Universidade Eduardo Mondlane
Faculdade de Engenharia
Departamento de Engenharia Electrotécnica
Curso de Engenharia Informática

Base de Dados II

Msc Sérgio Mavie
Eng. Cristiliano Maculuve

Clementina Elihud

Agenda

- ❑ **Serialização**
- ❑ **Bloqueios**

Transações

Sejam T_1 e T_2 duas transações que transferem fundos de uma conta para outra.

A transação T_1 transfere 50 dólares da conta A para a conta B.

A transação T_2 transfere 10 por cento do saldo da conta A para a conta B.

T_1:

```
read(A);
A := A – 50;
write(A);
read(B);
B := B + 50;
write(B);
```

T_2:

```
read(A);
temp := A * 0,1;
A := A – temp;
write(A);
read(B);
B := B + temp;
write(B);
```

Execução Sequencial- sempre preserva a consitencia da bases de dados

T_1	T_2

read(A);
A := A – 50;
write(A);
read(B);
B := B + 50;
write(B);	

	read(A);
	temp := A * 0,1;
	A := A – temp;
	write(A);
	read(B);
	B := B + temp;
	write(B);

T_1	T_2

	read(A);
	temp := A * 0,1;
	A := A – temp;
	write(A);
	read(B);
	B := B + temp;
	write(B);

read(A);
A := A – 50;
write(A);
read(B);
B := B + 50;
write(B);	

Podemos dizer então que temos:

- Premissa
  - A execução de uma transação é correta, se executada isoladamente. Pois, produz sempre um estado consistente.
- Teorema
  - Toda execução serial de transações é correta

$$S=T_1\ T_2\ T_3 \ldots\ T_{n-1}\ T_n$$

Este é o padrão para corretude de schedules!!!

Problemas da Execução Sequencial:

Solução ineficiente !!!

- várias transações podem esperar muito tempo para serem executadas.
- CPU pode ficar muito tempo ociosa.
  - enquanto uma transação faz I/O, por exemplo, outras transações poderiam ser executadas.

Uma planificacão **P** corresponde à ordem de execução das operações de várias transações executadas de forma concorrente.

• Duas operações estão em **conflito**, se elas pertencem a diferentes transações, acessam o mesmo item de dado e se uma das operações é de gravação.

É POSSIVEL ESCALONAR AS OPERACÕES DE TRANSACÕES CONCORRENTES DE MODO A QUE ESTAS OCORRAM COMO SE TIVESSEM SIDO EXECUTADAS EM SÉRIE!

A isto chamamos Serializacão. Ou Planificacão Série. (Schedule)

Exemplos:

Correta

T_1	T_2
read(A);	
A := A – 50;	
write(A);	
	read(A);
	temp := A * 0,1;
	A := A – temp;
	write(A);
read(B);	
B := B + 50;	
write(B);	
	read(B);
	B := B + temp;
	write(B);

Incorreta

T_1	T_2
read(A);	
A := A – 50;	
	read(A);
	temp := A * 0,1;
	A := A – temp;
	write(A);
write(A);	
read(B);	
B := B + 50;	
write(B);	
	read(B);
	B := B + temp;
	write(B);

Planificação serializável

- Uma Planificação **P** (não série) é serializável se for **equivalente a alguma execusão série das mesmas n transacções**
  - Uma planificação que não é equivalente a **nenhuma** execução em série, é uma planificação **não serializável**
- Toda planificação serializável é **correcta**
  - Produz os mesmos resultados que alguma execução em série
- Duas maneiras de definir a equivalência entre as planificações:
  - **Equivalência por conflictos**
  - **Equivalência por visão**

- **Serializabilidade por conflito**
  - **S** é serializavel por conflito, se **S** é equivalente por conflito a algum *schedule* serial $\mathbf{S_S}$ sobre o mesmo conjunto de transações
  - Exemplo

    T_1=$\mathbf{r_1(y)r_1(x)w_1(y)}$ T_2=$\mathbf{r_2(x)r_2(y)w_2(x)}$

    T_3=$\mathbf{r_3(y)r_3(x)w_3(z)}$

S = $\mathbf{r_3(y)r_1(y)r_1(x)r_2(x)w_1(y)r_2(y)r_3(x)w_2(x)w_3(z)}$ → Schedule original

S_S=$\mathbf{r_3(y)r_3(x)w_3(z)}$$\mathbf{r_1(y)r_1(x)w_1(y)}$$\mathbf{r_2(x)r_2(y)w_2(x)}$ → Schedule serial (T_3,T_1,T_2)

Operações em conflito do Schedule S	Operações em conflito do Schedule S_S
$r_3(y) <_S w_1(y)$	$r_3(y) <_S w_1(y)$
$r_1(x) <_S w_2(x)$	$r_3(x) <_S w_2(x)$
$w_1(y) <_S r_2(y)$	$r_1(x) <_S w_2(x)$
$r_3(x) <_S w_2(x)$	$w_1(y) <_S r_2(y)$

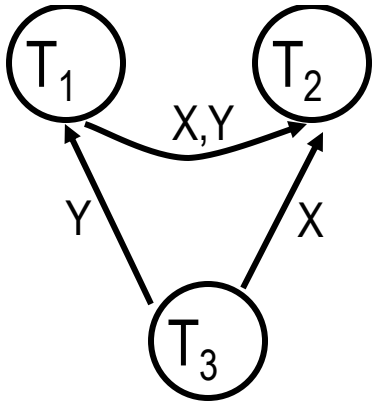
Como verificamos através da forma como os dois schedules ordenam suas operações em conflito, podemos dizer que o **Schedule S** é **serializável por conflito**. Esta é uma das formas de verificação da serializabilidade por conflito de um schedule.

Grafo de serialização de um schedule S

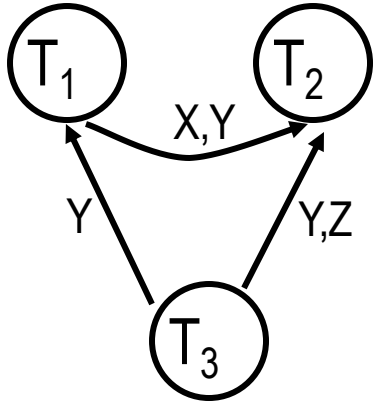
- S está definido para um conjunto $\Im=\{T_1, T_2, T_3\}$

$S=r_3(y)r_1(y)r_1(x)r_2(x)w_1(y)r_2(y)r_3(x)w_2(x)w_3(z)$

(Transações):	
Arestas (operações em conflito):	$r_3(y) < w_1(y)$ → aresta de 3 para 1 $r_1(x) < w_2(x)$ → aresta de 1 para 2 $w_1(y) < r_2(y)$ → aresta de 1 para 2 $r_3(x) < w_2(x)$ → aresta de 3 para 2



Planificacao serie equivalente:
T3->T1->T2



T1	T2	T3
		R(Y);
		R(Z);
R(X);		
W(X);		
		W(Y);
		W(Z);
	R(Z);	
R(Y);		
W(Y);		
	R(Y);	
	W(Y);	
	R(X);	
	W(X);	

Planificación F

Locking/Bloqueios

Locks são mecanismos que previnem conflitos entre transações que acedam o mesmo recurso.

Tipos de Lock:

exclusive:

Previne que recursos sejam compartilhados.
A primeira transação que realiza lock exclusivo em um recurso, é a única transação que pode alterar o recurso até que o lock exclusivo seja liberado.
MySQL → LOCK TABLE *nome da tabela* {READ | WRITE}
UNLOCK TABLES

shared lock:

Permite que recursos sejam compartilhados. Vários usuário podem ler os dados, realizando "shared lock" para prevenir acesso concorrente de escrita(necessita de um lock exclusive)
Várias transações podem adquirir um "shared lock" em um recurso.

- Exemplo: suponha as transações T_1 e T_2
  - T_1: Read (Aplic);
    Aplic.Saldo = Aplic.Saldo – 500;
    Write (Aplic);
    Read (Conta);
    Conta.Saldo = Conta.Saldo + 500;
    Write (Conta);
  - T_2: Read (Conta);
    Read (Aplic);
    Print (Conta.Saldo + Aplic.Saldo);
  - Operações de bloqueio devem ser adicionadas ao código para garantir o isolamento entre as transações

- Bloqueio só no acesso não garante isolamento:

T_1	T_2
Lock-X (Aplic); Read (Aplic); Aplic.Saldo = Aplic.Saldo – 500; Write (Aplic); Unlock (Aplic);	
	Lock-S (Conta); Read (Conta); Unlock (Conta);
Lock-X (Conta); Read (Conta); Conta.Saldo = Conta.Saldo + 500; Write (Conta); Unlock (Conta);	
	Lock-S (Aplic); Read (Aplic); Unlock (Aplic); Print (Conta.Saldo + Aplic.Saldo);

Bloqueio pode causar inanição (*starvation*)

T_1	T_2	T_3	T_4	T_5
lock-S(Q)				
	lock-X(Q)			
	bloqueada	lock-S(Q)		
	bloqueada		lock-S(Q)	
	bloqueada			lock-S(Q)

- T_2 nunca recebe o direiro de acesso!
- Pode ser evitado fazendo que o direito de acesso compartilhado não seja concedido se houver uma transação esperando por bloqueio exclusivo

Uso de Bloqueios “S” e “X”

- Não garantem escalonamentos serializáveis
- Exemplo

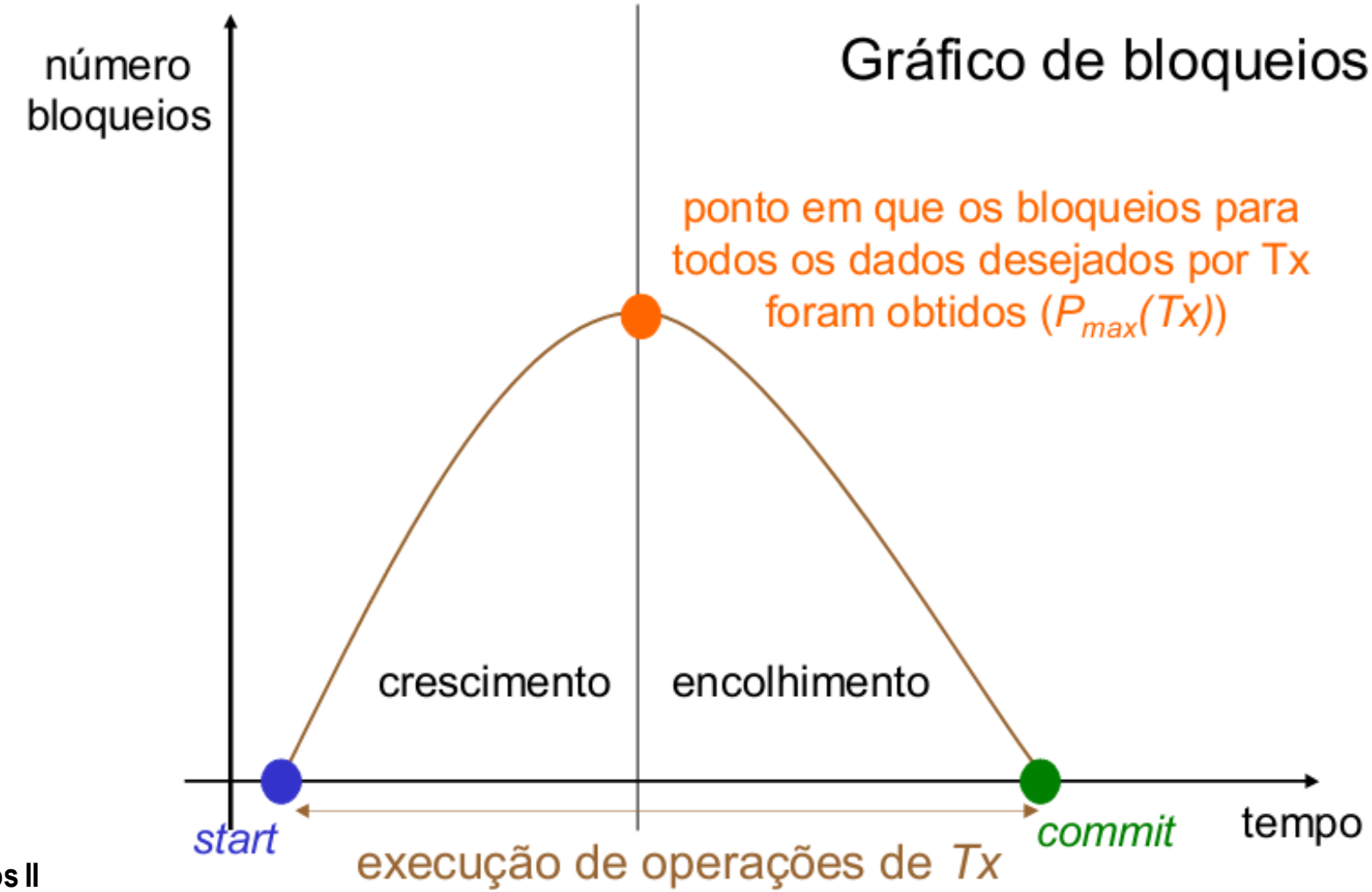
$H_{N\text{-}SR}$= ls1(Y) r1(Y) u1(Y) ls2(X) r2(X) u2(X) lx2(Y) r2(Y)
w2(Y) u2(Y) c2 lx1(X) r1(X) w1(X) u1(X) c1

- Necessita-se de uma técnica mais rigorosa de bloqueio para garantir a serializabilidade
  - técnica mais utilizada
    - bloqueio de duas fases (*two-phase locking* – 2PL)

Bloqueio de 2 Fases – 2PL

- Premissa
  - *"para toda transação Tx, todas as operações de bloqueio de dados feitas por Tx precedem a primeira operação de desbloqueio feita por Tx"*
- Protocolo de duas fases
  1. Fase de expansão ou crescimento
     - *Tx* pode obter bloqueios, mas não pode liberar nenhum bloqueio
  2. Fase de retrocesso ou encolhimento
     - *Tx* pode liberar bloqueios, mas não pode obter nenhum bloqueio

Scheduler 2PL – Funcionamento



- Bloqueio em Duas Fases – Exemplo:

T_1	T_2

Lock-X (Aplic);
Read (Aplic);
Aplic.Saldo = Aplic.Saldo – 500;
Write (Aplic);
Lock-X (Conta);
Unlock (Aplic); // Inicia 2ª fase
Read (Conta);	

	Lock-S (Conta);
Conta.Saldo = Conta.Saldo + 500; Write (Conta); Unlock (Conta);	*Bloqueada*

	Read (Conta);
	Lock-S (Aplic);
	Unlock (Conta); // Inicia 2ª fase
	Read (Aplic);
	Print (Conta.Saldo + Aplic.Saldo);
	Unlock (Aplic);

Scheduler 2PL - Crítica

Vantagem

- técnica que sempre garante escalonamentos SR sem a necessidade de se construir um grafo de dependência para teste!
  - se *Tx* alcança P_{max}, *Tx* não sofre interferência de outra transação *Ty*, pois se *Ty* deseja um dado de *Tx* em uma operação que poderia gerar conflito com *Tx*, *Ty* tem que esperar (evita ciclo *Ty* ? *Tx*!)
  - depois que *Tx* liberar os seus dados, não precisará mais deles, ou seja, *Tx* não interferirá nas operações feitas futuramente nestes dados por *Ty* (evita também ciclo *Ty* ? *Tx*!)

Scheduler 2PL - Crítica

Desvantagens

- limita a concorrência
  - um dado pode permanecer bloqueado por *Tx* muito tempo até que *Tx* adquira bloqueios em todos os outros dados que deseja
- 2PL básico (técnica apresentada anteriormente) não garante escalonamentos
  - livres de *deadlock*
    - *Tx* espera pela liberação de um dado bloqueado por *Ty* de forma conflitante e vice-versa
  - adequados à recuperação pelo *recovery*

- Bloqueio em duas fases pode causar *deadlock*:

T_1	T_2
Lock-X (Aplic); Read (Aplic); Aplic.Saldo = Aplic.Saldo – 500;	
	Lock-S (Conta); Read (Conta); Lock-S (Aplic);
Write (Aplic); Lock-X (Conta);	*Bloqueada*
Bloqueada	*Bloqueada*
Não executa: Unlock (Aplic); Read (Conta); Conta.Saldo = Conta.Saldo + 500; Write (Conta); Unlock (Conta);	**Não executa:** Unlock (Conta); Read (Aplic); Print (Conta.Saldo + Aplic.Saldo); Unlock (Aplic);

- Bloq. em 2 fases não evita *rollback* em cascata

T_1	T_2
Lock-X (Aplic); Read (Aplic); Aplic.Saldo = Aplic.Saldo – 500; Write (Aplic); Lock-X (Conta); Unlock (Aplic); Read (Conta);	
	Lock-S (Aplic); Read (Aplic); Lock-S (Conta);

Conta.Saldo = Conta.Saldo + 500;
Write (Conta); // ABORTA
Unlock(Conta);
	Bloqueada

	Read (Conta); // ABORTA
	Unlock (Conta);
	Print (Conta.Saldo + Aplic.Saldo);
	Unlock (Aplic);

Existem:

Variantes do Bloqueio em Duas Fases

- Evitam o *rollback* em cascata
- Usados pela maioria dos SGBDs
- Protocolo de Bloqueio em Duas Fases Severo
  - Obriga que os bloqueios exclusivos sejam mantidos até a efetivação da transação
- Protocolo de Bloqueio em Duas Fases Rigoroso
  - Obriga que todos os bloqueios (compartilhados e exclusivos) sejam mantidos até o *commit*

- Bloqueio em Duas Fases Severo – Exemplo:

T_1	T_2
Lock-X (Aplic); Read (Aplic); Aplic.Saldo = Aplic.Saldo – 500; Write (Aplic); Lock-X (Conta); Read (Conta);	
	Lock-S (Aplic);
Conta.Saldo = Conta.Saldo + 500; Write (Conta); **Unlock (Aplic);** Unlock(Conta);	*Bloqueada*
	Read (Aplic); Lock-S (Conta); **Unlock (Aplic);** Read (Conta); Print (Conta.Saldo + Aplic.Saldo); Unlock (Conta);

- Bloqueio em Duas Fases Rigoroso – Exemplo:

T_1	T_2
Lock-X (Aplic); Read (Aplic); Aplic.Saldo = Aplic.Saldo – 500; Write (Aplic); Lock-X (Conta); Read (Conta);	
	Lock-S (Aplic);
Conta.Saldo = Conta.Saldo + 500; Write (Conta); **Unlock (Aplic);** Unlock(Conta);	*Bloqueada*
	Read (Aplic); Lock-S (Conta); Read (Conta); Print (Conta.Saldo + Aplic.Saldo); **Unlock (Aplic);** Unlock (Conta);

TPC

- Ler e discutir sobre serialização por visão

Referências

1. ELMASRI, R.; NAVATHE, S. B. , *Fundamentals of Database Systems*, Addison-Wesley Publishing; 2000, ISBN: 013057591
2. DATE, C. J. , *An Introduction to Database Systems*, Addison-Wesley Pub Co; 6th edition, 2000, ASIN: 020154329X
3. PEREIRA, J. L. , *Tecnologias de Base de Dados*, FCA, 3 edição, ISBN: 972-722-143-2
4. SILBERSCHATZ, A., KORTH, H. F. , SUDARSHAN, S.. *Sistemas de Bancos de Dados*. Campus, 1999.

OBRIGADO!