

Apresentação baseada em:

- Ramez Elmasri e Shamkant B. Navathe 6° Ed (2010)
- Material da Profa. Dra. Sarajane Marques Peres (UNIVESP)
- Material do Prof. Dr. André Santanchè (UNICAMP)

Banco de Dados

Controle de concorrência (parte 1)

FACOM – UFMS

Vanessa Borges

vanessa.a.borges@ufms.br

- Capítulo 22: técnicas de controle de concorrência

Propriedade das transações - ACID

- **Isolamento**

- Transações são isoladas umas das outras
- Garante que transações executadas concorrentemente sejam isoladas umas das outras, de modo que cada uma tenha a impressão de que nenhuma outra transação está sendo executada concorrentemente a ela

T1

```
read_item(A);  
A:=A-50  
write_item(A);  
read_item(B);  
B:=B+50  
write_item(B);
```

T2

```
read_item(A);  
A:=A*1,5;  
write_item(A);
```

Assegurar o **ISOLAMENTO** de uma transação é **responsabilidade do Controle de Concorrência**.



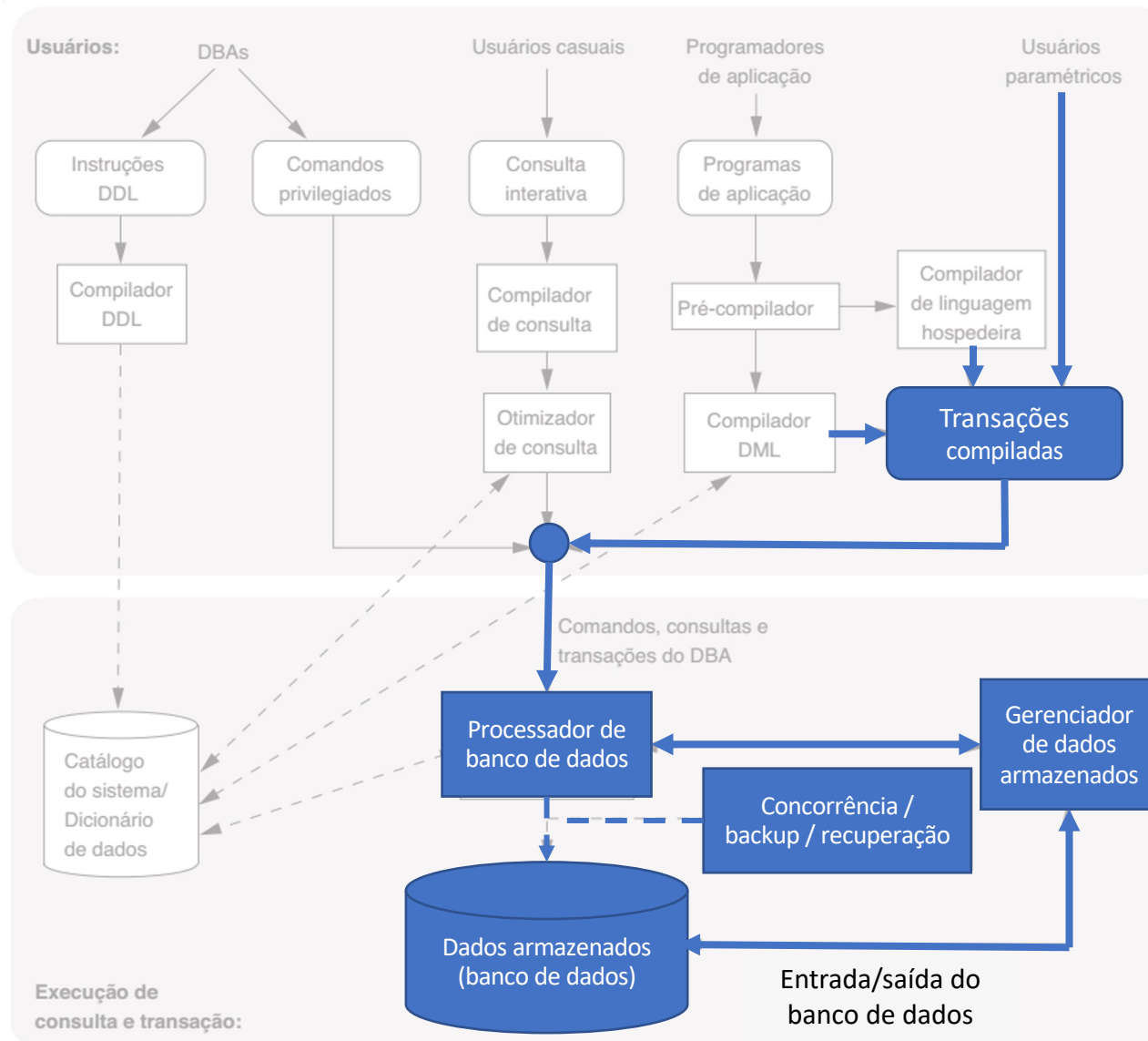
Controle de Concorrência

- O **ISOLAMENTO** é uma das propriedades fundamentais de uma transação.
 - Quando diversas transações são executadas de **modo concorrente**, há o risco de ocorrer violação dessa propriedade
 - É necessário que o sistema controle a forma como as operações das transações são intercaladas durante a execução concorrente. Esse controle é realizado pelo **módulo de controle de concorrência**
 - Os o controle de concorrência têm por base a **propriedade de serialização**, ou seja, os esquemas devem garantir que a ordenação da execução das operações das diferentes transações concorrentes seja serializável



Controle de concorrência

Componentes do SGBD



Protocolos de controle de concorrência

- Uma das formas de garantir que apenas escalonamentos serializáveis sejam produzidos é **obrigar que o acesso aos itens de dados seja feito de maneira mutuamente exclusiva**:
 - Enquanto uma transação acessa um item de dados, nenhuma outra transação pode modificá-lo.
 - Para implementar isso pode-se usar o **método de bloqueio (*lock*)**.



Bloqueio

- Um bloqueio (***lock***) é uma variável associada a um item de dado
 - Descreve a condição do item em relação às possíveis operações que podem ser aplicadas a ele
 - Geralmente há um bloqueio para cada item de dado
- Bloqueios a serem analisados
 - Bloqueio binário
 - Bloqueio compartilhado/exclusivo (leitura/gravação)



Protocolo baseado em bloqueios

Bloqueio Binário

- Possui dois estados
 - Bloqueado (***locked***)
 - O item não pode ser acessado quando solicitado
 - Desbloqueado (***unlocked***)
 - O item pode ser acessado quando solicitado
- Operações atômicas de bloqueio binário
 - **lock_item(X)**
 - **unlock_item(X)**



Protocolo baseado em bloqueios

Bloqueio Binário

- Operação de **lock_item(X)**

lock_item(X):

B: se $LOCK(X) = 0$ (* item está desbloqueado *)

então $LOCK(X) \leftarrow 1$ (* bloqueia o item *)

se não

início

wait (until $LOCK(X) = 0$

e o gerenciador de bloqueio desperta a transação);

go to **B**

fim;



Protocolo baseado em bloqueios

Bloqueio Binário

- Operação de **unlock_item(X)**

unlock_item(X):

LOCK(X) \leftarrow 0; (* desbloqueia o item *)

se alguma transação estiver esperando

então acorda uma das transações em espera;



Protocolo baseado em bloqueios

Bloqueio Binário

- Regras
 - Para cada transação
 - **lock_item(X)**
 - Antes de read_item(X) ou write_item(X) serem realizadas em T
 - Se X ainda não possuir o lock
 - **unlock_item(X)**
 - Depois de todas as operações
 - Apenas se possuir o lock de X



Protocolo baseado em bloqueios

Bloqueio Binário

- **Conclusões**

- Simples mas muito restritivo
 - Desnecessário o bloqueio quando existe apenas operação de leitura
- Não é utilizado na prática



Protocolo baseado em bloqueio

Bloqueio Compartilhado / Exclusivo

- É uma variação do bloqueio binário
- Um bloqueio associado ao item X possui três estados possíveis
 - **Bloqueado para leitura (*read lock*) - bloqueio compartilhado (*shared lock*):**
 - Se uma transação T_i obteve um bloqueio **compartilhado** sobre o item de dado X, então T_i **pode ler**, mas **não pode escrever** em X.
 - Mais de uma transação pode empregá-lo
 - **Bloqueado para gravação (*write lock*) - bloqueio exclusivo (*exclusive lock*):**
 - Se uma transação T_i obteve um bloqueio **exclusivo** do item de dado X, então T_i **pode tanto ler como escrever** em X.
 - Somente uma transação pode solicitá-lo
 - **Desbloqueado**

Matriz de compatibilidade

	Compartilhado	Exclusivo
Compartilhado	Sim	Não
Exclusivo	Não	Não



Protocolo baseado em bloqueio

Bloqueio Compartilhado / Exclusivo

- Operação de **read_lock(X) / shared_lock(X)**

read_lock(X):

B: se LOCK(X) = “unlocked”

então **início** LOCK(X) ← “read-locked”;

num_de_leituras(X) ← 1

fim

se não se LOCK(X) = “read-locked”

então num_de_leituras(X) ← num_de_leituras(X) + 1

se não **início**

wait (até que LOCK(X) = “unlocked”

e o gerenciador de bloqueio desperta a transação);

go to **B**

fim;



Protocolo baseado em bloqueio

Bloqueio Compartilhado / Exclusivo

- Operação de **write_lock(X)/ exclusive_lock(X)**

write_lock(X):

B: se LOCK(X) = “unlocked”

então LOCK(X) ← “write-locked”

então **início**

wait (até que LOCK(X) = “unlocked”

e o gerenciador de bloqueio desperta a transação);

go to **B**

fim;



Protocolo baseado em bloqueio

Bloqueio Compartilhado / Exclusivo

- Operação de **unlock(X)**

unlock (X):

se $LOCK(X) = \text{"write-locked"}$

então **início** $LOCK(X) \leftarrow \text{"unlocked"}$;

desperta uma das transações aguardando, se houver

fim

se não se $LOCK(X) = \text{"read-locked"}$

então **início**

$num_de_leituras(X) \leftarrow num_de_leituras(X) - 1$;

se $num_de_leituras(X) = 0$

então **início** $LOCK(X) = \text{"unlocked"}$;

desperta uma das transações aguardando, se houver

fim

fim;



Protocolo baseado em bloqueio

Bloqueio Compartilhado / Exclusivo

- Regras

1. Uma transação T precisa emitir a operação `read_lock(X)` ou `write_lock(X)` antes que qualquer operação `read_item(X)` seja realizada em T.
2. Uma transação T precisa emitir a operação `write_lock(X)` antes que qualquer operação `write_item(X)` seja realizada em T.
3. Uma transação T precisa emitir a operação `unlock(X)` após todas as operações `read_item(X)` e `write_item(X)` serem completadas em T.
4. Uma transação T **não** emitirá uma operação `read_lock(X)` se ela já mantiver um bloqueio de leitura (compartilhado) ou um bloqueio de gravação (exclusivo) no item X. **Essa regra pode ser flexível, conforme discutiremos em breve.**
5. Uma transação T **não** emitirá uma operação `write_lock(X)` se ela já mantiver um bloqueio de leitura (compartilhado) ou um bloqueio de gravação (exclusivo) no item X. **Essa regra pode ser flexível, conforme discutiremos em breve.**
6. Uma transação T **não** emitirá uma operação `unlock(X)` a menos que já mantenha um bloqueio de leitura (compartilhado) ou um bloqueio de gravação (exclusivo) no item X.



Protocolo baseado em bloqueio

Bloqueio Compartilhado / Exclusivo (UPGRADE / DOWNGRADE)

- **Conversão de bloqueios**

- Foi descrito com o objetivo de flexibilizar os itens 4 e 5 das regras de bloqueio compartilhado / exclusivo

- **LOCK UPGRADE**

- $\text{read_lock}(X) \rightarrow \text{write_lock}(X)$
 - Condição: não há outro read_lock em X

- **LOCK DOWNGRADE**

- $\text{write_lock}(X) \rightarrow \text{read_lock}(X)$



Exemplo: Bloqueio Compartilhado / Exclusivo

Serialização

Valores iniciais: **X= 20, Y=30**

Plano Serial - A

T1	T2
read_item(Y); read_item(X); X=X+Y; write_item(X);	 read_item(X); read_item(Y); Y=X+Y; write_item(Y);

Resultado: X=50, Y=80

Plano Serial - B

T1	T2
 read_item(Y); read_item(X); X=X+Y; write_item(X);	read_item(X); read_item(Y); Y=X+Y; write_item(Y);

Resultado: X=70, Y=50

Plano Intercalado - S

T1	T2
read_item(Y); read_item(X); X=X+Y; write_item(X);	 read_item(X); read_item(Y); Y=X+Y; write_item(Y);

**Resultado: X=50, Y=50
(Não serializável)**

Exemplo: Bloqueio Compartilhado / Exclusivo

Serialização

T1
read_item(Y); read_item(X); X=X+Y; write_item(X);

T2
read_item(X); read_item(Y); Y=X+Y; write_item(Y);

T1	T2	Controle de concorrência
read_lock(Y); read_item(Y); unlock(Y);		grant-read_lock(Y, T1)
	read_lock(X); read_item(X); unlock(X);	grant-read_lock(X, T2)
	write_lock(Y); read_item(Y); Y=X+Y; write_item(Y); unlock(Y);	grant-write_lock(Y, T2)
write_lock(X); read_item(X); X=X+Y; write_item(X); unlock(X);		grant-write_lock(X, T1)

Não é serializável.

Não há ocorrência de impasses/deadlocks.

Controle de concorrência

Protocolos de bloqueio em duas fases

- Protocolo de bloqueio em duas fases (*two-phase locking protocol* – 2PL)
 - Esse protocolo exige que cada transação emita solicitações de bloqueio e desbloqueio em duas fases:
 - **FASE DE EXPANSÃO:** uma transação pode obter bloqueios, mas não pode libertar nenhum
 - **FASE DE ENCOLHIMENTO:** uma transação pode liberar bloqueios, mas não consegue obter nenhum bloqueio novo.
 - Esse protocolo garante apenas a geração de esquemas de execução concorrente serializáveis. Ele não garante ausência de *deadlocks*.



Exemplo: Bloqueio em duas fases

Ocorrência de *Deadlock*

T1
read_item(Y); read_item(X); X=X+Y; write_item(X);

T2
read_item(X); read_item(Y); Y=X+Y; write_item(Y);

T1	T2	Controle de concorrência
read_lock(Y); read_item(Y); write_lock(X); unlock(Y); read_item(X); X=X+Y; write_item(X); unlock(X);	 read_lock(X); read_item(X); write_lock(Y); unlock(X); read_item(Y); Y=X+Y; write_item(Y); unlock(Y);	grant-read_lock(Y, T1); grant-read_lock(X, T2); wait(Y, T1); wait(X, T2);



Exemplo: Bloqueio em duas fases

Ocorrência de *Deadlock*

T1
read_item(Y); read_item(X); X=X+Y; write_item(X);

T2
read_item(X); read_item(Y); Y=X+Y; write_item(Y);

T1	T2	Controle de concorrência
read_lock(Y); read_item(Y); write_lock(X); unlock(Y); read_item(X); X=X+Y; write_item(X); unlock(X);	read_lock(X); read_item(X); write_lock(Y); unlock(X); read_item(Y); Y=X+Y; write_item(Y); unlock(Y);	grant-read_lock(Y, T1); grant-read_lock(X, T2); wait(Y, T1); wait(X, T2);

Deadlock!



Controle de concorrência

Protocolos de bloqueio

- Outros esquemas de controle de concorrência:
 - Protocolo de bloqueio em duas fases SEVERO
 - Protocolo de bloqueio em duas fases RIGOROSO
 - Protocolo de bloqueio de granularidade múltipla
 - Protocolo de bloqueio baseado em *timestamp*
 - Esquema de multiversão



Apresentação baseada em:

- Ramez Elmasri e Shamkant B. Navathe 6° Ed (2010)
- Material da Profa. Dra. Sarajane Marques Peres (UNIVESP)
- Material do Prof. Dr. André Santanchè (UNICAMP)

Banco de Dados

Introdução aos conceitos de Processamento de Transação (parte 2)

FACOM – UFMS

Vanessa Borges

vanessa.a.borges@ufms.br

- Capítulo 22: técnicas de controle de concorrência

Controle de Concorrência

Tratamento de Deadlock

O **deadlock** (impasse) ocorre quando cada transação T em um conjunto de duas ou mais transações está esperando por algum item que está bloqueado por alguma outra transação T' no conjunto.

É necessário prevenir ou detectar a ocorrência de deadlocks

T1	T2	Controle de concorrência
read_lock(Y); read_item(Y); write_lock(X); unlock(Y); read_item(X); X=X+Y; write_item(X); unlock(X);	read_lock(X); read_item(X); write_lock(Y); unlock(X); read_item(Y); Y=X+Y; write_item(Y); unlock(Y);	grant-read_lock(Y, T1); grant-read_lock(X, T2); wait(Y, T1); wait(X, T2); <i>Deadlock!</i>

Controle de Concorrência

- Existem duas abordagens principais para tratamento de *deadlock*
 - **Prevenção de *deadlock*:**
 - Forma de impedir o *deadlock* por meio de protocolos de prevenção
 - Garante que o sistema nunca entrará em situação de *deadlock*
 - *Deadlocks* ocorrem com maior frequência (pessimista)
 - **Deteção e recuperação de *deadlock*:**
 - Permite que o sistema entre em um estado de *deadlock* e então o remove desse estado recuperando-o.
 - *Deadlocks* não ocorrem com frequência (otimista)



Controle de Concorrência

Prevenção de *deadlock*

- **Abordagem 1**

- Utilizado no 2PL conservador
- Obriga que cada transação bloqueie **todos os itens de dados antes de sua execução.**
 - Se qualquer um dos itens não puder ser obtido, nenhum item é bloqueado (a transação entra em estado de espera).
- **Esse esquema limita a concorrência**



Controle de Concorrência

Prevenção de *deadlock*

- **Abordagem 2**

- Utiliza o conceito de ***timestamp*** (selo de tempo)
 - Quando uma transação T2 solicita o bloqueio que está sendo mantido pela transação T1, o bloqueio concedido a T1 pode ser revisto por meio do *rollback* de T1 concedido a T2.
 - Para controlar esse sistema, considera-se um único *timestamp* para cada transação. Eles são usados para decidir se a transação esperará pelo bloqueio ou será desfeita.
 - Se uma transação for desfeita, ela manterá seu *timestamp* original quando for reiniciada.
- Duas técnicas de prevenção, que utilizam o conceito de registro de *timestamp*, são:
 - esperar-morrer (*wait-die*)
 - ferir-esperar (*wound-wait*)

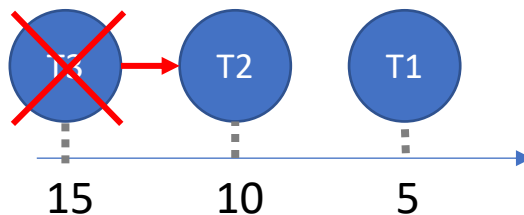


Tratamento de impasse – DEADLOCK

Protocolo de prevenção – esperar-morrer

Considere a seguinte situação:

- Em um determinado escalonamento, a transação T_i tenta bloquear um item X mas não consegue porque X está bloqueado pela transação T_j com um bloqueio em conflito.
- **Esperar-morrer (*wait-die*)**
 - Se $TS(T_i) < TS(T_j)$, ou seja, T_i mais antiga do que T_j , então T_i é autorizada a esperar
 - Uma transação mais antiga é autorizada a esperar por uma transação mais nova
 - Se $TS(T_i) > TS(T_j)$, ou seja, T_i mais nova do que T_j , então T_i é abortada e reiniciada posteriormente com o mesmo valor de registro de *timestamp*
 - Uma transação mais nova, que requeira um item bloqueado por uma transação mais antiga, é abortada e reiniciada

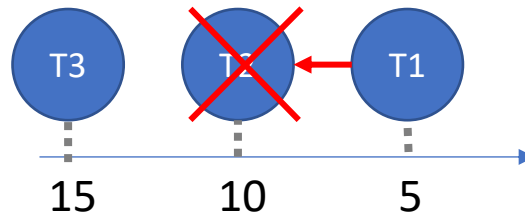


Tratamento de impasse – DEADLOCK

Protocolo de prevenção – ferir-esperar

Considere a seguinte situação:

- Em um determinado escalonamento, a transação T_i tenta bloquear um item X mas não consegue porque X está bloqueado pela transação T_j com um bloqueio em conflito.
- **ferir-esperar (*wound-wait*)**
 - Se $TS(T_i) < TS(T_j)$, ou seja, T_i mais antiga do que T_j , então T_j é abortada (T_i fere T_j) e reiniciada posteriormente com o mesmo valor de registro de *timestamp*
 - Uma transação mais antiga, que requeira um item bloqueado por uma transação mais nova, apropria-se da transação mais nova, abortando a mesma



- Se $TS(T_i) > TS(T_j)$, ou seja, T_i mais nova do que T_j , então T_i é autorizada a esperar
 - Uma transação mais nova é autorizada a esperar pela mais antiga

Tratamento de impasse – DEADLOCK

Protocolo de prevenção

- esperar-morrer (*wait-die*) / ferir-esperar (*wound-wait*)
- Vantagens
 - As duas técnicas são livres de *deadlocks*
 - Acabam abortando a transação mais nova (que começou mais tarde), que poderia estar envolvida em *deadlocks*
- Desvantagens
 - As duas técnicas podem fazer com que transações sejam abortadas e reiniciadas sem necessidade, já que não necessariamente geram um *deadlock*



Tratamento de impasse – DEADLOCK

Protocolo de prevenção

- **Abordagem 3**

- **Sem espera (*no waiting*)**

- Se a transação for incapaz de obter o bloqueio, ela é imediatamente abortada e, posteriormente reiniciada
- Não há transações em espera
- Transações abortam e reiniciam sem necessidade

- **Espera cuidadosa (*cautious waiting*)**

- Considere a seguinte situação:
 - Em um determinado escalonamento, a transação T_i tenta bloquear um item X mas não consegue porque X está bloqueado pela transação T_j com um bloqueio em conflito.
- Se T_j não estiver bloqueada (não esperando por outro item bloqueado), então T_i está bloqueada e tem permissão para esperar; caso contrário, aborte T_i .
- É livre de *deadlock*, pois nenhuma transação esperará por outra transação bloqueada.

Controle de concorrência

Detecção e recuperação de *deadlocks*

- Solução utilizada quando há **pouca ocorrência de *deadlocks***
- *Timeout* (tempo-limite)
 - Se uma transação esperar por um período maior que o período de *timeout* definido pelo sistema, o sistema pressupõe que a transação pode entrar em *deadlock* e a aborta — independentemente de um *deadlock* realmente existir ou não.
- Grafo de espera
 - Um mecanismo é evocado periodicamente para examinar o estado do sistema e determinar se um *deadlock* está ocorrendo.
 - Se há um *deadlock*, seleciona-se uma **transação vítima** por meio de um ou mais critérios (com custo mínimo)



Tratamento de impasse

Detecção de *deadlock* – grafo de espera

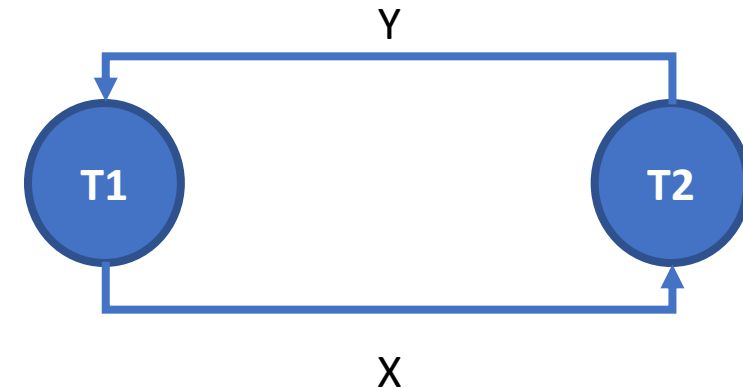
- Uma forma de se detectar um estado de deadlock é construir e manter um **grafo de espera**
 - Um nó é criado no grafo de espera para cada transação que esteja sendo executada no momento
 - Uma aresta direcionada (nó $T_i \rightarrow$ nó T_j) é criada no grafo de espera sempre que uma transação T_i estiver esperando para bloquear um item que esteja bloqueado por uma transação T_j .
 - Quando T_j libera o bloqueio nos itens que T_i está esperando, a aresta direcionada é retirada do grafo de espera.
- Há um *deadlock* se, e somente se, o grafo de espera tiver um ciclo



Tratamento de impasse

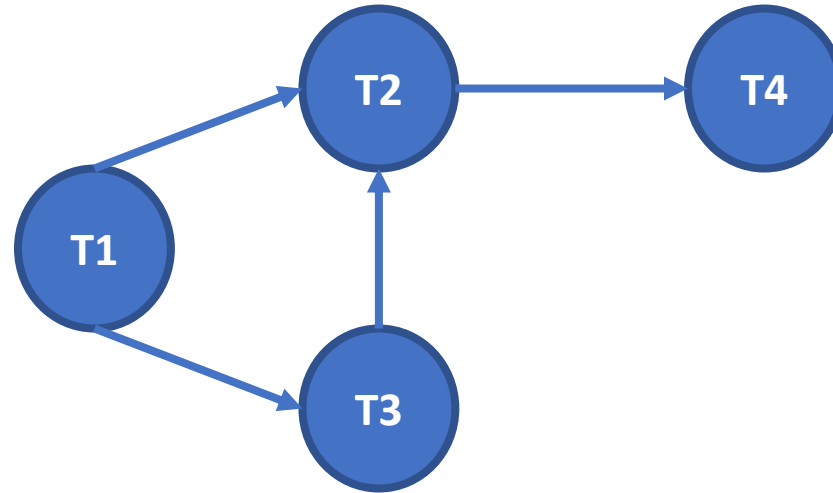
Detecção de deadlock – grafo de espera

T1	T2	Controle de concorrência
<code>read_lock(Y);</code> <code>read_item(Y);</code> <code>write_lock(X);</code> <code>unlock(Y);</code> <code>read_item(X);</code> <code>X=X+Y;</code> <code>write_item(X);</code> <code>unlock(X);</code>	<code>read_lock(X);</code> <code>read_item(X);</code> <code>write_lock(Y);</code> <code>unlock(X);</code> <code>read_item(Y);</code> <code>Y=X+Y;</code> <code>write_item(Y);</code> <code>unlock(Y);</code>	<code>grant-read_lock(Y, T1);</code> <code>grant-read_lock(X, T2);</code> <code>wait(Y, T1);</code> <code>wait(X, T2);</code> <i>Deadlock!</i>



Tratamento de impasse

Detecção de *deadlock* – grafo de espera

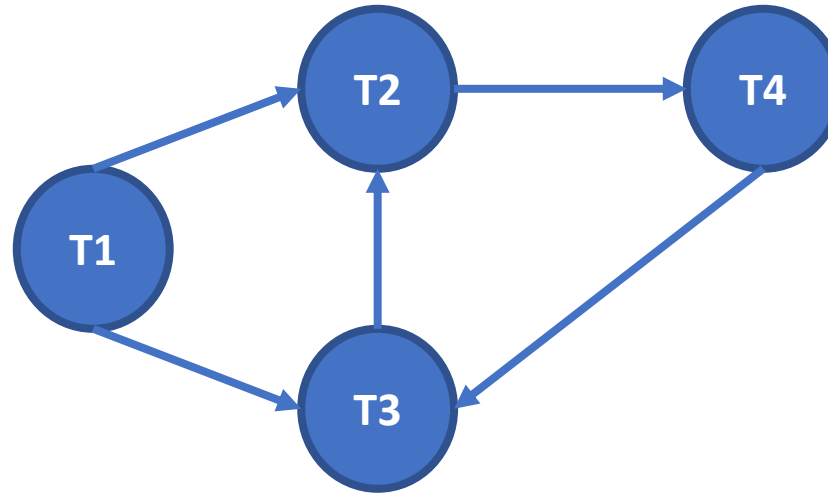


A transação T1 está esperando as transações T2 e T3.
A transação T3 está esperando a transação T2.
A transação T2 está esperando a transação T4.



Tratamento de impasse

Detecção de *deadlock* – grafo de espera



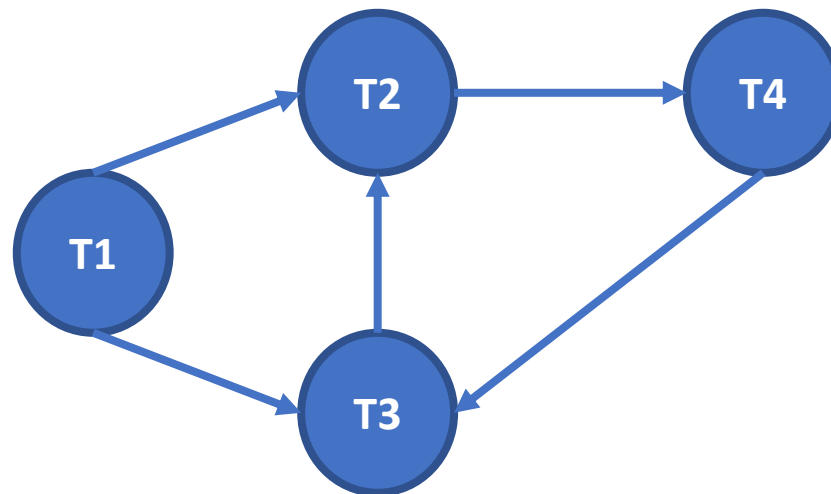
A transação T4 está solicitando um item bloqueado por T3.

Há um ciclo no grafo! O sistema está em *deadlock*



Tratamento de impasse

Detecção de *deadlock* – grafo de espera



A transação T4 está solicitando um item bloqueado por T3.

Há um ciclo no grafo! O sistema está em *deadlock*

É preciso selecionar uma transação vítima quer será abortada.



Tratamento de impasse

Detecção de *deadlock* – inanição

- A inanição (*starvation*) pode ocorrer quando:
 - O esquema de espera para itens bloqueados for injusto, priorizando algumas transações em relação a outras
 - Solução 1: quanto mais uma transação espera, aumentar a sua prioridade de execução
 - Solução 2: utilizar um esquema de fila de tal forma que a primeira transação a chegar será a primeira a ser atendida



Controle de concorrência

- O controle de concorrência também precisa assegurar que os esquemas de execução concorrentes permitam a execução de procedimentos de **recuperação de falhas**.
- Para isso, os esquemas resultantes das execuções concorrentes devem ser:
 - **RECUPERÁVEIS**
 - Uma transação que depende de outra (usa um dado já alterado pela outra naquele esquema de execução concorrentemente) não pode ser efetivada antes que a outra seja.
 - **ESCALAS SEM CASCATA**
 - Uma transação usa um dado alterado por outra (naquele esquema de execução concorrente) apenas se a outra já foi efetivada.

