

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?

T₀

Read (A)

A: A-50

Write (A)

Read (B)

B: B + 50

Write (B)

T₁

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 seriabilidade

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 a 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 seriabilidade

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 seriabilidade

Se **A** precede **B** 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 seriabilidade

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

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

Escalonadores serializáveis em conflito

“Dado um escalonamento não-serial E2 para um conjunto de Transações T, **E2 é serializável** se a ordem de quaisquer 2 operações em conflito é a mesma em E2 e em algum escalonamento **serial** E.”

Escalonamento não-serial serializável

S- serial

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

S' - não serial

T1	T2
read(X)	
$X = X - 20$	
write(X)	
	read(X)
	$X = X + 10$
	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 - O escalonador é serializável?

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)

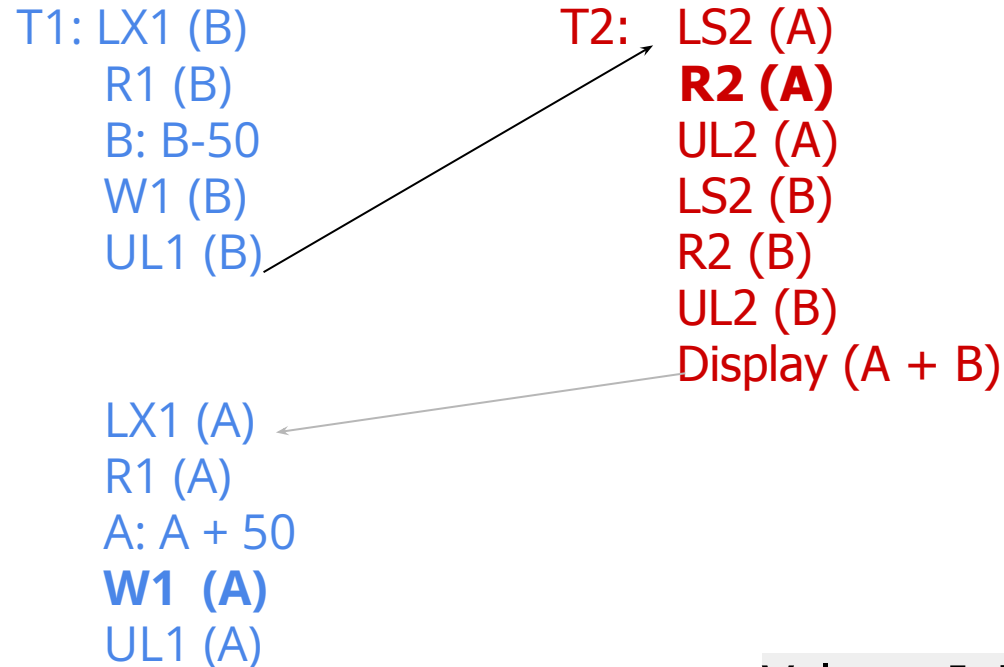
Seja E1:

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?

A)	LS1(A)	B)	LS1(A)	C)	LS1(A)
	R1(A)		R1(A)		R1(A)
	LS2(A)		LX1(B)		UL1(A)
	LX1(B)		R1(B)		LX1(B)
	UL1(A)		W1(B)		R1(B)
	R1(B)		UL1(A)		W1(B)
	W1(B)		UL1(B)		UL1(B)
	R2(A)		R2(A)		R2(A)
	UL2(A)		UL2(A)		UL2(B)
	UL1(B)				

Atividade B1

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?

Atividade B2

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 e compartilhados?

Resolução B-1

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

1	2
----------	----------

Resolução B

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

1	2
---	---

Atividade C

S: r²(A), r³(B), w¹(A), r²(C), r²(D), w¹(D),

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

t1	t2	t3
	LX(A)	

t1	t2	t3

t1	t2	t3

Atividade D

S: r1(A), r2(B), w1(A), r2(C), r2(A), w1(A),

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

t1	t2	

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**, já 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;

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)

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?

Controle de concorrência no Postgres

- Delete não **apaga** de fato o dado
- Update não atualiza o dado de fato



Multiversion Concurrency Control- MVCC

- Baseado no conceito que **conflitos são infrequentes**
- Deixa executar as transações concorrentemente
- Conflito -> Abortar
- Cada transação enxerga sobre uma “cópia” dos dados e não “lê” alterações de outras transações não comitadas

Multiversion Concurrency Control- MVCC

Variáveis:

- **Xmin**: armazena a transação que fez a última alteração
- **Xmax**: reporta se o registro está em processo de remoção
- **Ctid**: mostra o deslocamento e a página que a tupla se encontra;

Multiversion Concurrency Control- MVCC



Exemplo 1

A) Crie a tabela teste (id integer, value char(1000))

B) Adicione 2 linhas com os seguintes valores:

```
Insert into teste values (1,'a'),(2,'b'),(3,'c'),(4,'d');
```

C) Verifique o tamanho da tabela: \dt+ teste

D) Verifique a posição de cada registro com o comando

```
SELECT ctid,* from teste;
```

E) Atualize todas as linhas para o id receber +1

```
Update teste set id=id+1;
```

F) Verifique o tamanho da tabela novamente e a posição de cada registro. **Descreva o que aconteceu.**

Atividade 1

A. Abra um terminal no Postgres:

```
sudo -u postgres psql postgres
```

B. Crie a seguinte tabela:

```
cliente(numero serial primary key, cpf int , nome varchar(50))
```

C. Crie as seguintes operações e descreva o que acontece em cada caso

- a. Uma transação com “commit” (faça 3 inserts)
- b. Uma transação com “rollback” (faça 3 inserts)
- c. Uma transação tentando acessar dados de outra transação ainda não “comitados”. (abra dois terminais com o postgres). O que acontece?

Atividade 2

A) Abra uma transação A:

Rode `SELECT txid_current()`

Rode `SELECT xmin, xmax, ctid, * FROM teste`

B) Apague uma tupla em outra transação (B) (outro terminal)

`SELECT txid_current()`

C) Rode `SELECT xmin, xmax, ctid, * FROM teste`

D) Rode novamente o `SELECT xmin, xmax, ctid, * FROM teste`. **Explique o que aconteceu?**

E) Faça o mesmo com o comando `update`, atividade A-D.. **Explique novamente o que aconteceu?**

Atividade 3

A) Crie a tabela teste (id integer primary key, value char(2000))

B) Adicione 4 tuplas

C) Verifique o tamanho da tabela

D) Insira uma tupla com id =10 em uma transação A (sem comitar)

E) Insira uma tupla com id =10 em uma transação B (sem comitar)

Explique o que aconteceu com as transações já que os dados são operados em snapshots diferentes?

Atividade 4

Qual é a saída da linha 9?

Linha	Terminal 1	Terminal 2
1	create table txn (id int);	
2	begin;	
3	insert into txn values (1);	
4	insert into txn values (2);	
5		insert into txn values (3);
6		insert into txn values (4);
7	insert into txn values (4);	
8	rollback;	
9	select xmin, xmax, ctid, * from txn;	

Extra

Construa uma aplicação em uma linguagem de programação capaz de executar 1000 inserções usando uma **transação** na tabela. A aplicação deve ser capaz também de listar a tabela após as inserções. Trate a exceção no caso de uma inserção de uma chave já existente (**rollback**).

Os dados podem ser gerados usando a ferramenta:
<https://www.mockaroo.com/>

Atividade Extra

Usar o dataset [reddit vm](#) para responder às seguintes perguntas:

- 1- Qual o tempo de execução na inserção de 10000 tuplas com o autocommit True e False? Explique o que aconteceu. OBS: rodar 5 vezes e fazer a média e desvio padrão dos tempos de execução
- 2- Abra dois terminais e execute, ao mesmo tempo, o código da questão anterior com o autocommit False. Além disso, setar o nível de isolamento SERIALIZABLE. Reportem o tempo (5 execuções com o desvio padrão). Explique o que acontece na prática neste caso?

Atividade prática

- Vamos voltar ao [SQL-DOJO](#);
- Execute o código data.py com o N=10000 e com as transações auto-commit=True. Qual o tempo de execução total?
(time python data.py)
- Execute o código data.py com o N=10000 e com as transações auto-commit=true. Qual o tempo de execução total?

Vacuum Postgres

Vacuum -> executa a limpeza das tuplas mortas sem reconstruir a página. Não reduz o espaço ocupado pela tabela.

Vacuum Full -> executa a limpeza reconstruindo a tabela em uma nova página (bloqueio). Reduz o espaço ocupado.

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;
```