

# Escalonamento

Elmasri – Capítulos 21 e 22

Ramakrishnan – Capítulos 16 e 17

Silberchatz – Capítulos 15 e 16

Complete Book – Chapter 18



# Introdução

- Controle de concorrência assegura o *isolamento* das transações
- Garantem a serialização dos escalonamentos das transações
  - Uso de protocolos (conjunto de regras)

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

# Escalonamento

- Um escalonador é uma sequência de operações realizadas por uma ou mais transações ordenadas em relação ao tempo

**Definição:** escalonador é dito serial se **não** existe intercalação de operações

# Escalonamento serial

Considere as seguintes transações e as possíveis execuções **sequenciais** :

$T_0$ : Read (A)

A: A-100

Write (A)

Read (B)

B: B + 100

Write (B)

$T_1$ : Read (A)

x: A \* 0.10

A: A - x

Write (A)

Read (B)

B: B + x

Write (B)

Valores	A	B
Iniciais	1000	2000
$T_0 \rightarrow T_1$		
$T_1 \rightarrow T_0$		

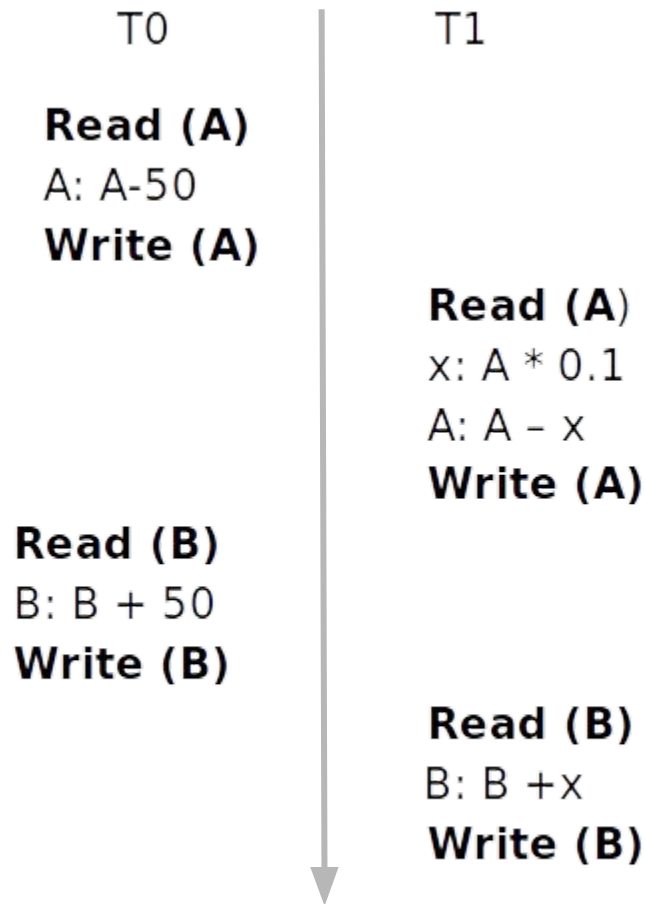
# Escalonamento não-serial



**O que é:** Um escalonamento com operações concorrentes com o mesmo efeito de transações seriais.

Um resultado correto é obtido por escalonadores concorrentes sempre que o resultado obtido seja **igual** ao produzido por um escalonador serial.

# Escalonamento não-serial



Valores	A	B
Iniciais	1000	2000
T0 -> T1		

# Escalonamento não-serial?

**T<sub>0</sub>**

**Read (A)**

A: A-50

**Write (A)**

**Read (B)**

B: B + 50

Write (B)

**T<sub>1</sub>**

**Read (A)**

x: A \* 0.1

A: A - x

**Write (A)**

**Read (B)**

B: B + x

**Write (B)**

Valores

A

B

Iniciais 1000

2000

T1 -> T0

# Exemplo

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

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



# Escalonamento com conflito de serialibidade

Diz-se que duas operações são **conflitantes** se elas operam sobre o mesmo item de dados, sendo que no mínimo uma delas é uma gravação, e são emitidas por **diferentes transações**.

Usaremos a seguinte notação :

- $R_i(x)$  - para operação de leitura do item  $x$  realizada pela transação  $i$ .
- $W_i(x)$  - para operação de gravação do item  $x$  realizada pela transação  $i$ .

# Escalonamento com conflito de serialidade

Diz-se que duas operações são **conflitantes** se elas operam sobre o mesmo item de dados, sendo que no mínimo uma delas é uma gravação, e são emitidas por **diferentes transações**.

E1 : R1(x), R2(x), W2(x), W1(x)

T1	T2
R(X)	
	R(X)
	W(X)
W(X)	

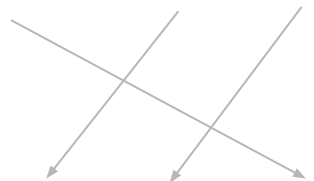
# Escalonamento com conflito de serialibidade

Se **A** precede **B** e são instruções diferentes e *não conflitantes* então pode-se trocar a ordem entre elas gerando-se assim um novo escalonador que difere apenas na ordem destas operações.

Exemplo :

E1 : R1(x), R2(x), W2(y), W1(x)

E2 : R2(x), W2(y), R1(x), W1(x)



# Escalonamento com conflito de serialibidade

Se **A** precede **B** e são instruções diferentes e *não conflitantes* então pode-se trocar a ordem entre elas gerando-se assim um novo escalonador que difere apenas na ordem destas operações.

Exemplo :

E1 : **R1**(x), **R2**(x), W2(y), **W1**(x) (não-serial)

E2 : **R2**(x), W2(y), R1(x), **W1**(x) (serial)

## Escalonadores serializáveis em conflito

“dado um escalonamento não-serial E1 para um conjunto de Transações T, **E1** é **serializável** se a ordem de quaisquer 2 operações em conflito é a mesma em E1 e em algum escalonamento **serial** E.”

# Escalonamento não-serial serializável

**S- serial**

T1	T2
read(X)	
$X = X - 20$	
<b>write(X)</b>	
read(Y)	
$Y = Y + 20$	
write(Y)	
	<b>read(X)</b>
	$X = X + 10$
	<b>write(X)</b>

**S' - não serial**

T1	T2
read(X)	
$X = X - 20$	
<b>write(X)</b>	
	<b>read(X)</b>
	$X = X + 10$
	write(X)
read(Y)	
$Y = Y + 20$	
write(Y)	

# Exemplo

escalonamento serial  $E$

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

escalonamento não-serial  $E1$     escalonamento não-serial  $E2$

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

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

# Atividade - Os escalonadores são serializáveis?

Serial

T1	T2
	read(X)
	read(Y)
read(X)	
read(y)	
read(z)	
read(O)	
O = O + 20	
write(O)	

Não-serial

T1	T2
read(X)	
	read(X)
read(y)	
	read(Y)
read(z)	
read(O)	
O = O + 20	
write(O)	

# Protocolos Baseados em Bloqueio

- Idéia Básica

Quando uma transação acessa um item de dados deve antes bloqueá-lo, caso este já esteja bloqueado por outra transação deve esperar até que o item seja liberado

- Modos de Bloqueio

- Compartilhado **LS**
- Exclusivo **LX**





# Protocolos Baseados em Bloqueio

## Compartilhado - LS

Quando o item desejado não está bloqueado por nenhuma transação ou está bloqueado em modo compartilhado.

## Exclusivo - LX

Somente quando o item desejado não está bloqueado

# Transações Bem Formadas

- Aquelas que sempre bloqueiam o item de dados em modo compartilhado antes de lê-lo e sempre bloqueiam em modo exclusivo antes de gravá-lo
- Duas transações estão em conflito se elas desejam bloquear o mesmo item de dados em modos incompatíveis.

# Exemplo 1

T1: LX1 (B)  
R1 (B)  
B: B-50  
W1 (B)  
UL1 (B)  
LX1 (A)  
R1 (A)  
A: A + 50  
**W1 (A)**  
UL1 (A)

T2: LS2 (A)  
**R2 (A)**  
UL2 (A)  
LS2 (B)  
R2 (B)  
UL2 (B)  
Display (A + B)

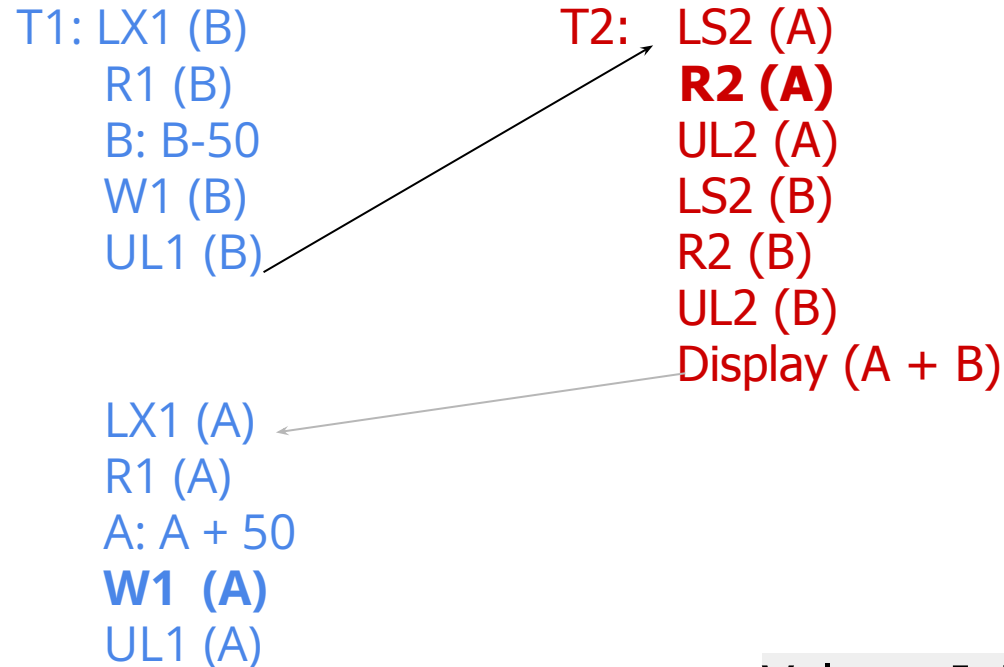
Seja E1:

Valores Iniciais: A=100 B=200

LX1 (B) R1(B) W1 (B) UL1 (B) LX1 (A) R1 (A) W1 (A) UL1 (A) LS2 (A) R2 (A)  
UL2 (A) LS2 (B) R2 (B) UL2 (B) Display (A + B)

**O valor da linha "Display (A+B)" está correto?**

# Exemplo 2



Valores Iniciais: A=100 B=200


Seja E1:

LX1 (B) R1(B) W1 (B) UL1 (B) LS2 (A) R2 (A) UL2 (A) LS2 (B) R2 (B) UL2 (B)  
Display (A + B) LX1 (A) R1 (A) W1 (A) UL1 (A)

**O valor da linha "Display (A+B)" está correto?**

# Exemplo 2

Valores Iniciais: A=100 B=200



LX1 (B)  
R1 (B)  
B: B-50  
W1 (B)  
UL1 (B)  
LS2 (A)  
R2 (A)  
UL2 (A)  
LS2 (B)  
R2 (B)  
UL2 (B)  
Display (A + B)  
LX1 (A)  
R1 (A)  
A: A + 50  
W1 (A)  
UL1 (A)

# Protocolo de Bloqueio Bifásico (2PL)

A execução concorrente de transações é correta se observada as seguintes regras:

1. Transações bem formadas
2. Regras de compatibilidade de bloqueio são obedecidas
3. Cada transação após liberar um bloqueio não solicita um novo bloqueio

A condição 3 pode ser expressa dizendo que as transações são *Bifasicamente Bloqueadas*

# Protocolo de Bloqueio Bifásico (2PL)

Todas as transações devem obedecer as seguintes fases:

## Primeira Fase – *Crescimento*

Durante a qual obtém seus bloqueios, mas não libera bloqueio algum.

## Segunda Fase – *Retração*

Na qual os bloqueios são liberados mas nenhum bloqueio pode ser requerido

# Exemplo

T1	T2	A
LX(A)		25
Read(A)		
A=A+100	LX(A)	125
W(A)	LX(A)	125
UL(A)	LX(A)	
	LX(A)	
	read(A)	125
	A=A*2	250
	write(A)	250
	UL(A)	



# 2PL

- Quais transações obedecem 2PL?

$T_1$	$T_2$	$T_3$	$T_4$
LS(Y) Read(Y) UL(Y) LX(X) Read(X) $X := X + Y$ Write(X) UL(X)	LS(X) Read(X) UL(X) LX(Y) Read(Y) $Y := X + Y$ Write(Y) UL(Y)	LS(Y) Read(Y) LX(X) UL(Y) Read(X) $X := X + Y$ Write(X) UL(X)	LS(X) LX(Y) Read(X) Read(Y) $Y := X + Y$ Write(Y) UL(X) UL(Y)

# Atividade A

Os escalonadores abaixo seguem o 2PL?

<b>A)</b>	LS1(A)	<b>B)</b>	LS1(A)	<b>C)</b>	LS1(A)
	R1(A)		R1(A)		R1(A)
	LS2(A)		LX2(A)		LX2(A)
	LX1(B)		LX1(B)		UL1(A)
	UL1(A)		R1(B)		LX1(B)
	R1(B)		W1(B)		R1(B)
	W1(B)		UL1(A)		W1(B)
	R2(A)		UL1(B)		UL1(B)
	UL2(A)		R2(A)		R2(A)
	UL1(B)		W2(A)		W2(A)
			UL2(A)		UL2(B)

# Atividade B

S1: r1(A), r2(D), w1(A), r2(C), r2(B), w2(B), w1(C)

1- O escalonador S1 respeita o protocolo 2PL usando somente bloqueio exclusivos?

2- Com bloqueios exclusivos e compartilhados, respeita o 2PL?

# Resolução B-1

S1: r1(A), r2(D), w1(A), r2(C), r2(B), w2(B), w1(C)

<b>1</b>	<b>2</b>
----------	----------

# Resolução B

S1: r1(A), r2(D), r3(B), w1(A), r2(C), r2(B), w2(B), w1(C)

1	2
---	---

# Atividade C

S: r<sup>2</sup>(A), r<sup>3</sup>(B), w<sup>1</sup>(A), r<sup>2</sup>(C), r<sup>2</sup>(D), w<sup>1</sup>(D),

O escalonador S respeita o **protocolo 2PL** usando bloqueio **exclusivo**?

t1	t2	t3
	LX(A)	

t1	t2	t3

t1	t2	t3

# Seriabilidade e Isolamento

- O 2PL garante a **seriabilidade** de transações
- A propriedade de **isolamento** só é alcançada caso todos os bloqueios exclusivos sejam mantidos até a confirmação (commit).

**OBS:** A vulnerabilidade do 2PL a impasses continua

# Impasse (Deadlock)



Considere que a transação  $T_i$  tenta bloquear  $X$ , mas  $X$  já está bloqueado por  $T_j$

**Esperar-morrer**: transações mais antigas esperam, as mais novas são abortadas

Exemplo:  $T_5$ , e  $T_{50}$ ,

Se  $T_5$  precisa de um dado bloqueado por  $T_{50}$ ,  $T_5$  então espera;

Se  $T_{50}$  precisa de um dado bloqueado por  $T_5$ ,  $T_{50}$  então é abortada;



# Impasse (Deadlock)



Considere que a transação  $T_i$  tenta bloquear  $X$ , mas  $X$  já está bloqueado por  $T_j$

**Ferir-esperar:** transações mais novas esperam pelas antigas e as mais antigas abortam as mais novas (voltam com o mesmo TS)

Exemplo:  $T_5$ , e  $T_{50}$ ,

Se  $T_5$  precisa de um dado bloqueado por  $T_{50}$ ,  $T_{50}$  é abortado;

Se  $T_{50}$  precisa de um dado bloqueado por  $T_5$ ,  $T_{50}$  então espera;

# Atividade

1- Crie uma situação de deadlock no postgres.

1- Crie uma tabela com chave primária;

2- Crie duas transações em dois terminais;

3- Faça um update no terminal 1 na tupla X;

4- Faça um update no terminal 2 na tupla Y;

5- Faça um update no terminal 1 na tupla Y;

6- Faça um update no terminal 2 na tupla X;

2- Descreva o que aconteceu. Como o Postges controla as alterações em transações? Qual das duas políticas foi aplicada?

# 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)

# Simulação

<https://github.com/amughrabi/cc.git>

# Trigger

```
CREATE OR REPLACE function atualiza_score()
```

```
RETURNS trigger AS $$
```

```
DECLARE
```

```
    idDirector int; i record;
```

```
BEGIN
```

```
for i in SELECT d.director_id  from directors as d
```

```
    join content_directors as cd on d.director_id = cd.director_id
```

```
    join contents as c on c.content_id = cd.content_id
```

```
    where c.content_id = NEW.content_id;
```

```
loop
```

```
    idDirector = i.director_id;
```

```
raise notice '% ', idDirector;
```

```
UPDATE imdb_score set imdb_score =
```

```
(SELECT avg(imdb_score)from directors as d
```

```
join content_directors as cd on d.director_id = cd.director_id
```

```
join contents as c on c.content_id = cd.content_id
```

```
where d.director_id = idDirector)
```

```
where director_id = idDirector;
```

```
END loop;
```

```
RETURN NEW;
```

```
END;
```

```
$$ LANGUAGE plpgsql;
```