



UNIVERSIDADE FEDERAL DE ITAJUBÁ

Banco de Dados II

COM 231

Controle de Concorrência

Vanessa Cristina Oliveira de Souza



Classificação dos SGBD's quanto ao número de usuários suportados

■ Mono-usuários

- ☐ no máximo um usuário pode usar o mesmo sistema por vez

■ Multi-usuários

- ☐ muitos usuários podem usar o mesmo sistema coincidentemente
- ☐ Para garantir a integridade do BD é necessário usarmos o conceito de Transações **'serializáveis'**



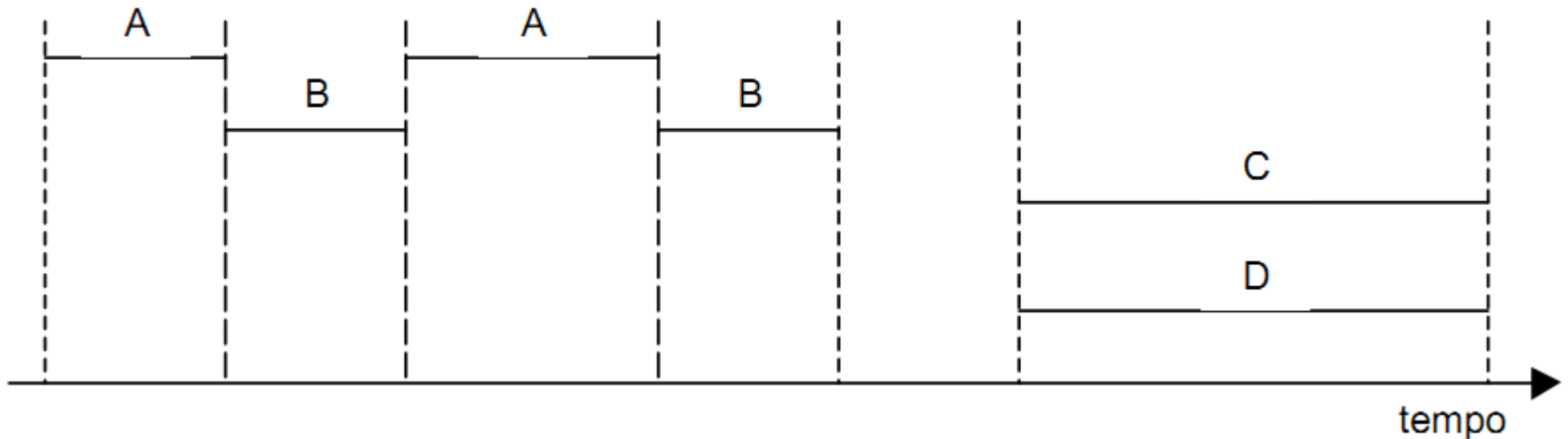
Concorrência

- Transações podem ser executadas:
 - Sequencialmente
 - uma transação T termina antes do início ou após o final de uma transação T_j
 - Concorrentemente
 - partes das transações T e T_j podem ser processadas em paralelo



Transações Concorrentes

- Geralmente, a execução concorrente de transações é realizada de forma concorrente e não sequencial.





Transações Concorrentes

- Concorrência entre transações é necessária para melhorar o desempenho;
- Aproveita-se o tempo que uma transação permanece bloqueada para leitura e escrita no disco para processar outras transações;
- Evita que transações muito longas retardem a execução de transações mais curtas;



Transações Concorrentes

- Quando diversas transações são executadas de forma concorrente em um banco de dados, a propriedade do **isolamento** pode não ser preservada.
- É necessário que o sistema controle a interação entre transações concorrentes.
- Esse controle é alcançado por mecanismos chamados de esquemas de **controle de concorrência**.



Controle de Concorrência

- Subsistema que garante a seguinte propriedade das transações:

- ☐ **Isolamento:**

- a execução da transação não deve ser afetada pela execução concorrente de outras transações



Por quê fazer controle de concorrência?

- Transações submetidas por vários usuários podem ser executadas concorrentemente e podem acessar os mesmos itens do banco de dados.
- Se esta execução for descontrolada, pode ocorrer problemas.



Controle de Concorrência

- O subsistema de controle de concorrência precisa gerenciar o **escalonamento** das transações concorrentes.



Escalonamentos

- Um escalonamento (***schedule***) S de n transações T_1, T_2, \dots, T_n é uma ordenação das operações dessas transações, tal que, para cada transação T_i , as operações de T_i em S devem aparecer na mesma ordem em que elas aparecem em T_i .
- Ou, escalonamento é quando se pega um conjunto de transações e define a ordem em que elas vão ser executadas, para prover concorrência no BD.



Escalonamentos

- No escalonamento somente as operações de `read_item(X, xi)` e `write_item(X,xi)` são importantes.
- Então um escalonamento S , para $T1$ e $T2$ pode ser representado na forma:
 - $S_a: r1(X); r2(X); w1(X); r1(Y); w2(X); c2; w1(Y);$
 - ou
 - $S_b: r1(X); w1(X); r2(X); w2(X); c2; r1(Y);$



Exemplo Escalonamento

T_1
read_item(A)
read_item(D)
write_item(D)

T_2
read_item(B)
write_item(B)
read_item(D)
write_item(D)

T_3
read_item(A)
write_item(A)
read_item(C)
write_item(C)

T_4
read_item(B)
write_item(B)
read_item(A)
write_item(A)

<T₁>

read_item(A)
read_item(D)
write_item(A)

<T₄>

read_item(B)
write_item(B)
read_item(A)
write_item(A)

<T₂>

read_item(B)
write_item(B)

<T₃>

read_item(A)
write_item(A)

<T₂>

read_item(D)

<T₃>

read_item(C)
write_item(C)

<T₂>

write_item(D)



Escalonamento Serializável

Garante que qualquer escalonamento produzido ao se processar um conjunto de transações concorrentemente, seja computacionalmente equivalente a um escalonamento executando essas transações serialmente em alguma ordem.



Escalonamento

T1

read_item(X,a)

a:=a-N

write_item(X,a)

read_item(Y,b)

b := b + N

write_item(Y,b)

T2

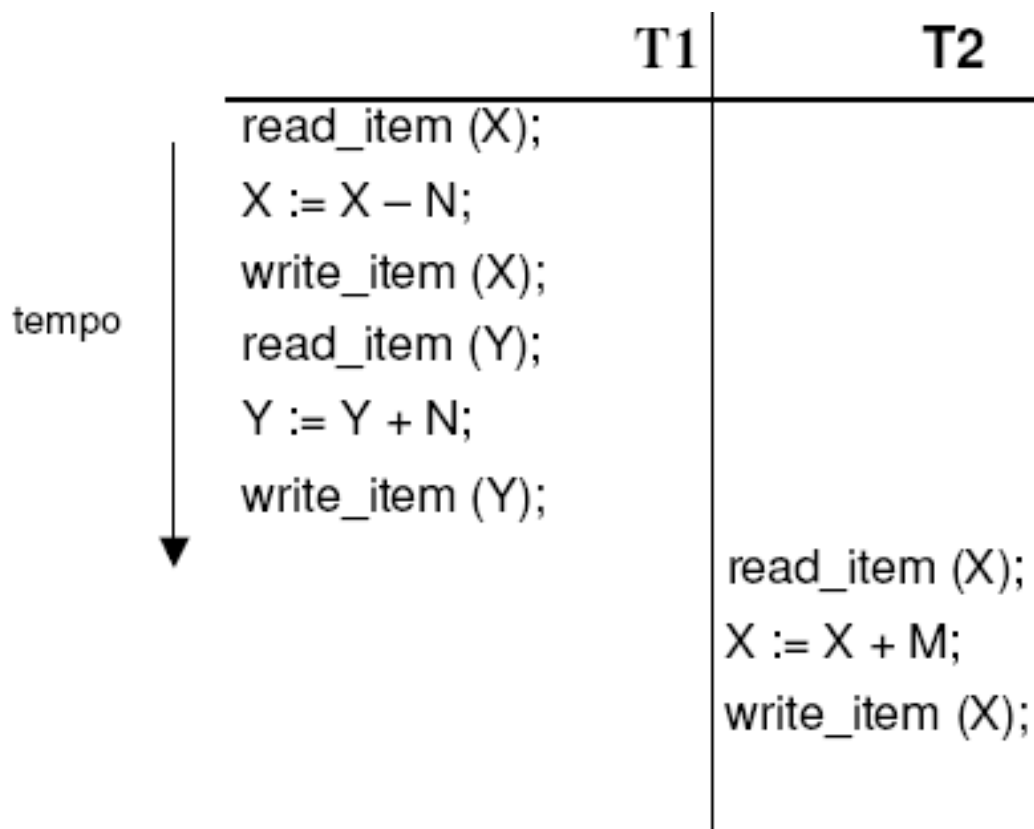
read_item(X,c)

c:=c+M

write_item(X,c)



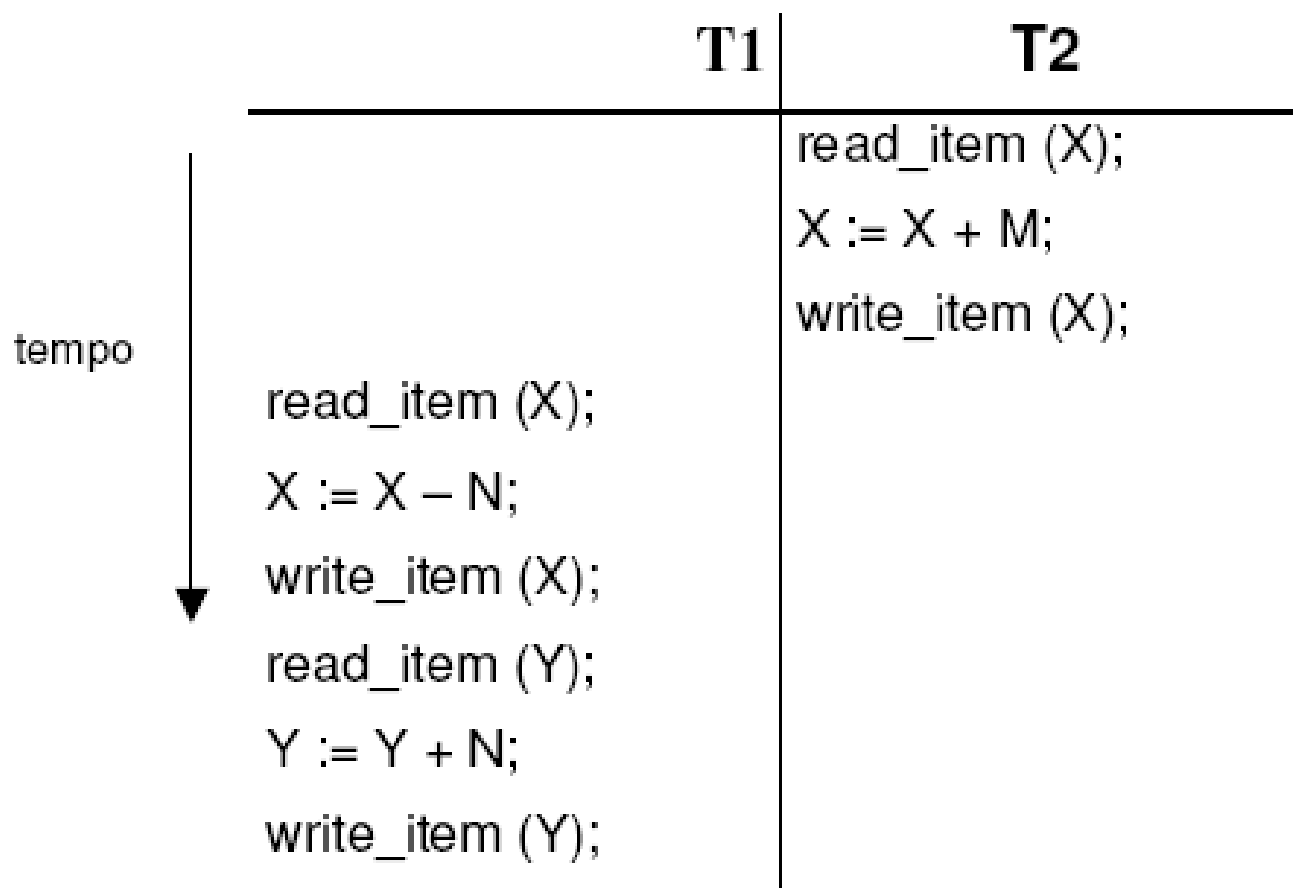
Escalonamento Serializável



- escalonamento da transação T1 seguida pela transação T2 – **notação reduzida**



Escalonamento Serializável



- escalonamento da transação T2 seguida pela transação T1



Escalonamento com entrelaçamento de operações

tempo
↓

T1

read_item(X,a)

a:=a-N

write_item(X,a)

read_item(Y,b)

b := b + N

write_item(Y,b)

T2

read_item(X,c)

c:=c+M

write_item(X,c)



Exercício

- Dada as transações T1 e T2 abaixo. Para valores iniciais de $X=20$ e $Y=30$, quais os valores finais dessas variáveis se:
 - ☐ T1 for executada antes de T2
 - ☐ T2 for executada antes de T1

T1	T2
read_item(Y,a)	read_item(X,c)
read_item(X,b)	read_item(Y,d)
b:=b+a	d:=d+c
write_item(X,b)	write_item(Y,d)



Exercício

- Dada as transações T1 e T2 abaixo. Para valores iniciais de $X=20$ e $Y=30$, quais os valores finais dessas variáveis no final da execução desse escalonamento?
 - O escalonamento é serializável?

T1	T2
read_item(Y,a)	
	read_item(X,c)
	read_item(Y,d)
	d:=d+c
	write_item(Y,d)
read_item(X,b)	
b:=b+a	
write_item(X,b)	



UNIVERSIDADE FEDERAL DE ITAJUBÁ

PROBLEMAS CLÁSSICOS DE CONCORRÊNCIA



Problemas Clássicos de Concorrência

- Na prática, são três os problemas de concorrência avaliados:
 - Atualização Perdida (*Lost Update*)
 - Leitura Suja (*Dirty Read*)
 - Leitura Fantasma (*Phantom Read*)
 - Leitura não-repetitiva (*Nonrepeatable read*)



Escalonamento com entrelaçamento de operações



tempo
↓

T1

read_item(X,a)

a:=a-N

write_item(X,a)

read_item(Y,b)

b := b + N

write_item(Y,b)

T2

read_item(X,c)

c:=c+M

Problema de atualização perdida

write_item(X,c)

O item x tem o valor incorreto porque sua alteração em T1 foi perdida (escrita por cima)



Escalonamento com entrelaçamento de operações

tempo
↓

T1

read_item(X,a)

a:=a-N

write_item(X,a)

read_item(Y,b)

b := b + N

write_item(Y,b)

T2

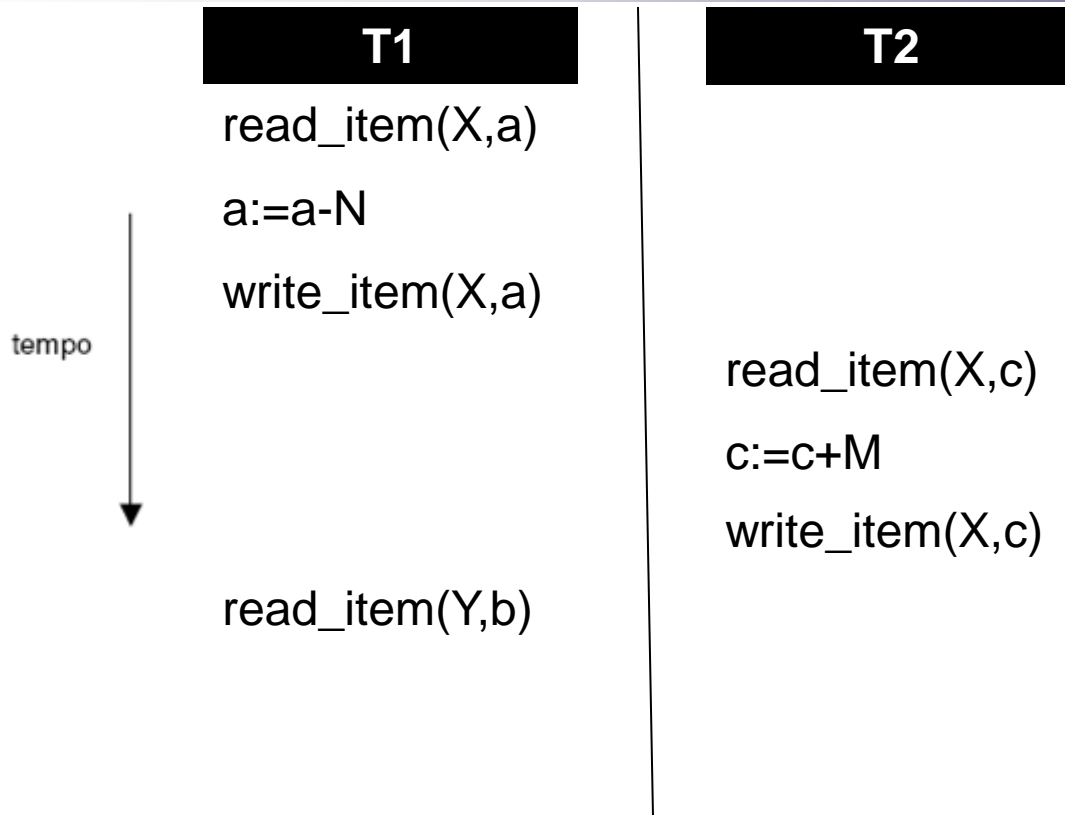
read_item(X,c)

c:=c+M

write_item(X,c)

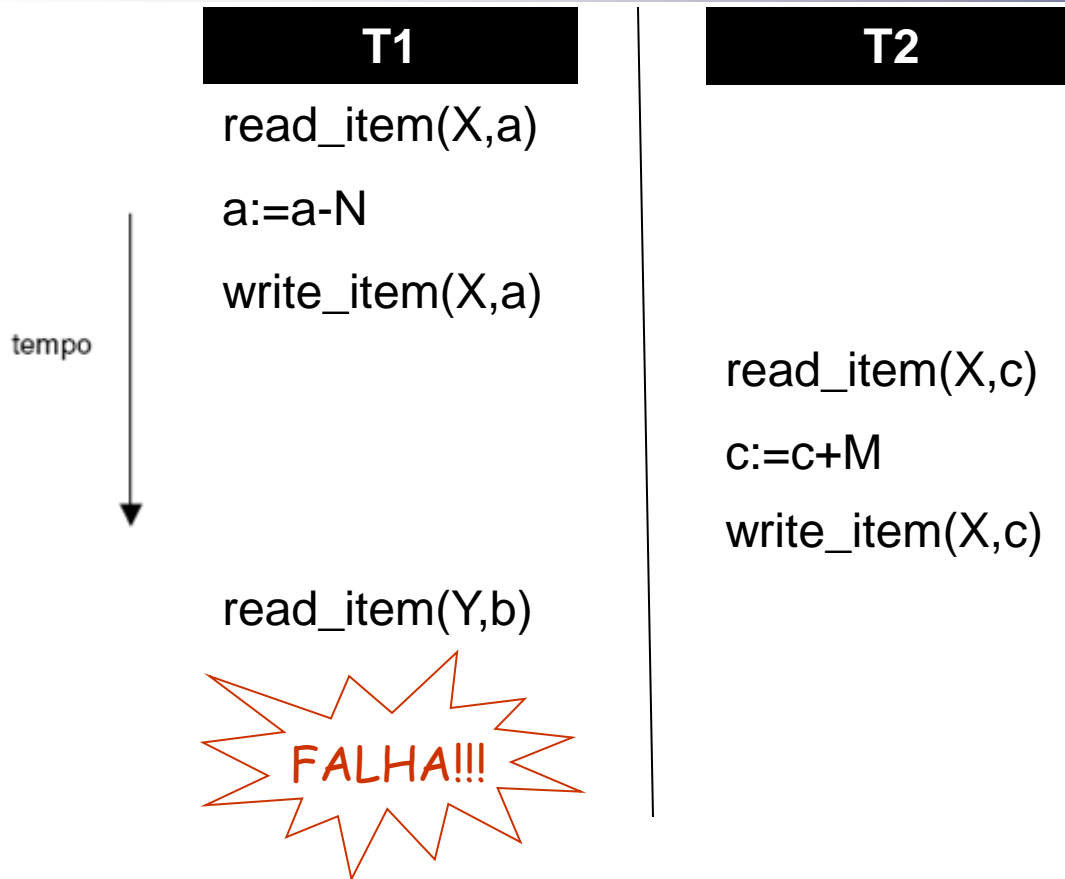


Escalonamento com entrelaçamento de operações



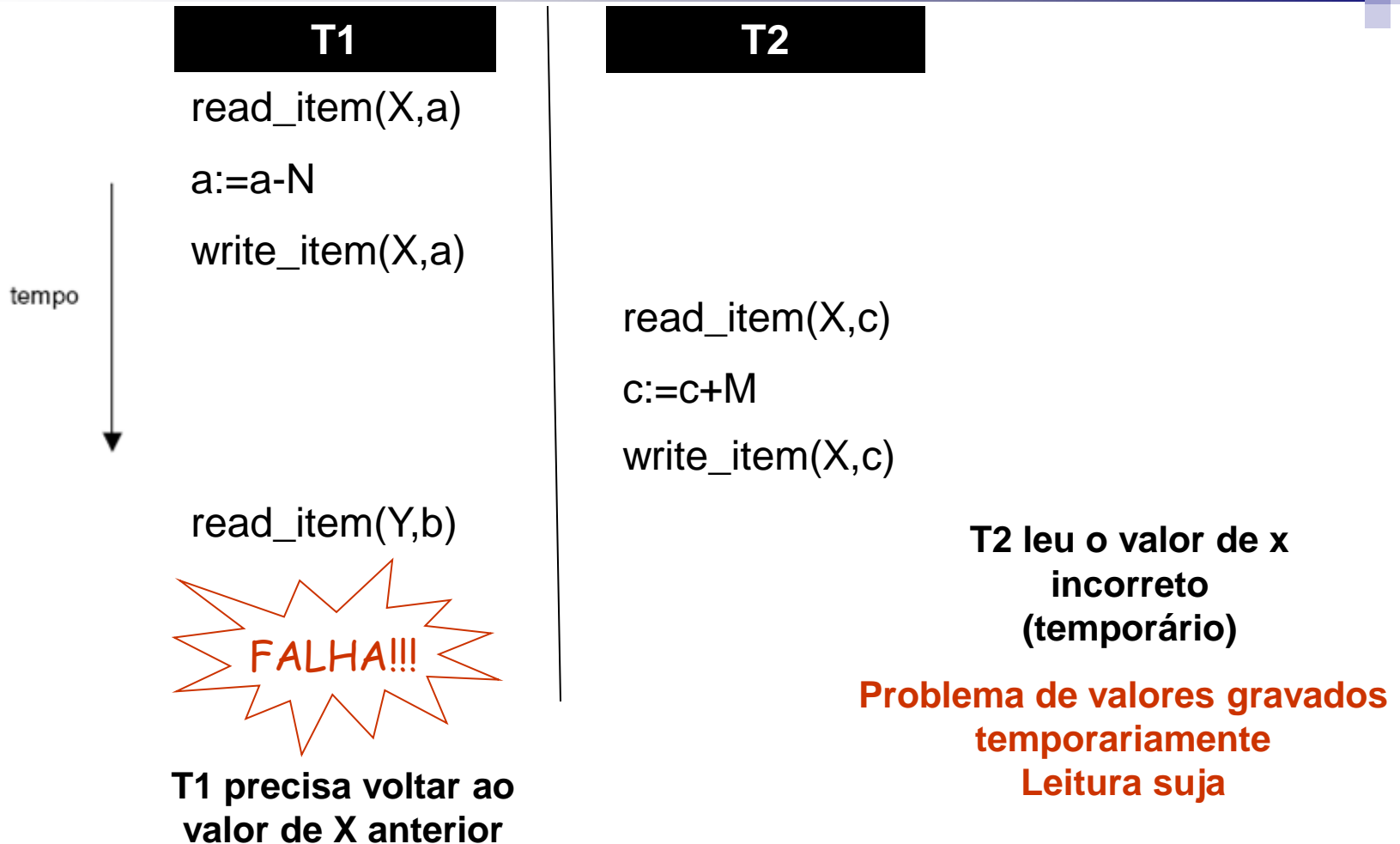


Escalonamento com entrelaçamento de operações





Escalonamento com entrelaçamento de operações





Testando a concorrência

- No PostgreSQL

- ☐ Abrir duas abas de “Query Tool”



Testando a concorrência

■ Atualização Perdida

- ☐ Ocorre quando duas transações que acessam os mesmos itens do banco de dados têm suas operações entrelaçadas, de modo que torne incorreto o valor de algum item.



Testando a concorrência

LOST UPDATE

TRANSAÇÃO 1		TRANSAÇÃO 2
1	START TRANSACTION	
2		START TRANSACTION
3	UPDATE northwind.categories SET description= 'avaliando a concorrência' WHERE categoryid = 4;	
4	SELECT * FROM northwind.categories;	
5		SELECT * FROM northwind.categories;
6		UPDATE northwind.categories SET description= 'concorrência ativa' WHERE categoryid = 4;
7	COMMIT	
8		COMMIT



Testando a concorrência

■ Leitura Suja

- ☐ Ocorre quando uma transação atualiza um item do banco de dados e, por algum motivo, outra transação ler esse item supostamente atualizado.



Testando a concorrência

DIRTY READ

TRANSAÇÃO 1		TRANSAÇÃO 2
1	START TRANSACTION	
2		START TRANSACTION
3	SELECT * FROM northwind.products WHERE productid = 1;	
4		SELECT * FROM northwind.products WHERE productid = 1;
5	UPDATE northwind.products SET unitsinstock = unitsinstock - 19 WHERE productid = 1;	
6		UPDATE northwind.products SET unitsinstock = unitsinstock - 29 WHERE productid = 1;
7	COMMIT	
8		COMMIT



Testando a concorrência

■ Leitura Não Repetitiva

- ☐ Ocorre quando uma transação lê itens em uma determinada condição e depois outra transação altera e efetiva um novo item que satisfaz a condição da transação anterior.
- ☐ Caso a primeira transação faça novamente uma leitura com a mesma condição, os registros podem aparecer de maneira diferente.



Testando a concorrência

Nonrepeatable read

TRANSAÇÃO 1		TRANSAÇÃO 2
1	START TRANSACTION	
2		START TRANSACTION
3		SELECT * FROM northwind.categories WHERE categoryid = 4;;
4	UPDATE northwind.categories SET categoryid = 40 WHERE categoryid = 4;	
5	SELECT * FROM northwind.categories;	
6		DELETE FROM northwind.categories WHERE categoryid = 4;
7	COMMIT	
8		COMMIT



Testando a concorrência

■ Leitura Fantasma

- ☐ Ocorre quando uma transação lê itens em uma determinada condição e depois outra transação insere e efetiva um novo item que satisfaz a condição da transação anterior.
- ☐ Caso a primeira transação faça novamente uma leitura com a mesma condição, um registro “fantasma” irá aparecer.



Testando a concorrência

Phanton Read

TRANSAÇÃO 1		TRANSAÇÃO 2
1	START TRANSACTION	
2		START TRANSACTION
3	INSERT INTO northwind.categories VALUES (100, 'Teste', 'teste', 0);	
4		SELECT * FROM northwind.categories;
5	COMMIT	
6		SELECT * FROM northwind.categories;
7		COMMIT



UNIVERSIDADE FEDERAL DE ITAJUBÁ

CONTROLE DE CONCORRÊNCIA



Controle de Concorrência

- Técnicas usadas para garantir a propriedade de não interferência ou isolamento das transações executadas concorrentemente.
- A maior parte dessas técnicas garante a serialização de *schedules* usando **protocolos de controle de concorrência**.



Controle de Concorrência

■ Técnicas

- Bloqueio (tranca) de itens de dados
 - É utilizado para evitar que múltiplas transações acessem os itens de dados concorrentemente
- Pré-ordenação (timestamp)
 - O sistema gera um identificador único (timestamp) para cada transação de forma que les fiquem ordenadas. Se T_i começa antes de T_j então $Ts(T_i) < Ts(T_j)$
- Multiversão
 - Utiliza múltiplas versões dos itens de dados
- Otimista
 - Baseado no conceito de validação (certificação) de uma transação.



Controle de Concorrência

■ Técnicas

☐ **Bloqueio**

- Binário
- Compartilhados/Exclusivos

☐ Rótulo de tempo (timestamp)

☐ Multiversão

☐ Validação ou certificação



Bloqueio

- Um bloqueio é uma variável associada a um item de dados que descreve o ***status*** do item em relação a possíveis operações que podem ser aplicadas a ele.
- Em geral, existe um bloqueio para cada item de dados no banco de dados.
- Os bloqueios são utilizados como um meio de sincronizar o acesso por transações concorrentes aos itens do banco de dados.



Bloqueio Binário

■ Bloqueio binário:

- ☐ Um bloqueio binário pode ter dois estados ou valores : bloqueado (1) e desbloqueado(0).
- ☐ Cada item do BD tem um valor associado. Se o valor for 1, o item não pode ser acessado por uma operação que o requisiite.
- ☐ A transação bloqueia objeto (*lock_item()*) antes de acessá-lo, liberando-o (*unlock_item()*) antes de terminar.



Bloqueio Binário

■ Operações:

☐ *lock_item(X)*

- Bloqueia o item X
- Aplicada antes de qualquer operação *read_item* e *write_item*

☐ *unlock_item(X)*

- Desbloqueia o item X
- Aplicada depois de qualquer operação *read_item* e *write_item*

■ Observações:

- ☐ As operações *lock* e *unlock* devem ser incluídas nas transações quando se utiliza o protocolo do bloqueio binário.



Bloqueio Binário

- Uma transação T deve executar uma operação de `lock_item(X)` antes de qualquer operação de `read_item(X)` ou `write_item(X)` ser executada em T .
- Uma transação T deve executar uma operação de `unlock_item(X)` depois de qualquer operação de `read_item(X)` ou `write_item(X)` ter sido executada completamente em T .
- Uma transação T não vai executar a operação de `lock_item(X)` se o item X já estiver bloqueado.
- Uma transação T não vai executar a operação de `unlock_item(X)` se o item X já estiver desbloqueado.



Bloqueio Binário

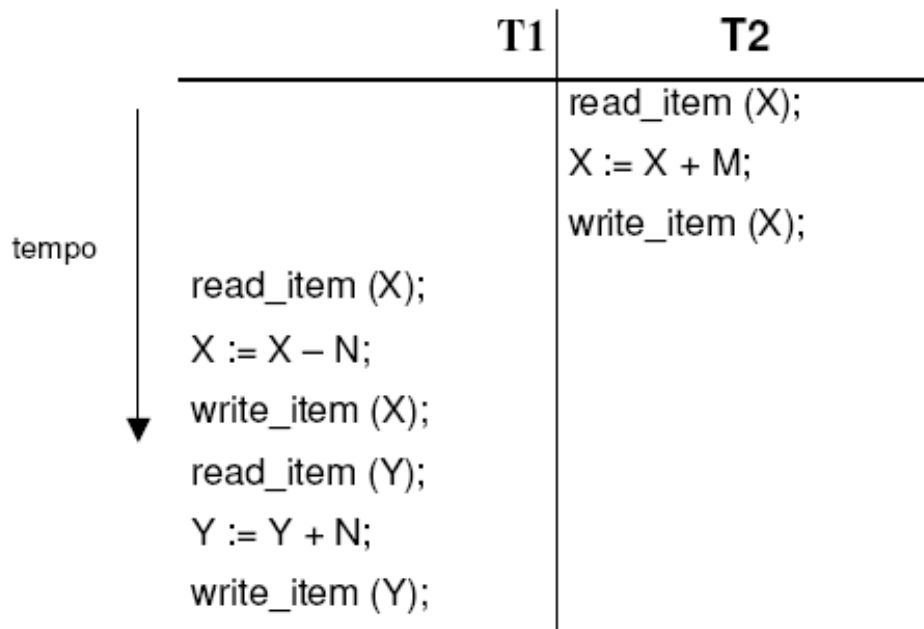
T1

T2

lock_item(X)
read_item(X,a)
unlock_item(X)
a:=a+M

lock_item(X)
write_item(X,a)
unlock_item(X)

lock_item(X)
read_item(X,a)
unlock_item(X)
a:=a-N
lock_item(X)
write_item(X,a)
unlock_item(X)
lock_item(Y)
...
unlock_item(Y)





Bloqueio Binário

T1

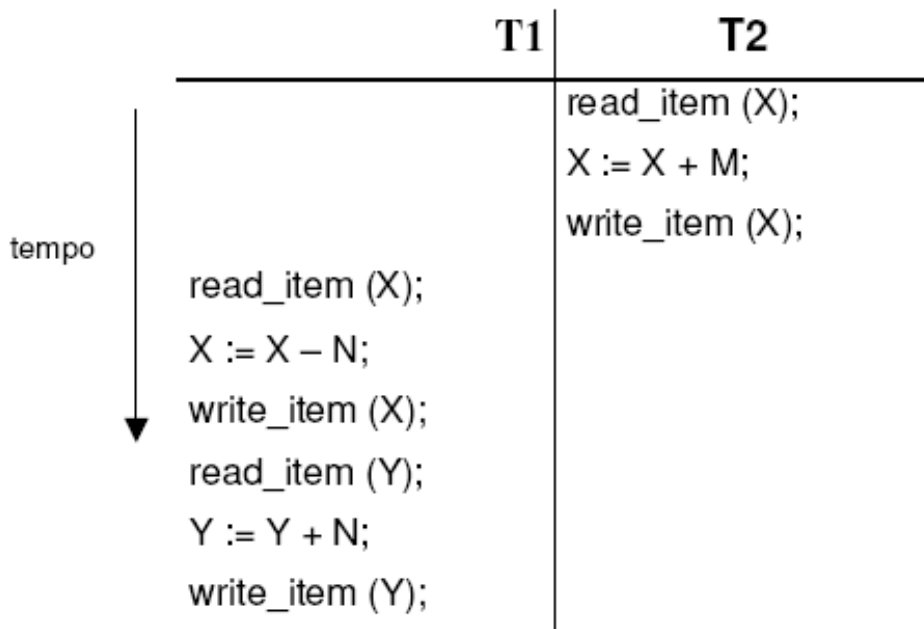
T2

T2 mantém o bloqueio do item X

lock_item(X)
read_item(X,a)
unlock_item(X)

a:=a+M

lock_item(X)
write_item(X,a)
unlock_item(X)



lock_item(X)
read_item(X,a)
unlock_item(X)
a:=a-N
lock_item(X)
write_item(X,a)
unlock_item(X)
lock_item(Y)
...
unlock_item(Y)



Bloqueio Binário

- Bloqueios binários são simples de implementar, mas muito restritivos para fins de controle de concorrência.
 - Porque, no máximo, uma transação pode manter um bloqueio em determinado item.
- Não são usados na prática.



Bloqueios compartilhados e exclusivos

- Operações de leitura no mesmo item por diferentes transações não estão em conflito.
- Assim, uma transação que utiliza bloqueio compartilhado/exclusivo utiliza bloqueios diferenciados para leitura e escrita de itens.
 - Um bloqueio associado a um item X , $\text{lock_item}(X)$, agora tem três estados:
 - Bloqueado para leitura
 - Bloqueado para escrita
 - Desbloqueado
- Bloqueio conhecido também como multi-modo ou leitura/gravação



Bloqueios compartilhados e exclusivos

■ Bloqueio compartilhado

- ☐ Bloqueia itens a serem **lidos**
- ☐ Mais de uma transação pode requerer um bloqueio compartilhado para ler um item X
- ☐ Nenhum bloqueio de escrita pode ser requerido por outra transação



Bloqueios compartilhados e exclusivos

■ Bloqueio exclusivo

- ☐ Para bloquear itens a serem **escritos**
- ☐ pode existir somente um bloqueio de escrita de um item X
- ☐ Nenhum bloqueio compartilhado pode ser requerido por outra transação



Bloqueios compartilhados e exclusivos

■ Operações:

□ *read_lock(X)*

- Bloqueia x na modalidade compartilhada
- Aplicada antes de qualquer operação *read_item* e *write_item*
- Abreviação - rl

□ *write_lock(X)*

- Bloqueia x na modalidade exclusiva
- Aplicada antes de qualquer operação *read_item* e *write_item*
- Abreviação - wl

□ *unlock_item(X)*

- Desbloqueia o item X
- Aplicada depois de qualquer operação *read_item* e *write_item*
- Abreviação - ul



Bloqueios compartilhados e exclusivos

■ Operações:

- Não é permitida a intercalação de *read_lock* (X) e *write_lock* (X) de diferentes transações
 - usa fila de espera



Bloqueios compartilhados e exclusivos

- Para uma transação T utilizar o bloqueio compartilhados e exclusivos:
 - T deve emitir um $rl(X)$ ou $wl(X)$ antes de qualquer $read_item(X)$ em T
 - T deve emitir um $wl(X)$ antes de qualquer $write_item(X)$ em T
 - T deve emitir $ul(X)$ depois de todo $read_item$ ou $write_item$ em T
 - T não emitirá um $rl(X)$ se já apareceu um $wl(X)$ em T
 - T não emitirá um $wl(X)$ se já apareceu um $rl(X)$ ou $wl(X)$ em T
 - T não emitirá um $ul(X)$ antes que apareça um $rl(X)$ ou $wl(X)$ em T



Bloqueios compartilhados e exclusivos

T1	T2
read_lock(x)	
read_item(x)	
Unlock_item(x)	
	write_lock(x)
	read_item(x)
	write_item(x)
	Unlock_item(x)
Write_lock(x)	
write_item(x)	
Unlock_item(x)	



Bloqueios compartilhados e exclusivos

T1	T2
read_lock(x)	
read_item(x)	read_lock(x)
unlock_item(x)	
	read_item(x)
	unlock_item(x)



Protocolo de bloqueio em duas fases – 2PL

- O uso de bloqueios binário ou de leitura/gravação não garante a serialização de escalonamentos por si só.
- Para garantir a serialização, temos que seguir um protocolo adicional em relação ao posicionamento das operações de bloqueio e desbloqueio em cada transação.
- O protocolo mais conhecido é o **bloqueio em duas fases**.



Protocolo de bloqueio em duas fases – 2PL

Diz-se que uma transação segue o protocolo de bloqueio em duas fases se todas as operações de bloqueio (read_lock, write_lock) precedem a primeira operação de desbloqueio da transação.



Protocolo de bloqueio em duas fases – 2PL

- Transação bloqueia objeto, na modalidade correta, antes de acessá-lo, liberando-o antes de terminar.
- Após liberar o primeiro objeto, a transação não bloqueia nenhum outro objeto.
- Ponto de Bloqueio de T - $pb(T)$
 - instante em que a transação libera o primeiro objeto



Protocolo de bloqueio em duas fases – 2PL

■ Por quê duas fases?

☐ Fase de expansão ou crescimento

- Novos bloqueios podem ser adquiridos, mas nenhum liberado

☐ Fase de encolhimento

- Bloqueios existentes podem ser liberados, mas nenhum novo bloqueio pode ser adquirido



Protocolo de bloqueio em duas fases

T1	T2
read_lock(Y)	read_lock(X)
read_item(Y,a)	read_item(X,c)
unlock(Y)	unlock(X)
write_lock(X)	write_lock(Y)
read_item(X,b)	read_item(X,d)
b:=b+a	d:=d+c
write_item(X,b)	write_item(X,d)
unlock(X)	unlock(X)

É bloqueio em duas fases?



Protocolo de bloqueio em duas fases

T1	T2
read_lock(Y)	read_lock(X)
read_item(Y,a)	read_item(X,c)
write_lock(X)	write_lock(Y)
unlock(Y)	unlock(X)
read_item(X,b)	read_item(X,d)
b:=b+a	d:=d+c
write_item(X,b)	write_item(X,d)
unlock(X)	unlock(Y)

É bloqueio em duas fases?



Protocolo de bloqueio em duas fases

T1	T2	
read_lock(Y)	read_lock(X)	Fase de expansão
read_item(Y,a)	read_item(X,c)	
write_lock(X)	write_lock(Y)	
unlock(Y)	unlock(X)	
read_item(X,b)	read_item(X,d)	Fase de recolhimento
b:=b+a	d:=d+c	
write_item(X,b)	write_item(X,d)	
unlock(X)	unlock(Y)	



Protocolo de bloqueio em duas fases – Exemplo 2

T1	T2
read_lock(x) read_item(x)	
	read_lock(y) read_item(y)
Unlock_item(x)	
	write_lock(x) Unlock_item(y) write_item(x) Unlock_item(x)
write_lock(x) write_item(x) Unlock_item(x)	

É serializável?

É bloqueio em duas fases?



Se toda transação em um escalonamento segue o protocolo 2PL, então a transação é serializável.



Métodos baseados em bloqueio

- Embora o protocolo 2PL garanta a serializabilidade, bloqueios podem gerar dois problemas:
 - impasse (*deadlock*)
 - inanição (*livelock*)



Métodos baseados em bloqueio

■ Deadlock

- ☐ Ocorre quando existem duas transações bloqueando o mesmo item e cada uma das duas está esperando que a outra libere este bloqueio.
- ☐ A solução mais comum para este problema de impasse é abortar uma das transações envolvidas.



Métodos baseados em bloqueio

- Existem diversas implementações diferentes para decidir qual transação deve ser interrompida no caso de um impasse, geralmente é escolhida a transação mais nova.
- Isso pode causar o *livelock* (quando o sistema não consegue decidir qual das transações deve ser abortada).
- É introduzida uma prioridade para as transações (por exemplo, maior prioridade para transações que já foram abortadas anteriormente).



Controle de Concorrência

■ Técnicas

- Bloqueio (tranca) de itens de dados
 - É utilizado para evitar que múltiplas transações acessem os itens de dados concorrentemente
- Pré-ordenação (timestamp)
 - O sistema gera um identificador único (timestamp) para cada transação de forma que les fiquem ordenadas. Se T_i começa antes de T_j então $Ts(T_i) < Ts(T_j)$
- Multiversão
 - Utiliza múltiplas versões dos itens de dados
- Otimista
 - Baseado no conceito de validação (certificação) de uma transação.



Para Casa



- Ler o **capítulo 18** do livro Sistemas de Banco de Dados – 4^a ed.
 - Elmasri e Navathe