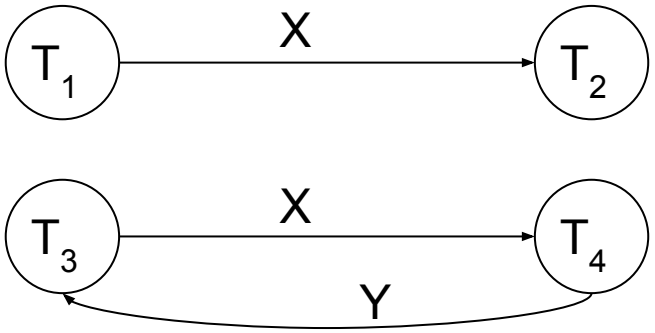
- Dois esquemas para evitar deadlock
  - Dadas duas transações T' e T'' e T'' tem o item X que T' quer acessar
  - Esperar-morrer (wait-die): transações mais antigas esperam, as mais novas são abortadas e voltam com o mesmo TS
    - If TS(T') < TS(T'') então T' espera senão Abort(T')
  - Ferir-esperar (wound-wait): transações mais novas esperam pelas antigas e as mais antigas abortam as mais novas (voltam com o mesmo TS)
    - If TS(T') < TS(T'') então Abort(T'') senão T' espera
  - Podem abortar transações sem necessidade mas evita starvation

Impasse (Deadlock)

- Técnicas sem rótulo de tempo
  - Sem Espera (no waiting - NW): se uma transação T for incapaz de conseguir um bloqueio, T aborta
    - Reiniciada após um certo tempo
    - Podem abortar sem necessidade
  - Espera Cuidadosa (cautious waiting - CW): tenta reduzir o número de abortos/reinícios
    - Se T deseja **I** e **I** está bloqueado por T' então se T' não está em fila de espera, T espera, senão aborta T e reinicia T.

Impasse (Deadlock)

- Técnicas sem rótulo de tempo
  - Detecção de deadlock
    - Um grafo de espera é construído, se duas transações têm arcos uma para outra, se existir um ciclo, existe deadlock



T_1 espera por **X**
bloqueado por T_2
T_3 espera por **X**
bloqueado por T_4
que espera **Y** de T_3

  - Timeouts: se uma transação T esperar por um item por um determinado tempo, T é abortada.

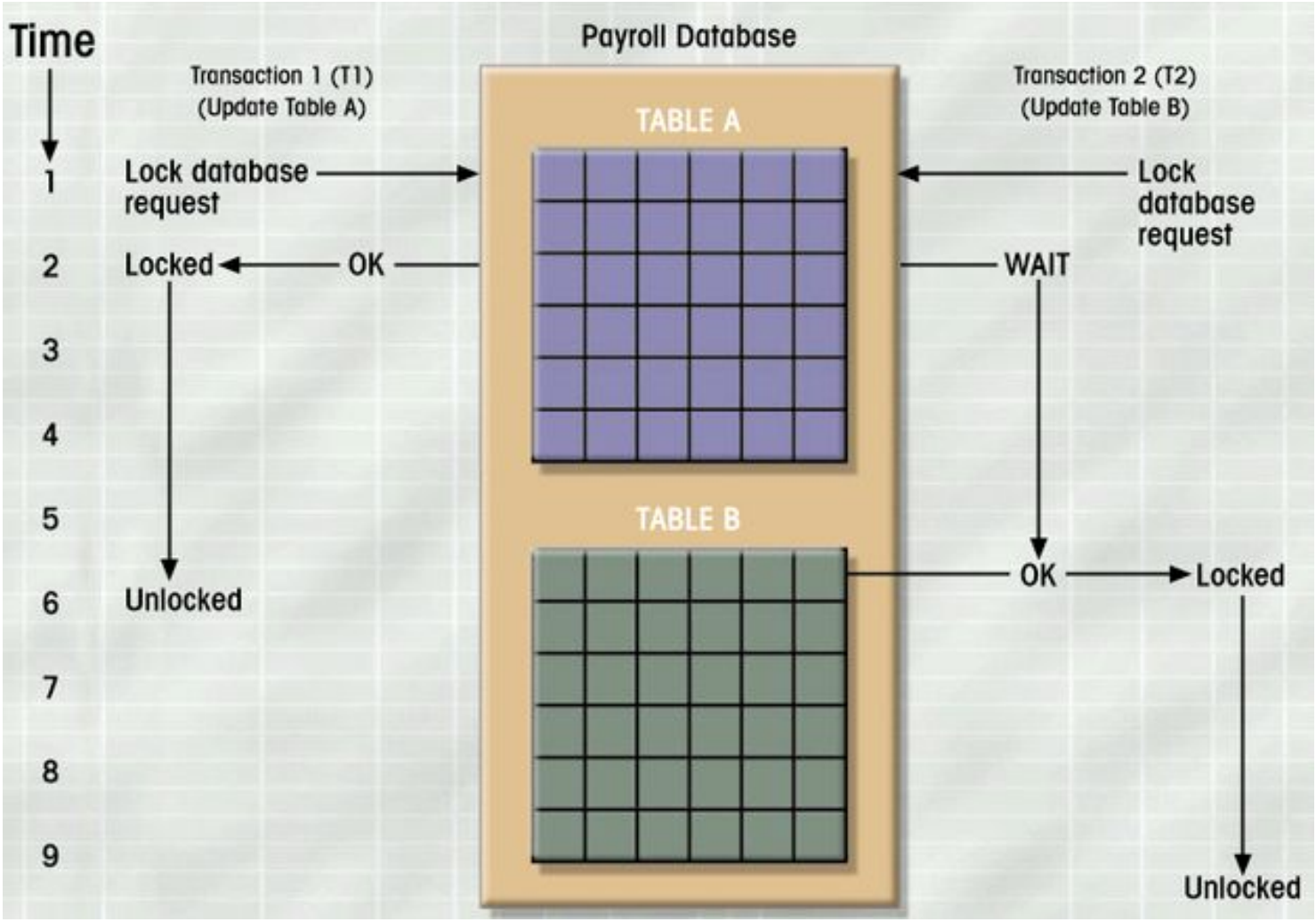
Inanição (starvation)

- Uma transação fica esperando por um período indefinido devido às políticas de espera por itens bloqueados for injusto
  - Uma transação com maior prioridade toma a vez de uma transação que espera
  - Pode ser resolvido com uma fila simples (FIFO – primeiro a chegar, primeiro a ser atendido)

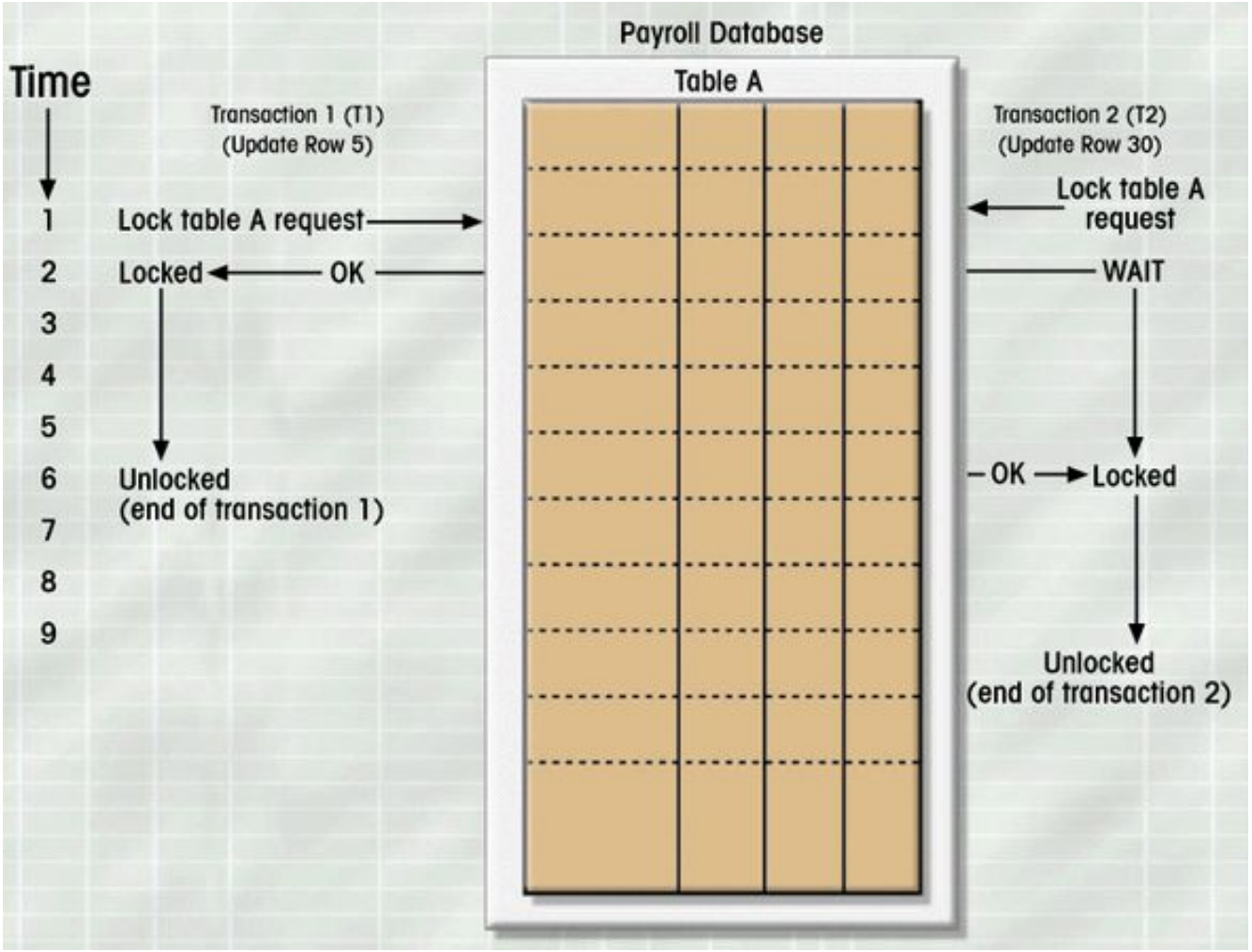
Granularidade de Bloqueio

- O que pode ser um item X?
  - Todo o banco
  - Uma tabela
  - Uma página
  - Um conjunto de tuplas
  - Um conjunto de colunas
  - Uma tupla
  - Uma coluna
- Quanto menor a granularidade, a concorrência é maior porém haverá mais overhead para controle
- Quanto maior a granulidade, menos concorrência e menos overhead de controle

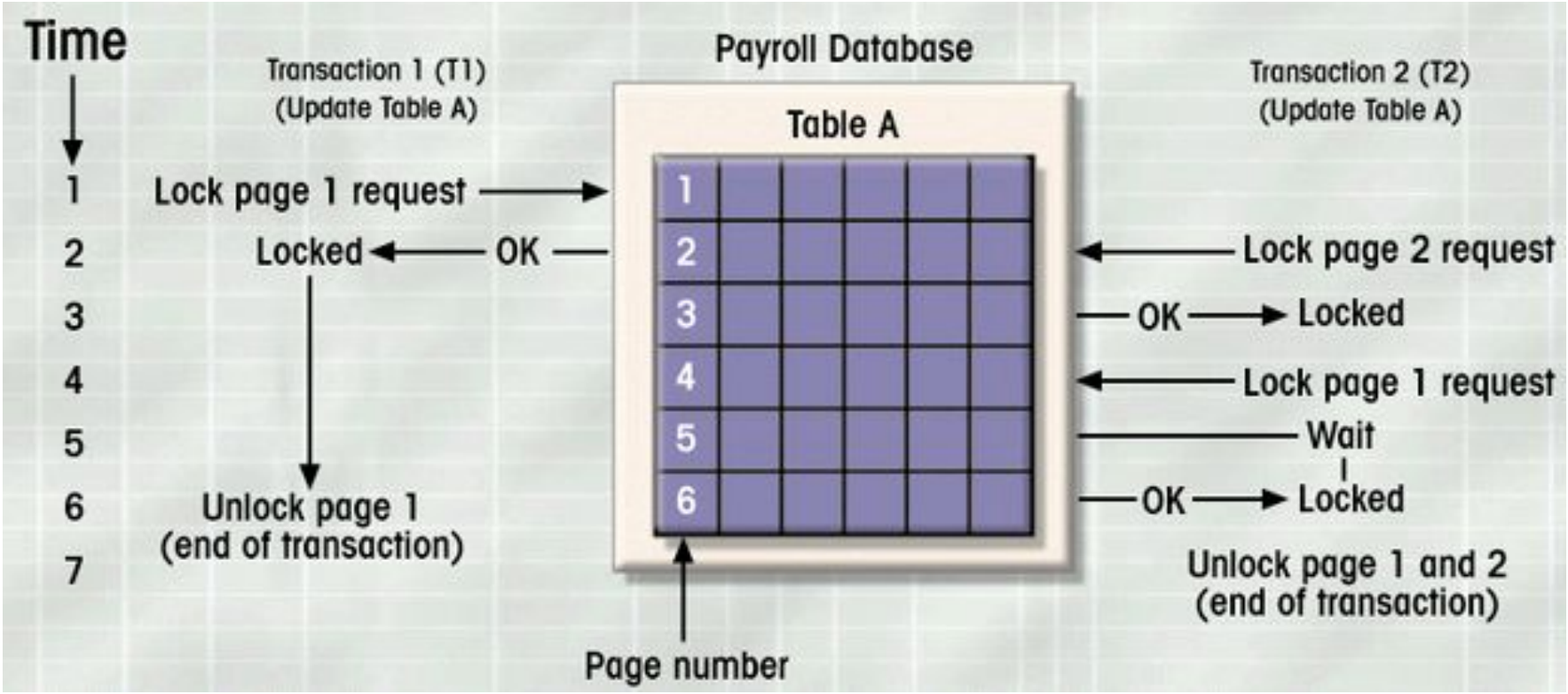
Granularidade BD



Granularidade Tabela



Granularidade Página



Granularidade Linha (Tupla)