UNIVERSIDADE FEDERAL DE ITAJUBÁ

# Banco de Dados II
## COM 231

**Controle de Concorrência**
**Aula 13**

**Vanessa Cristina Oliveira de Souza**

# Classificação dos SGBD´s quanto ao número de usuários suportados

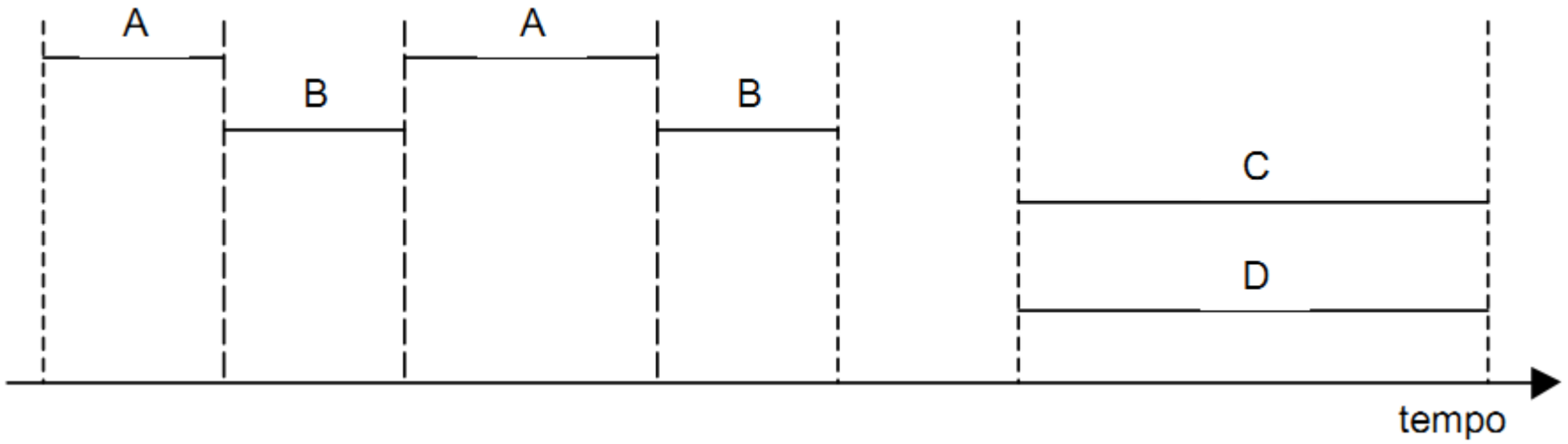
- Mono-usuários
  - no máximo um usuário pode usar o mesmo sistema por vez
- Multi-usuários
  - muitos usuários podem usar o mesmo sistema coincidentemente
  - Para garantir a integridade do BD é necessário usarmos o conceito de Transações '**serializáveis**'

# Concorrência

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

# Problemas causados por falta de Isolamento

- Perda de consistência do banco
- Acesso a dados inconsistentes
  - Leitura suja
- Perda de atualizações

# Escalonamentos

- Um escalonamento (***schedule***) *S* de *n* transações *T1*, *T2*, ..., *Tn* é uma ordenação das operações dessas transações, tal que, para cada transação *Ti*, as operações de *Ti* em *S* devem aparecer na mesma ordem em que elas aparecem em *Ti*.

- 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:
  - Sa: r1(X); r2(X); w1(X); r1(Y); w2(X); c2; w1(Y);
  - ou
  - Sb: 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) |

**<$T_1$>**
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 Serializável

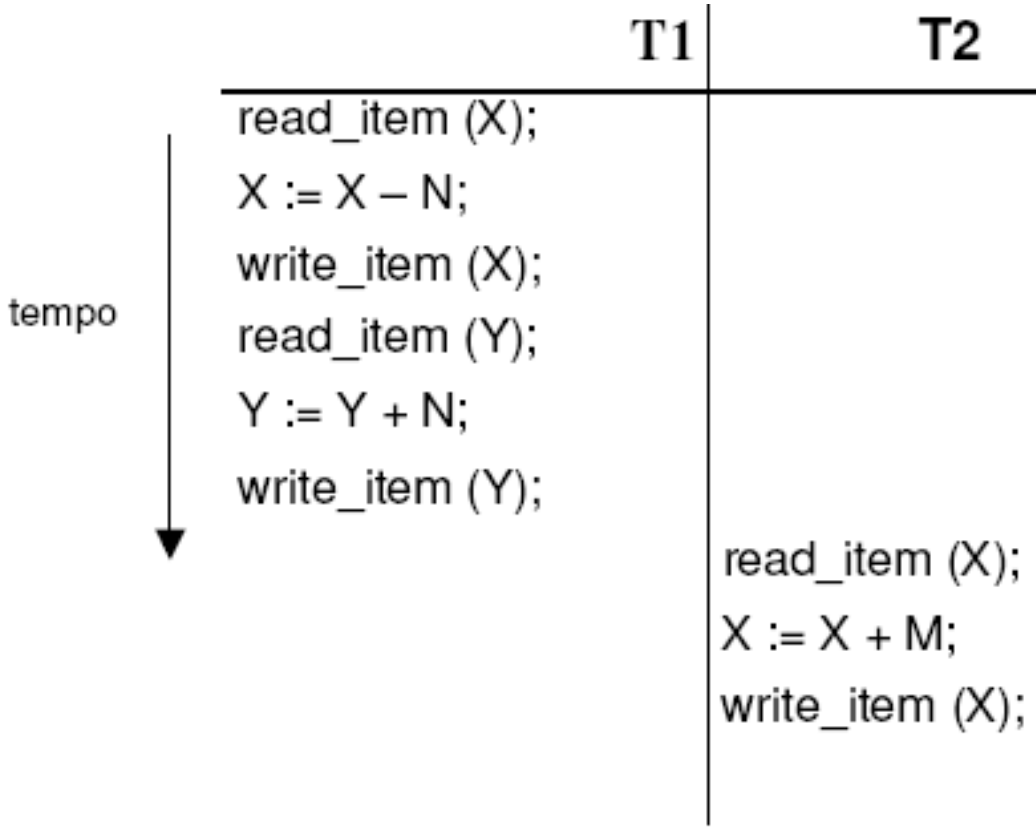
- Um escalonamento *S* é serial se, para toda transação *T* participante de *S*, todas as operações de *T* são executadas consecutivamente em *S*, caso contrário *S* é não serial.

- Um escalonamento *S* é serializável se for equivalente a um escalonamento serial *S'*.

# Escalonamento

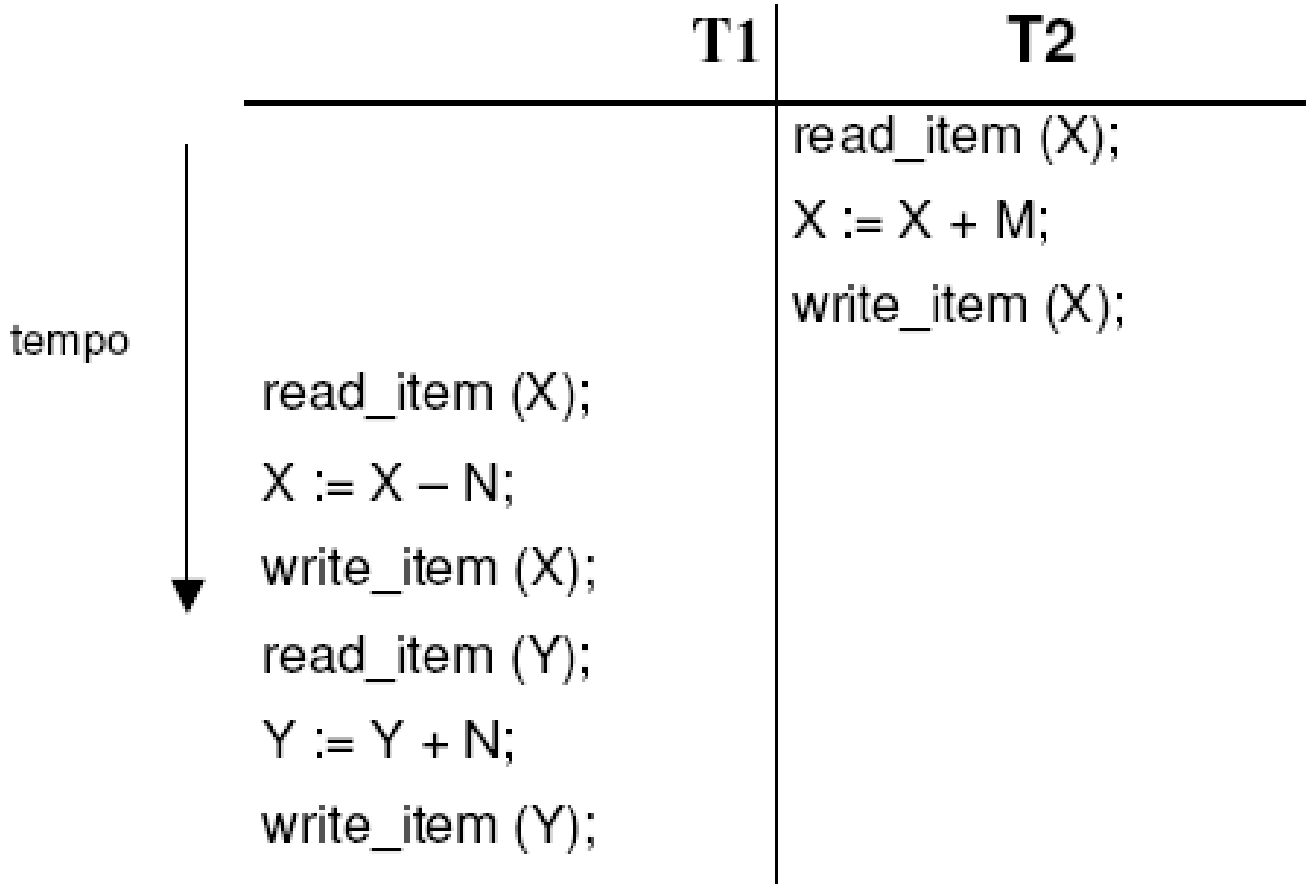
| T1 | T2 |
| --- | --- |
| read_item(X,a) | read_item(X,c) |
| a:=a-N | c:=c+M |
| write_item(X,a) | write_item(X,c) |
| read_item(Y,b) | |
| b := b + N | |
| write_item(Y,b) | |

# 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

| T1 | T2 |
| --- | --- |
| read_item(X,a) | |
| a:=a-N | |
| | read_item(X,c) |
| | c:=c+M |
| write_item(X,a) | |
| read_item(Y,b) | |
| | write_item(X,c) |
| b := b + N | |
| write_item(Y,b) | |

tempo

# Escalonamento com entrelaçamento de operações

| T1 | T2 |
| --- | --- |
| read_item(X,a) | |
| a:=a-N | |
| | read_item(X,c) |
| | c:=c+M |
| write_item(X,a) | |
| read_item(Y,b) | |
| | write_item(X,c) |
| b := b + N | |
| write_item(Y,b) | |

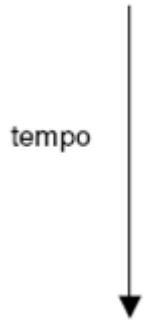
tempo

Problema de atualização perdida

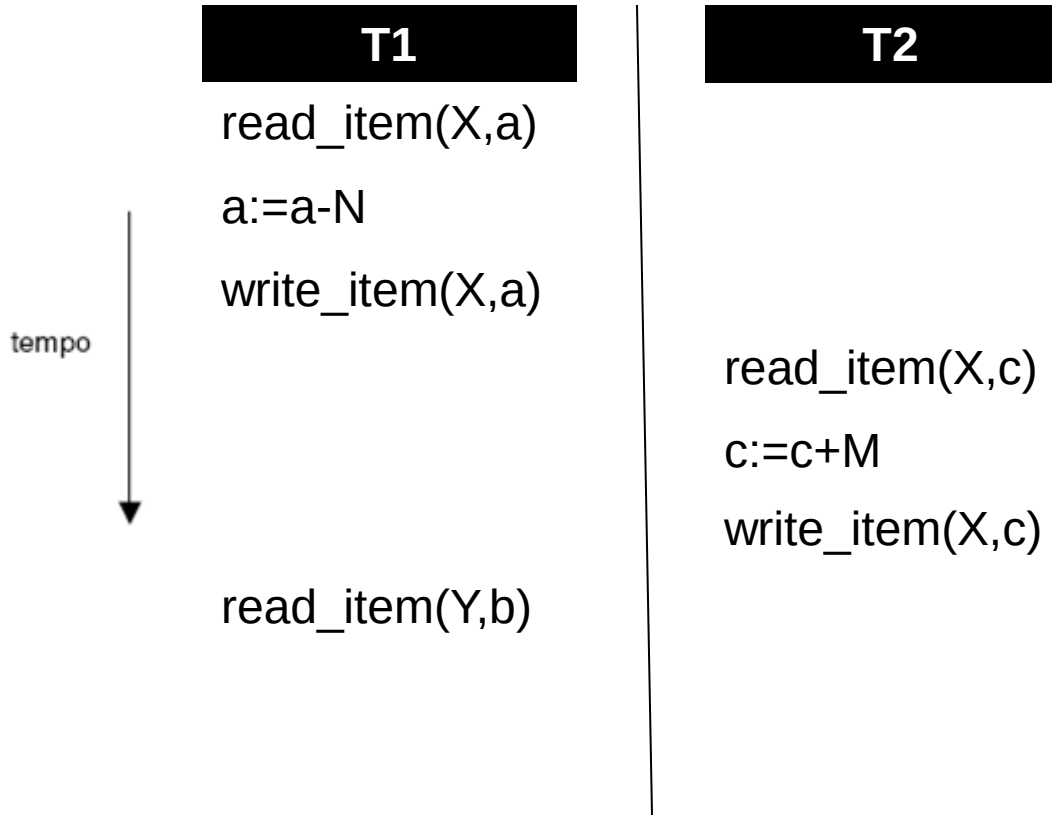
**O item x tem o valor incorreto porque sua alteração em T1 foi perdida (escrita por cima)**

# Escalonamento com entrelaçamento de operações

| T1 | T2 |
| --- | --- |
| read_item(X,a) | |
| a:=a-N | |
| write_item(X,a) | |
| | read_item(X,c) |
| | c:=c+M |
| | write_item(X,c) |
| read_item(Y,b) | |
| b := b + N | |
| write_item(Y,b) | |



# Escalonamento com entrelaçamento de operações



# Escalonamento com entrelaçamento de operações

| T1 | T2 |
| --- | --- |
| read_item(X,a) | |
| a:=a-N | |
| write_item(X,a) | |
| | read_item(X,c) |
| | c:=c+M |
| | write_item(X,c) |
| read_item(Y,b) | |

tempo

# Escalonamento com entrelaçamento de operações

| T1 | T2 |
| --- | --- |
| read_item(X,a) | |
| a:=a-N | |
| write_item(X,a) | |
| | read_item(X,c) |
| | c:=c+M |
| | write_item(X,c) |
| read_item(Y,b) | |

tempo


**T1 precisa voltar ao valor de X anterior**

**T2 leu o valor de x incorreto (temporário)**

**Problema de valores gravados temporariamente**
**Leitura suja**

# Grafo de Precedência

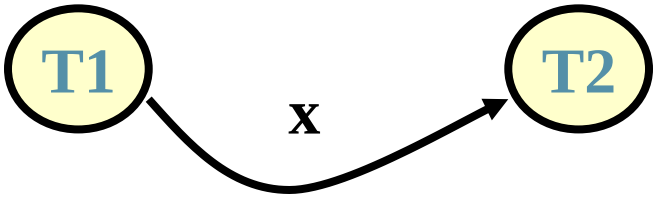
- O grafo de precedência é construído para testar se um determinado escalonamento é serializável ou não.

- Caso seja serializável este é um escalonamento seguro, caso contrário ele é conflitante.

# Escalonamento Serializável

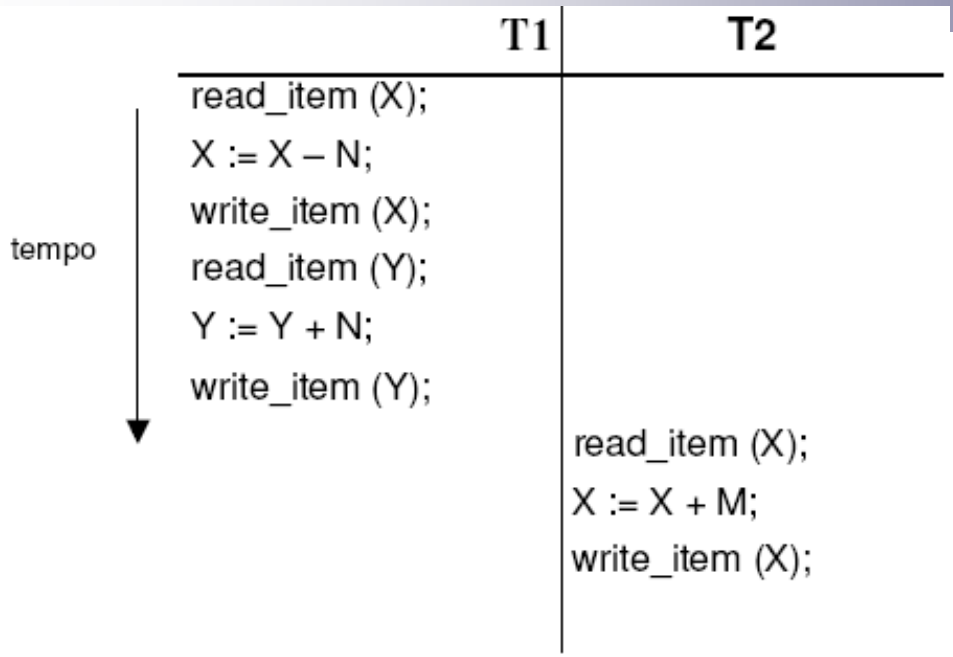
| T1 | T2 |
|---|---|
| read_item (X); | |
| X := X – N; | |
| write_item (X); | |
| read_item (Y); | |
| Y := Y + N; | |
| write_item (Y); | |
| | read_item (X); |
| | X := X + M; |
| | write_item (X); |

tempo




- Grafo de precedência de um escalonamento serializável da transação T1 seguida pela transação T2

# Grafo de Precedência




**1 – Criar um nó para cada transação**

2 –T1 faz um read_item(item) antes de T2 fazer um write_item(item)? E vice-versa**?**

3 –T1 faz um write_item(item) antes de T2 fazer um read_item(item)? E vice-versa?

4 –T1 faz um write_item(item) antes de T2 fazer um write_item(item)? E vice-versa?

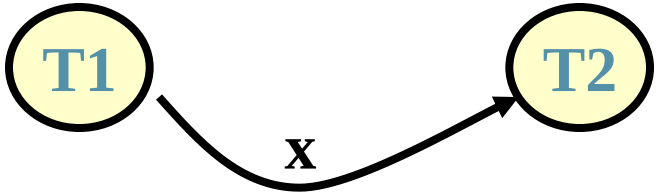
O grafo possui ciclos?

# Grafo de Precedência

| T1 | T2 |
|---|---|
| read_item (X); | |
| X := X – N; | |
| write_item (X); | |
| read_item (Y); | |
| Y := Y + N; | |
| write_item (Y); | |
| | read_item (X); |
| | X := X + M; |
| | write_item (X); |

tempo

Escalonamento Serializável




1 – Criar um nó para cada transação

**2 –T1 faz um read_item(item) antes de T2 fazer um write_item(item)? E vice-versa?**

3 –T1 faz um write_item(item) antes de T2 fazer um read_item(item)? E vice-versa?

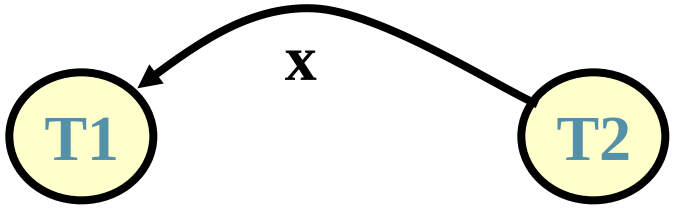
4 –T1 faz um write_item(item) antes de T2 fazer um write_item(item)? E vice-versa?

O grafo possui ciclos?

# Escalonamento Serializável

| T1 | T2 |
|---|---|
| | read_item (X); |
| | X := X + M; |
| | write_item (X); |
| read_item (X); | |
| X := X – N; | |
| write_item (X); | |
| read_item (Y); | |
| Y := Y + N; | |
| write_item (Y); | |

tempo



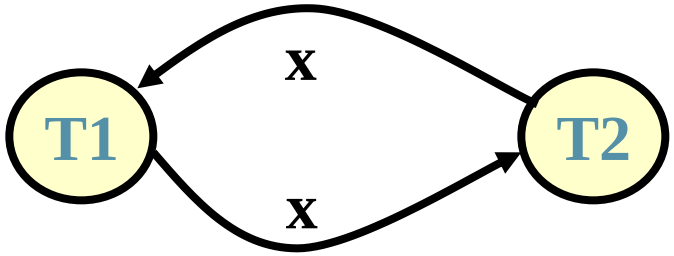

- Grafo de precedência de um escalonamento serializável da transação T2 seguida pela transação T1

# Grafo de Precedência

| T1 | T2 |
|---|---|
| read_item (X); | |
| X := X – N; | |
| | read_item (X); |
| | X := X + M; |
| write_item (X); | |
| read_item (Y); | |
| | write_item (X); |
| Y := Y + N; | |
| write_item (Y); | |

tempo

Escalonamento
Não Serializável




1 – Criar um nó para cada transação

2 –T1 faz um read_item(item) antes de T2 fazer um write_item(item)? E vice-versa?

3 –T1 faz um write_item(item) antes de T2 fazer um read_item(item)? E vice-versa?

4 –T1 faz um write_item(item) antes de T2 fazer um write_item(item)? E vice-versa?

**O grafo possui ciclos?**

# Exercício

- Dada as transações T5, T6 e T7, sendo executadas concorrentemente, como na figura abaixo. O escalonamento escolhido é serializável?

| T5 | T6 | T7 |
|---|---|---|
| Read(A) | | |
| | Read(A) | |
| Write(B) | | |
| | Write(A) | |
| | | Read(B) |
| | | Write(A) |

*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 Ti começa antes de Tj então Ts(Ti) < Ts(Tj)
  - 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 requisite.
  - A transação bloqueia objeto (*lock_item*()) antes de acessá-lo, liberando-o (*unlock_item*()) antes de terminar.

# Bloqueio Binário

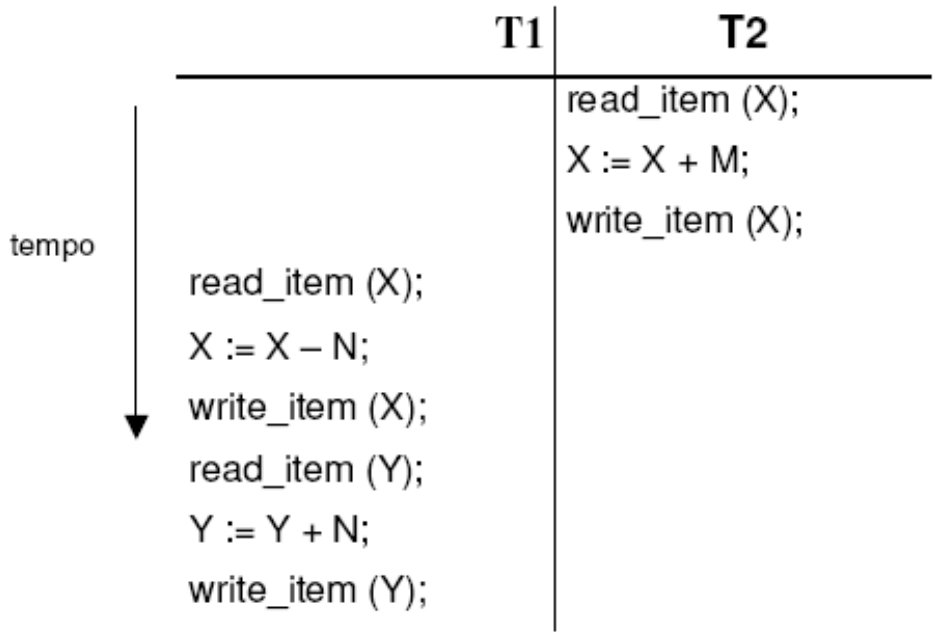
- Operações:
  - *lock_item(X)*
    - Bloqueia o item X
    - Aplicada antes de qualquer operação read_item e write_item
  - *unlock_item(X)*
    - Desbloqueia o item X
    - Aplicada depois de qualquer operação read_item e write_item
- Observações:
  - As operações lock e unlock devem ser incluídas 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.

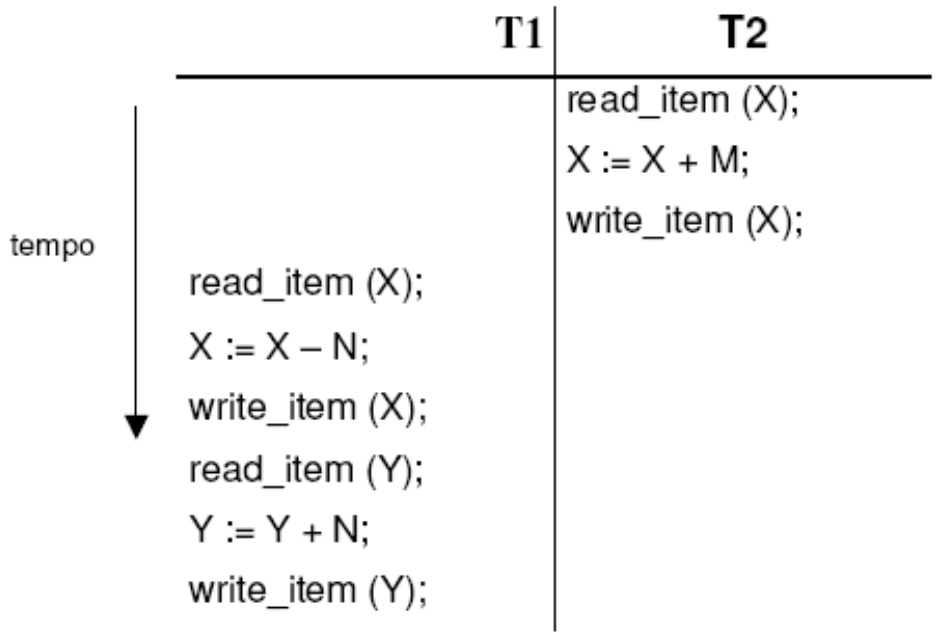
# Bloqueio Binário

| 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, 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

| 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_lock(x) |
| read_item(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) | |
| read_item(Y,a) | read_item(X,c) | Fase se expansão |
| write_lock(X) | write_lock(Y) | |
| unlock(Y) | unlock(X) | |
| read_item(X,b) | read_item(X,d) | |
| b:=b+a | d:=d+c | Fase se recolhimento |
| 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).

# Exercícios
# Entregar na próxima aula

- Dada as transações abaixo:

  T1: read_item(A); read_item(B); write_item(A); write_item(B)
  T2: read_item(C); read_item(A); write_item(A)

- Reescreva os códigos das transações dadas acima para que eles fiquem de acordo com os seguintes protocolos:
  - Protocolo do bloqueio binário
  - Protocolo do bloqueio em duas fases com bloqueio compartilhado/exclusivo

- Esquematize um escalonamento serializável com bloqueio compartilhado/exclusivo

- Esquematize um escalonamento não serializável com bloqueio compartilhado/exclusivo

*UNIVERSIDADE FEDERAL DE ITAJUBÁ*

# BANCOS DE DADOS RELACIONAIS

# Bancos de Dados Relacionais

- **SÃO TRANSACIONAIS**
  - Atomicidade
  - Consistência
  - Isolamento
  - Durabilidade

**Por meio das técnicas de recuperação de falhas, bloqueio e por seu modelo rígido, garantem SEMPRE as propriedades ACID de cada transação**