

## Banco de Dados II

Transações

# Transação: Definição

Uma transação é uma unidade de execução definida pelo usuário que pode conter:

- Apenas uma consulta stand-alone (ex: comando SQL);
- Uma coleção de consultas stand-alone;
- Um programa inteiro que acessa o BD;

"Uma transação é uma unidade de execução definida pelo usuário que contém um ou mais acessos à Base de Dados."

(Hellman, Paul. *The Science of Database Management*. Burr Ridge, Ill.: Irwin, 1994.)

# Transação: Exemplo

# Transferência de fundos de uma conta bancária (C1) para outra (C2):

### **Operações:**

- Decrementar a conta C1
- Incrementar a conta C2

- > Atomicidade
- > Consistência
- > Isolamento
- Durabilidade

#### > Atomicidade:

> Executa todas as operações da transação ou nenhuma

#### > Consistência:

- A execução de uma transação deve levar o BD de um estado consistente a outro estado consistente.
- Escrever transações consistentes é uma responsabilidade do programador
- O SGBD deve ter mecanismos para garantir a consistência dos dados quando as transações são executadas concorrentemente (veja isolamento).

#### > Isolamento:

- A execução concorrente de transações deve deixar o BD em um dos estados possíveis de serem atingidos por alguma execução seqüencial;
- A execução de uma transação não pode interferir na execução de outra transação sendo executada simultaneamente;
- O SGBD deve ter mecanismos para garantir o isolamento das transações.

#### > Durabilidade:

- Alterações efetuadas por transações efetivadas devem persistir após a efetivação, mesmo se houver falhas;
- O SGBD deve ter mecanismos para garantir a durabilidade.

## Transação: Tamanho



Por questões de eficiência, uma transação não deve ser muito longa. Ela deve conter somente as operações estritamente necessárias para garantir a integridade do Banco de Dados.

## Transação: Exemplo



Transferência de fundos da conta A para a conta B:

```
T1: read(A,a) {Lê A (saldo da conta A) do BD no buffer local a}

a = a - 50

write(A,a) {Escreve o conteúdo do buffer local a para A do BD}

read(B,b) {Lê B (saldo da conta B) do BD no buffer local b}

b = b + 50

write(B,b) {Escreve o conteúdo do buffer local b para B do BD}
```

Atomicidade: Não se deve escrever o novo valor de a sem também escrever o novo valor de b. É função do SGBD garantir a atomicidade. Se o sistema falha depois de escrever em A e antes de escrever em B, ele precisa detectar a falha e restaurar o valor inicial de A.

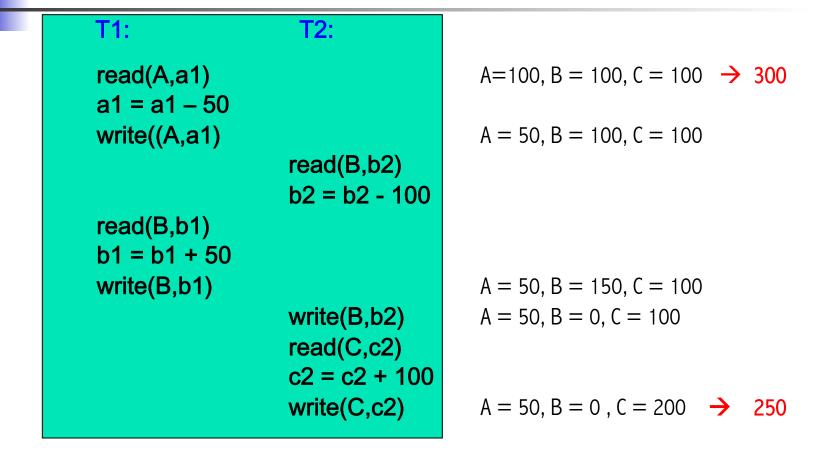
Consistência: Neste caso, a soma dos saldos de todas as contas não pode mudar (deve ser o mesmo antes e depois da execução da transação). É função do programador da aplicação garantir que a execução completa e isolada da transação preserva a consistência do banco de dados;

Durabilidade: Se a transação é concluída, então os novos valores de A e B devem persistir, mesmo se o sistema falhar após a conclusão da transação.

Durabilidade (cont.): O SGBD é responsável por garantir a durabilidade. Para tanto, ou (1) o SGBD garante que todos as escritas são executadas em disco, antes da transação ser concluída ou (2) o SGBD garante que informações suficientes sobre todas as escritas efetuadas pela transação são armazenadas em memória estável (arquivos de log) antes da transação ser concluída, de tal modo que, se o sistema falhar, o arquivo de log pode ser utilizado para reconstruir o banco de dados quando o sistema reiniciar.

Isolamento: O sistema deve garantir que nenhum estado inconsistente seja atingido pelo banco de dados após T1 ser executada concorrentemente com outras transações.

```
T2:
T1:
read(A,a1)
a1 = a1 - 50
write((A,a1)
                  read(B,b2)
                  b2 = b2 - 100
read(B,b1)
b1 = b1 + 50
write(B,b1)
                  write(B,b2)
                  read(C,c2)
                  c2 = c2 + 100
                  write(C,c2)
```



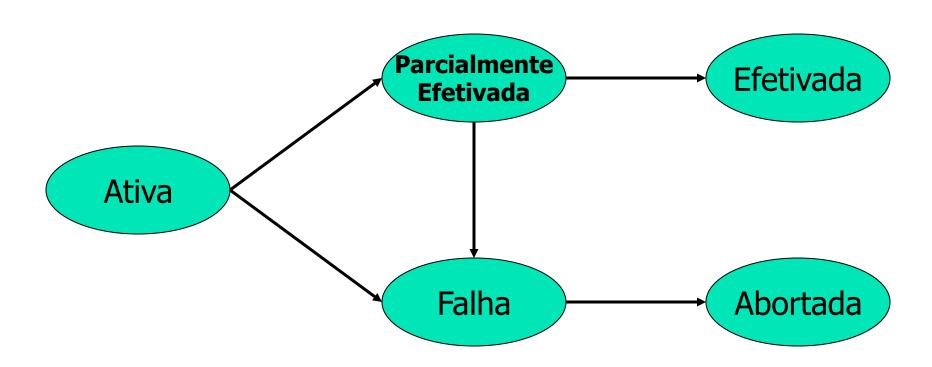
Esta execução concorrente resultaria numa perda de \$50 - um estado inconsistente. Tal execução concorrente deve ser evitada pelo sistema.

## Transação: Estados



- Ativa: Entra neste estado quando é iniciada e permanece nele enquanto está executando.
- Parcialmente Efetivada: Entra neste estado depois que a última operação da transação é executada. Neste estado, a transação ainda pode ser abortada.
- Efetivada: Entra neste estado depois de ser concluída com êxito.
- Falha: Entra neste estado se descobre que a execução normal não pode continuar.
- Abortada: Entra neste estado depois de ser desfeita (rolled back).

## Transação: Estados



## Transação: Estados



- Ativa: Executando as instruções da transação.
- Parcialmente Efetivada: Transação executada, mas o sistema ainda precisa garantir a durabilidade. Falhas podem ocorrer durante esta fase.
- Falha: Sistema desfaz (rolls back) as escritas efetuadas pela transação
- Abortada: Banco de dados está em um estado consistente que ele estava antes da transação iniciar.
- Efetivada: Banco de Dados está em um estado consistente criado pela transação.
- Concluída: Efetivada ou abortada.



- Concorrência: Duas ou mais transações executando em paralelo (simultaneamente).
- Benefícios: aumenta o throughput (melhor tempo de resposta).
- Problemas: Duas transações que, quando executadas serialmente, mantém a consistência, podem levar o banco de dados a um estado inconsistente, se forem executadas concorrentemente.

# Controle de Concorrência Escalas

- Escala: Sequência de execução de um conjunto de transações.
- Escalas Seriais: Para cada par de transações Ti e Tj do conjunto de transações, ou todas as instruções de Ti precedem todas as instruções de Tj, ou todas as instruções de Ti seguem todas as instruções de Tj.

# Controle de Concorrência Escalas Seriais

T0 precede T1

	T0	T1	
T0 transfere	read(A)		
\$50 de A	A = A - 50		
para B	write(A)		
	read(B)		
	B = B + 50		
	write(B).		
T1 transfere		read(A)	
10% do saldo		temp = A*0.1	
de A para B		A = A - temp	
•		write(A)	
		read(B)	
		B = B + temp	
		write(B)	
		<b>、</b> /	

### Escalas Seriais

T1 precede T0

	T0	T1	
T1 transfere		read(A)	
10% do saldo		temp = $A*0.1$	
de A para B		A = A - temp	
		write(A)	
		read(B)	
		B = B + temp	
		write(B)	
T0 transfere	read(A)		
\$50 de A	A = A - 50		
para B	write(A)		
	read(B)		
	B = B + 50		
	write(B).		

## Controle de Concorrência Escalas Seriais

Para um conjunto de n transações, existem n! escalas seriais possíveis.

### Escalas Concorrentes

T0	T1	
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)	

### Escalas Concorrentes

T0	T1	
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).	

Esta escala Preserva a Consistência do Banco de Dados?

# Controle de Concorrência Escalas Concorrentes

T0	T1	
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).	Esta escala Preserva a Consistência do Banco de Dados?

# Controle de Concorrência Escalas Concorrentes

O número de escalas concorrentes para um conjunto de n transações e muito maior do que n!

Se uma escala possui duas instruções consecutivas, Ii and Ij, de duas transações Ti e Tj, respectivamente, então podemos trocar Ii e Ij na escala sem alterar o efeito se Ii e Ij referemse a itens de dados diferentes.

Se Ii e Ij referem-se ao mesmo item de dados, então a ordem da execução é importante, pois pode produzir resultados diferentes.

### Mesmo item de dados

### **Quatro casos:**

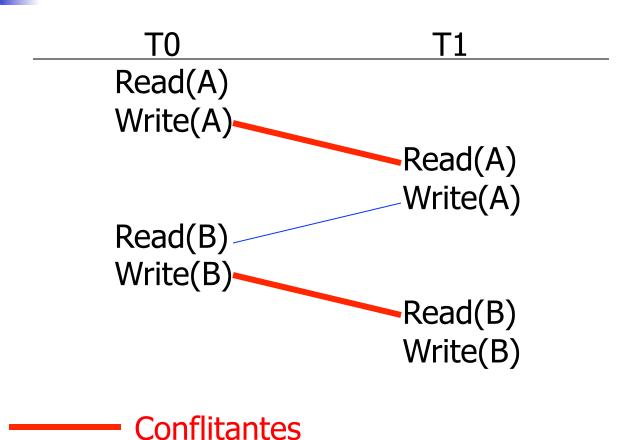
- Ii = read(Q) e Ij = read(Q)
- Ii = read(Q) e Ij = write(Q)
- Ii = write(Q) e Ij = read(Q)
- Ii = write(Q) e Ij = write(Q)

- Ordem não importa
- Ordem importa
- Ordem importa
- Ordem importa

### Instruções Conflitantes

### Definição:

Ii e Ij são conflitantes se elas são operações de diferentes transações sobre o mesmo item de dados e pelo menos uma delas é uma instrução write.



Não Conflitantes

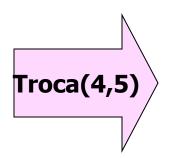
Duas instruções consecutivas em uma escala podem ser trocadas se elas não são conflitantes. A troca gera uma nova escala equivalente à escala original.

A execução de duas escalas equivalentes geram o mesmo resultado.

## Serialização no Conflito

#### 1º Passo

	T0	T1
1	Read(A)	
2	Write(A)	
3		Read(A)
4		Write(A)
5	Read(B)	
6	Write(B)	
7		Read(B)
8		Write(B)



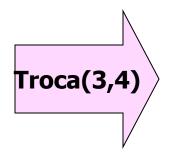
	T0	T1
1	Read(A)	
2	Write(A)	
3		Read(A)
4	Read(B)	
5		Write(A)
6	Write(B)	
7		Read(B)
8		Write(B)

Escala S1

## Serialização no Conflito

#### 2º Passo

	ТО	T1
1	Read(A)	
2	Write(A)	
3		Read(A)
4	Read(B)	
5		Write(A)
6	Write(B)	
7		Read(B)
8		Write(B)



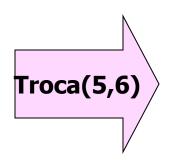
	T0	T1
1	Read(A)	
2	Write(A)	
3	Read(B)	
4		Read(A)
5		Write(A)
6	Write(B)	
7		Read(B)
8		Write(B)

Escala S2

## Serialização no Conflito

#### 3º Passo

	T0	T1
1	Read(A)	
2	Write(A)	
3	Read(B)	
4		Read(A)
5		Write(A)
6	Write(B)	
7		Read(B)
8		Write(B)



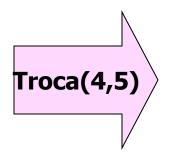
ТО	T1
1 Read(A)	
2 Write(A)	
3 Read(B)	
4	Read(A)
5 Write(B)	
6	Write(A)
7	Read(B)
8	Write(B)

Escala S3

## Serialização no Conflito

### 4º Passo

T0	T1
1 Read(A)	
2 Write(A)	
3 Read(B)	
4	Read(A)
5 Write(B)	
6	Write(A)
7	Read(B)
8	Write(B)



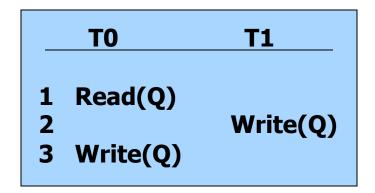
	T0	T1
1	Read(A)	
2	Write(A)	
3	Read(B)	
4	Write(B)	
5		Read(A)
6		Write(A)
7		Read(B)
8		Write(B)

Escala S4

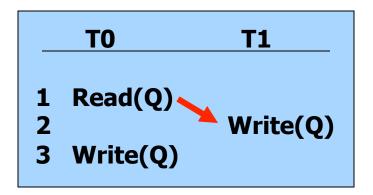
Se uma escala S1 pode ser transformada em uma escala S2 por meio de uma série de trocas de instruções não-conflitantes, então S1 e S2 são ditas escalas equivalentes no conflito.

Uma escala S é serializável no conflito se ela é equivalente no conflito à uma escala serial.

Exemplo de Escala Não-Serializável no Conflito

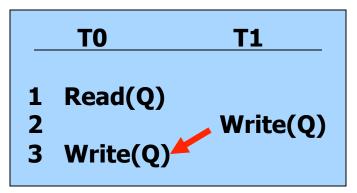


Exemplo de Escala Não-Serializável no Conflito



Conflito Read-Write

Exemplo de Escala Não-Serializável no Conflito



Conflito Write-Write

É possível existir duas escalas que não são equivalentes no conflito e que produzem o mesmo resultado?

É possível existir duas escalas que não são equivalentes no conflito e que produzem o mesmo resultado !!

	T0 T1	
1	Read(A)	
2	A ← A - 50	
5	Write(A)	
6	Read(B)	
7	$B \leftarrow B - 10$	
8	Write(B)	
9	Read(B)	
10	B ← B + 50	
11	Write(B)	
12	Read(A)	
13	$A \leftarrow A + 10$	
14	Write(A)	

	T0	T1
1	Read(A)	_
2	$A \leftarrow A - 50$	
3	Write(A)	
4	Read(B)	
5	$B \leftarrow B + 50$	
6	Write(B)	
7		Read(B)
8		$B \leftarrow B - 10$
9		Write(B)
10		Read(A)
11		$A \leftarrow A + 10$
12		Write(A)

	T0	T1	А В
1	Read(A)		1000, 2000
2	<b>A</b> ← <b>A</b> - <b>50</b>		950,2000
5	Write(A)		950,2000
6		Read(B)	950,2000
7		$B \leftarrow B - 10$	•
8		Write(B)	950,1990
9	Read(B)		
10	$B \leftarrow B + 50$		950,1990
11	Write(B)		950,2040
12		Read(A)	050 2040
13		$A \leftarrow A + 10$	950,2040
14		Write(A)	960,2040

	T0	T1
1	Read(A)	
2	$A \leftarrow A - 50$	
3	Write(A)	
4	Read(B)	
5	$B \leftarrow B + 50$	
6	Write(B)	
7		Read(B)
8		$B \leftarrow B - 10$
9		Write(B)
10		Read(A)
11		$A \leftarrow A + 10$
12		Write(A)

## Serialização no Conflito

	T0	T1	А В
1	Read(A)		1000, 2000
2	A ← A - 50		950,2000
5	Write(A)		930,2000
6		Read(B)	950,2000
7 8		<b>B</b> ← <b>B</b> − <b>10</b> Write( <b>B</b> )	050 1000
9	Read(B)	Write(b)	950,1990
10	$B \leftarrow B + 50$		950,1990
11	Write(B)		950,2040
12		Read(A)	950,2040
13		$A \leftarrow A + 10$	J30,20 <del>1</del> 0
14		Write(A)	960,2040

			Ì
	T0	T1	А В
1	Read(A)		1000, 2000
2	$A \leftarrow A - 50$		
3	Write(A)		950,2000
4	Read(B)		950,2000
5	$B \leftarrow B + 50$		<i>J</i> 50,2000
6	Write(B)		950,2050
7		Read(B)	950,2050
8		$B \leftarrow B - 10$	
9		Write(B)	950,2040
10		Read(A)	950,2040
11		$A \leftarrow A + 10$	960,2040
<b>12</b>		Write(A)	J00,20 <del>1</del> 0

# Controle de Concorrência Teste para Serialização no Conflito

Dada uma escala S, construa um Grafo de Precedência G=(V,E), onde os vértices V representam as transações pertencentes a S e E é o conjunto de todas as arestas Ti →Tj para as quais uma das 3 condições se verifica sobre o mesmo item de dado Q:

- 1. Ti executa write(Q) antes de Tj executa read(Q)
- 2. Ti executa read(Q) antes de Tj executar write(Q)
- 3. Ti executa write(Q) antes de Tj executar write(Q)

Uma aresta Ti → Tj no Grafo de Precedência diz que, em qualquer escala serial S', equivalente a S, Ti deve preceder Tj.