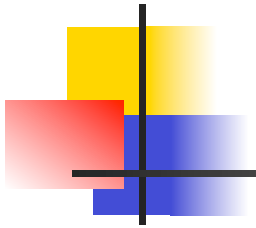
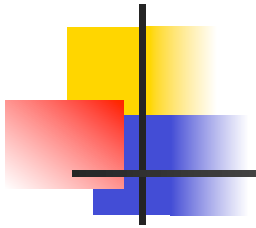


Banco de Dados II

Transações



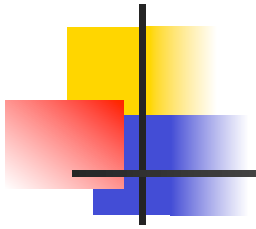
Transação: Definição

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.)



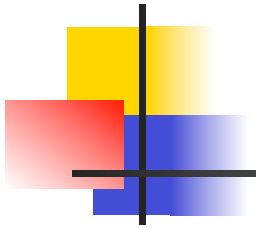
Transação: Exemplo

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

Operações:

- **Decrementar a conta A**
- **Incrementar a conta B**

Transação: Exemplo



Transferência de fundos da conta A para a conta B:

T1: read(A,a) {Lê A (saldo da conta A) do BD na memória local a}

a = a - 50

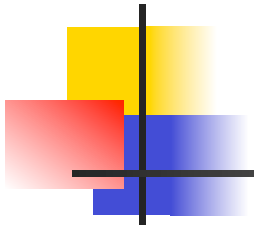
write(A,a) {Escreve o conteúdo da memória local a para A do BD}

read(B,b) {Lê B (saldo da conta B) do BD na memória local b}

b = b + 50

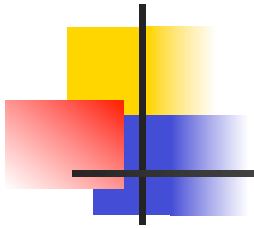
write(B,b) {Escreve o conteúdo da memória local b para B do BD}

Transação: Propriedades ACID



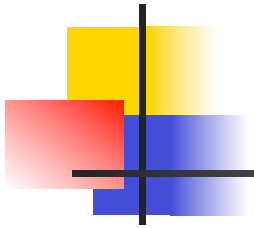
- **Atomicidade**
- **Consistência**
- **Isolamento**
- **Durabilidade**

Transação: Propriedades ACID



- **Atomicidade:**
  - **Executa <u>todas</u> as operações da transação ou <u>nenhuma</u>**

Transação: Propriedades ACID



- **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).**

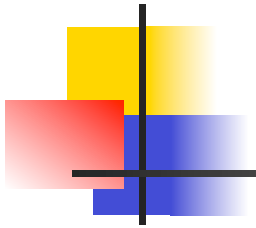
Transação: Propriedades ACID

- **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.**

Transação: Propriedades ACID

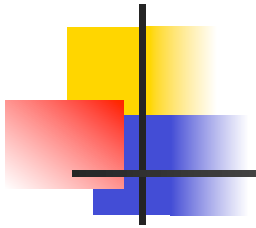
- **Durabilidade:**
  - **Alterações efetuadas por transações efetivadas devem persistir após a efetivação, mesmo se houver falhas;**
  - **O SGBD deve ter mecanismos para garantir a durabilidade.**

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: Propriedades



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.

Transação: Propriedades

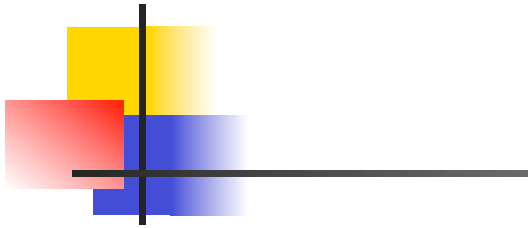
Durabilidade:

- 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.
- SGBD é responsável por garantir a durabilidade. Para tanto:
  - O SGBD garante que todos as escritas são executadas em disco, antes da transação ser concluída

  ou

  - 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.

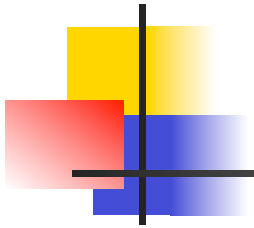
Transação: Propriedades



Isolamento:

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

Transação: Propriedades



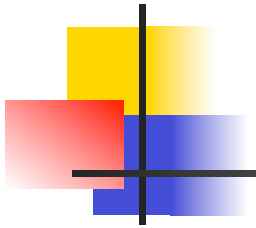
T1:	T2:
read(A,a1)	
a1 = a1 – 50	
write((A,a1)	
	read(B,b2)
	b2 = b2 - 100
read(B,b1)	
b1 = b1 + 50	
write(B,b1)	
	write(B,b2)
	read(C,c2)
	c2 = c2 + 100
	write(C,c2)

Transação: Propriedades

T1:	T2:	
read(A,a1)		A=100, B = 100, C = 100 → 300
a1 = a1 – 50		
write((A,a1)		A = 50, B = 100, C = 100
	read(B,b2)	
	b2 = b2 - 100	
read(B,b1)		
b1 = b1 + 50		
write(B,b1)		A = 50, B = 150, C = 100
	write(B,b2)	A = 50, B = 0, C = 100
	read(C,c2)	
	c2 = c2 + 100	
	write(C,c2)	A = 50, B = 0 , C = 200 → 250

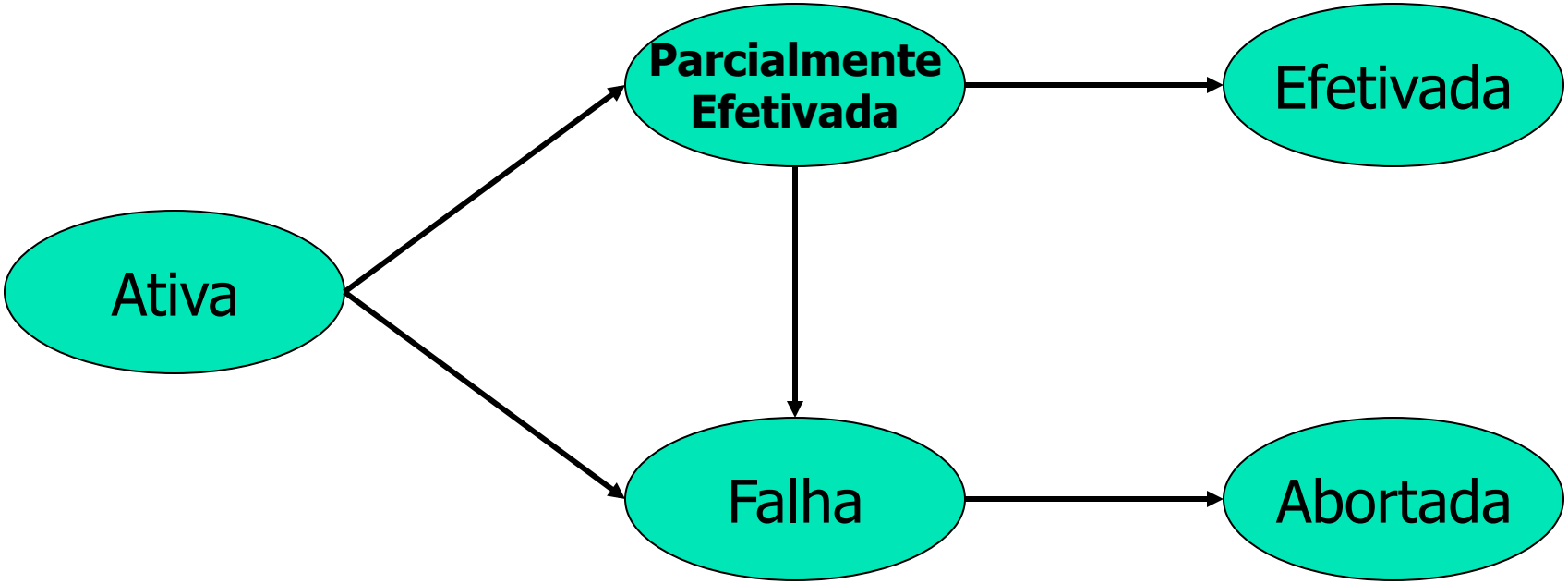
Esta execução concorrente resultaria numa perda de $50 – um estado inconsistente. Tal execução concorrente deve ser evitada pelo sistema.

Transação: Estados

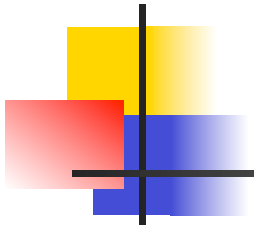


- **Ativa:** Entra neste estado quando é iniciada e permanece nele enquanto está executando;
- **Parcialmente Efetivada:** Entra neste estado depois que a última operação da transação é executada. Neste estado, a transação ainda pode ser abortada;
- **Efetivada:** Entra neste estado depois de ser concluída 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).

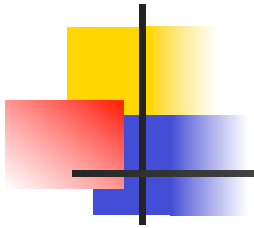
Transação: Estados



Transação: Estados

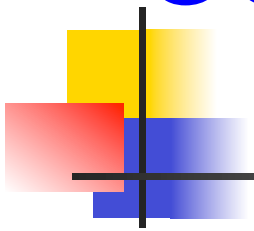


- **Ativa:** Executando as instruções da transação;
- **Parcialmente Efetivada:** Transação executada, mas o sistema ainda precisa garantir a durabilidade. Falhas podem ocorrer durante esta fase;
- **Falha:** Sistema desfaz (rolls back) as escritas efetuadas pela transação;
- **Abortada:** Banco de dados 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.



Controle de Concorrência

Controle de Concorrência



- **Concorrência:** Duas ou mais transações executando 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:
  - todas as instruções de Ti precedem todas as instruções de Tj

  ou
  - todas as instruções de Ti sucedem todas as instruções de Tj.

Controle de Concorrência

Escalas Seriais

T0 precede T1

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

Controle de Concorrência

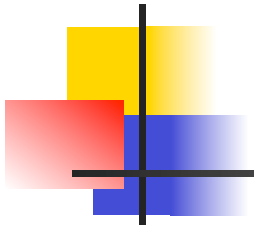
Escalas Seriais

T0 sucede T1

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

Controle de Concorrência

Escalas Seriais

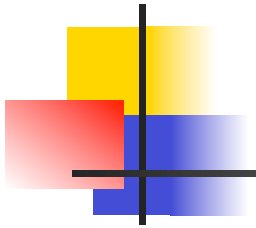


Para um conjunto de n transações,

existem n! escalas seriais possíveis.

Controle de Concorrência

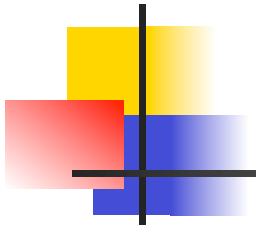
Escalas Concorrentes



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)

Controle de Concorrência

Escalas Concorrentes

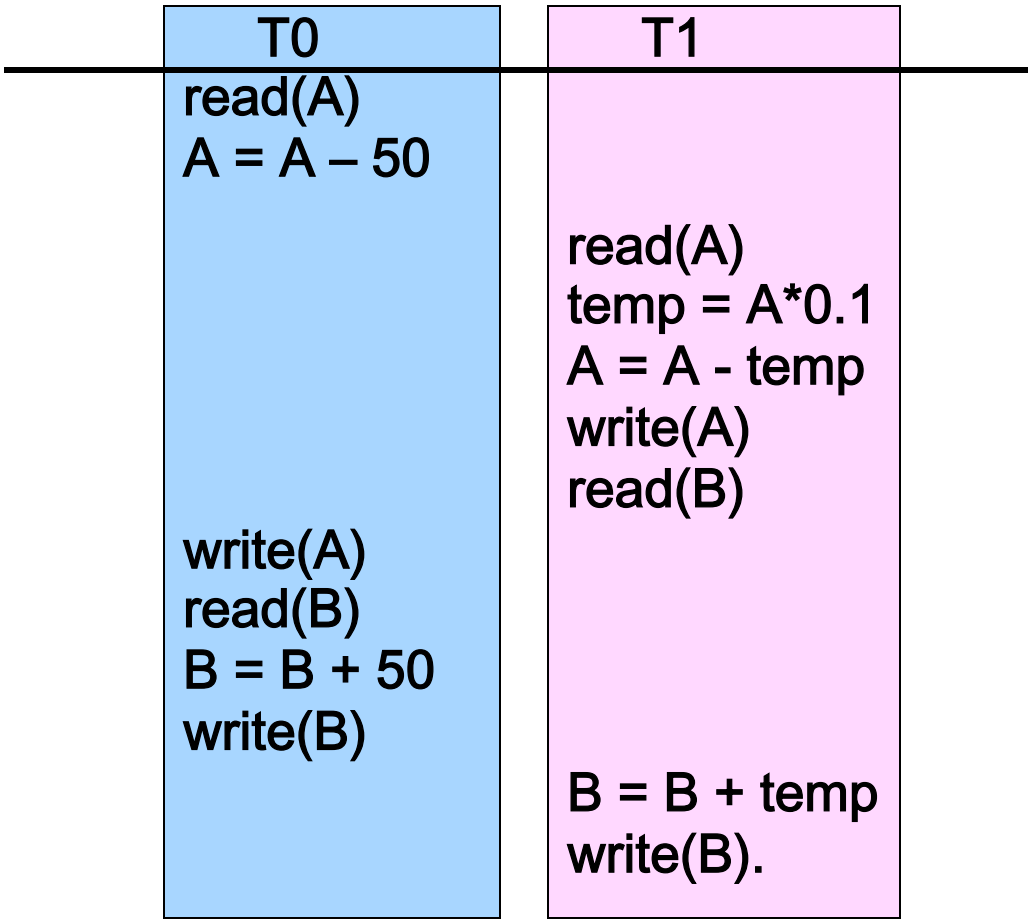


T0	T1
read(A)	
A = A – 50	
write(A)	
	read(A)
	temp = A*0.1
	A = A - temp
	write(A)
read(B)	
B = B + 50	
write(B)	
	read(B)
	B = B + temp
	write(B).

Esta escala Preserva a Consistência do Banco de Dados?

Controle de Concorrência

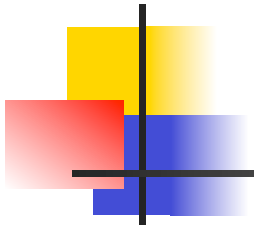
Escalas Concorrentes



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 é muito maior do que n!

Ele depende também da quantidade de instruções de cada transação.

Controle de Concorrência

Escalas Serializáveis no Conflito

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 referem-se 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.

Controle de Concorrência

Escalas Serializáveis no Conflito

Mesmo item de dados

Quatro casos:

- Ii = read(Q) e Ij = read(Q) - Ordem não importa
- Ii = read(Q) e Ij = write(Q) - Ordem importa
- Ii = write(Q) e Ij = read(Q) - Ordem importa
- Ii = write(Q) e Ij = write(Q) - Ordem importa

Controle de Concorrência

Escalas Serializáveis no Conflito

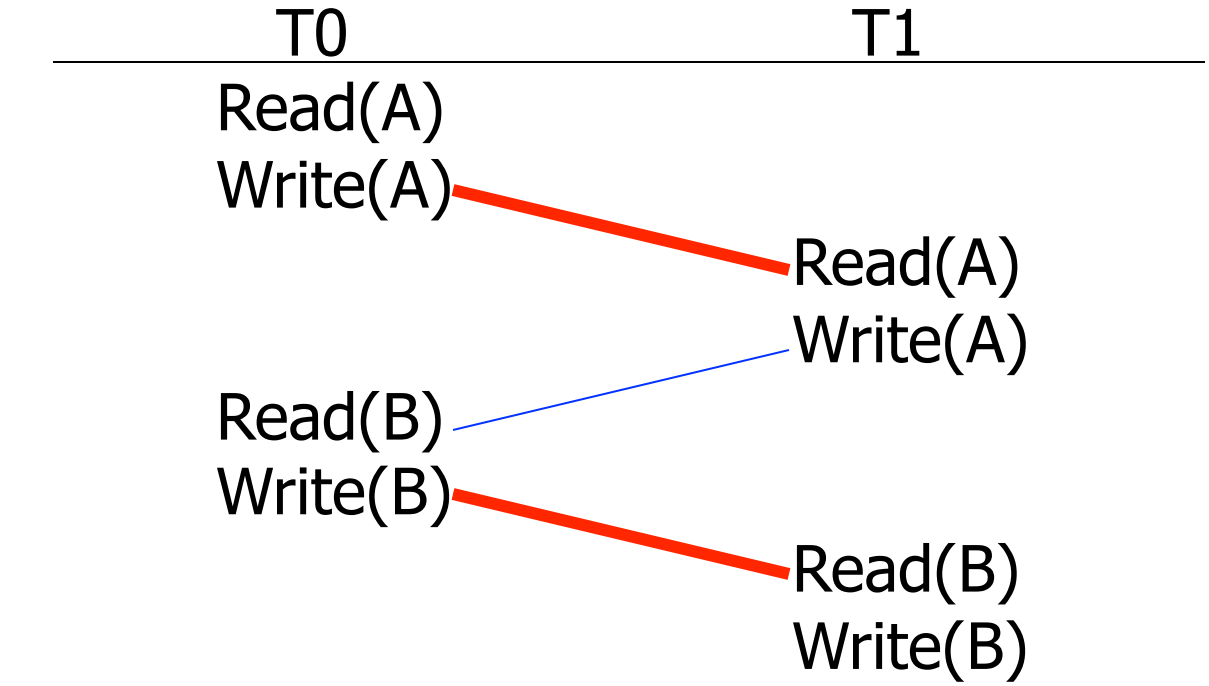
Instruções Conflitantes

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.

Controle de Concorrência

Escalas Serializáveis no Conflito



Conflitantes

Não Conflitantes

Controle de Concorrência

Escalas Serializáveis no Conflito

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.

Controle de Concorrência

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)

Escala S1

Troca(4,5)

	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 S2

Controle de Concorrência

Serialização no Conflito

2º Passo

	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 S2

Troca(3,4)

	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 S3

Controle de Concorrência

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)

Escala S3

Troca(5,6)

	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)

Escala S4

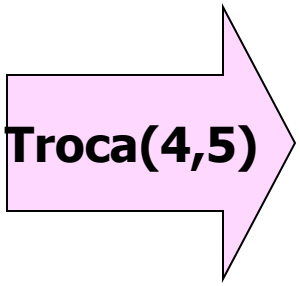
Controle de Concorrência

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)

Escala S4

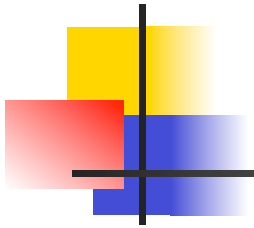


	T0	T1
1	Read(A)	
2	Write(A)	
3	Read(B)	
4	Write(B)	
5		Read(A)
6		Write(A)
7		Read(B)
8		Write(B)

Escala S5

Controle de Concorrência

Serialização no Conflito



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.

Controle de Concorrência

Serialização no Conflito

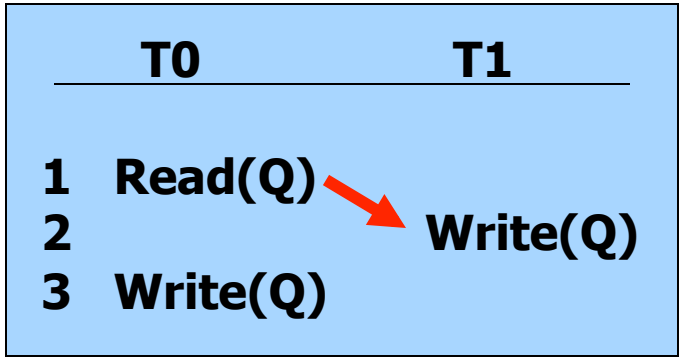
Exemplo de Escala Não-Serializável no Conflito

	T0	T1
1	Read(Q)	
2		Write(Q)
3	Write(Q)	

Controle de Concorrência

Serialização no Conflito

Exemplo de Escala Não-Serializável no Conflito

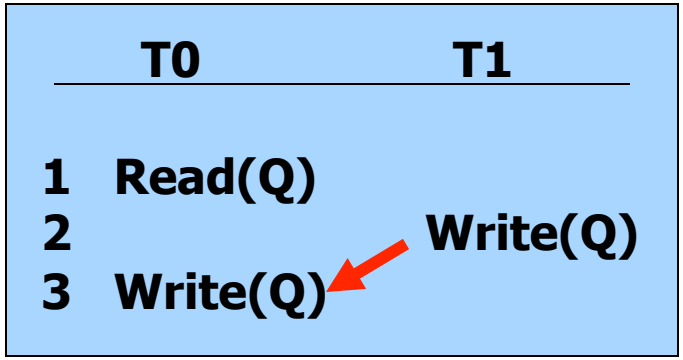


Conflito Read-Write

Controle de Concorrência

Serialização no Conflito

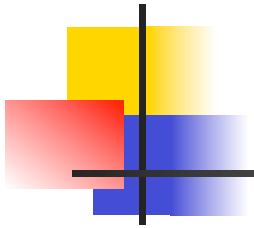
Exemplo de Escala Não-Serializável no Conflito



Conflito Write-Write

Controle de Concorrência

Serialização no Conflito



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

Controle de Concorrência

Serialização no Conflito

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	
3	Write(A)	
4		Read(B)
5		B ← B - 10
6		Write(B)
7	Read(B)	
8	B ← B + 50	
9	Write(B)	
10		Read(A)
11		A ← A + 10
12		Write(A)

	T0	T1
1	Read(A)	
2	A ← A – 50	
3	Write(A)	
4	Read(B)	
5	B ← B + 50	
6	Write(B)	
7		Read(B)
8		B ← B – 10
9		Write(B)
10		Read(A)
11		A ← A + 10
12		Write(A)

Controle de Concorrência

Serialização no Conflito

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	
3	Write(A)	
4		Read(B)
5		B ← B - 10
6		Write(B)
7	Read(B)	
8	B ← B + 50	
9	Write(B)	
10		Read(A)
11		A ← A + 10
12		Write(A)

A	B
1000,	2000
950,	2000
950,	2000
950,	1990
950,	1990
950,	2040
950,	2040
960,	2040

	T0	T1
1	Read(A)	
2	A ← A – 50	
3	Write(A)	
4	Read(B)	
5	B ← B + 50	
6	Write(B)	
7		Read(B)
8		B ← B – 10
9		Write(B)
10		Read(A)
11		A ← A + 10
12		Write(A)

Controle de Concorrência

Serialização no Conflito

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

	T0	T1	A	B
1	Read(A)		1000,	2000
2	A ← A – 50			
3	Write(A)		950,	2000
4		Read(B)		
5		B ← B - 10	950,	2000
6		Write(B)	950,	1990
7	Read(B)			
8	B ← B + 50		950,	1990
9	Write(B)		950,	2040
10		Read(A)		
11		A ← A + 10	950,	2040
12		Write(A)	960,	2040

	T0	T1	A	B
1	Read(A)		1000,	2000
2	A ← A – 50			
3	Write(A)		950,	2000
4	Read(B)			
5	B ← B + 50		950,	2000
6	Write(B)		950,	2050
7		Read(B)	950,	2050
8		B ← B – 10		
9		Write(B)	950,	2040
10		Read(A)	950,	2040
11		A ← A + 10		
12		Write(A)	960,	2040

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.