

Banco de Dados II

Gerenciador de Transações

Cap. 15 (Silberschatz)

Cap. 16 (Ramakrishnan)

Denio Duarte



Introdução

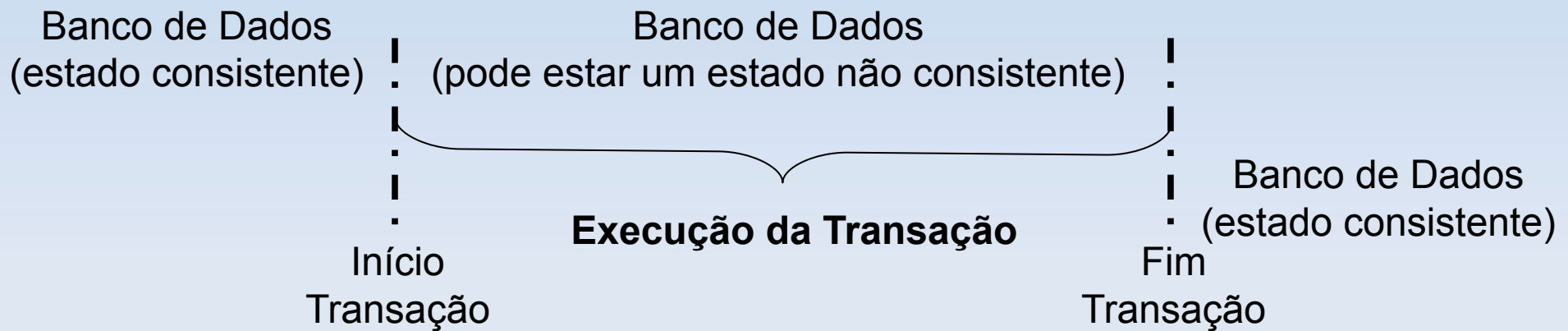
- **Transação**

- Sequência de operações (sendo atualização a operação principal) que desempenha uma função específica dentro de uma aplicação.
- Uma transação bancária de transferência de valores entre contas é uma operação única para o usuário porém é composta de várias operações.
- É uma unidade de execução de programa que acessa, e possivelmente atualiza, vários itens de dados em um banco de dados.

Introdução

■ Transação

- Um sistema gerenciador de banco de dados (SGBD) deve garantir a execução apropriada das transações em caso de falhas e execução simultânea.
- Para evitar problemas, um SGBD deve garantir a execução correta de uma transação.



Introdução

- **Transação**

- Atualiza o saldo de uma conta

```
UPDATE Contas  
SET Saldo = Saldo - 50  
WHERE NoConta = 094567;
```

- Transação com mais operações: transfere 50 reais da conta 094567 para conta 462364

```
UPDATE Contas  
SET Saldo = Saldo - 50  
WHERE NoConta = 094567;
```

```
UPDATE Contas  
SET Saldo = Saldo + 50  
WHERE NoConta = 462364;
```

Introdução

```
UPDATE Contas  
SET Saldo = Saldo - 50  
WHERE NoConta = 094567;
```

```
UPDATE Contas  
SET Saldo = Saldo + 50  
WHERE NoConta = 462364;
```

- Perceba que se uma das transações falhar o banco de dados fica inconsistente
 - Ou um cliente fica com 50 reais a menos ou outro fica com 50 reais a mais.

Introdução

■ Transação

- Fim normal = **commit** / Fim anormal = **abort (rollback)**

Begin Transaction

Input (ccor,ccde,val);

SELECT saldo, lim into s,l **FROM** cc **WHERE** ccnb=ccor;

If (s+l < val) **rollback**;

// atualiza cc origem

UPDATE cc

SET saldo = saldo - val

WHERE ccnb=ccor;

// atualiza cc destino

UPDATE cc

SET saldo = saldo + val

WHERE ccnb=ccde;

End Transaction

Introdução

- **Simplificando as operações**

Read (A)

A=A-50

Write (A)

Read (B)

B=B+50

Write (B)

- Onde A e B representam os saldos das duas contas correntes conhecidas
- Note: a transação tem sua própria **região de memória** (os dados são **locais** à ela)

Introdução

- Dadas duas contas A e B, ambas com 100 reais de saldo

- Transação T_1

```
Read(A)
A=A-50
Write(A)
Read(B)
B=B+50
Write(B)
```

- Mesma anterior

- Transação T_2

```
Read(A)
A=A+300
Write(A)
```

- Deposita 300 reais na conta A

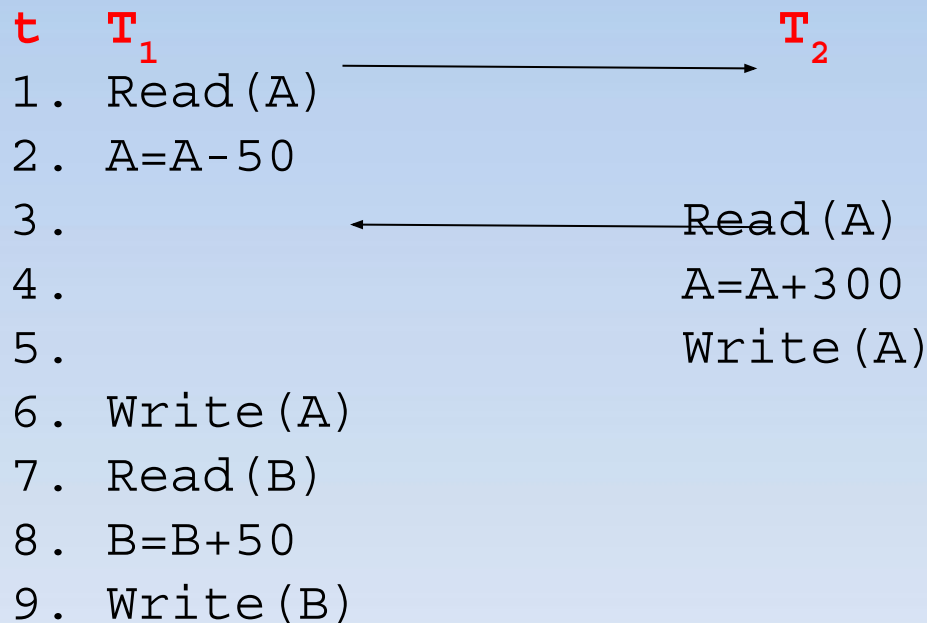
Após execução de T_1 e T_2 o saldo de A é 350 e B é 150.

Problemas

- O SGBD não **deve** executar as transações sequencialmente
 - Subutilização do processador
 - Demora na resposta das atualizações do usuário
- Executar transações em paralelo pode causar problemas

Problemas

- Atualização perdida (lost update) – conflito **WW**



Após a execução das transações T_1 e T_2 , $A=50$ e $B=150$
O **write(A)** no $t=5$ de T_2 foi "perdido".

Problemas

- Leitura suja (dirty read) – conflito **WR**

t	T₁	T₂
1.	Read(A)	
2.	A=A-50	
3.	Write(A)	
4.	_____	Read(A)
5.		A=A+300
6.	←_____	Write(A)
7.	Abort!	

A transação T_2 no tempo (**t=4**) leu o valor **50** de **A** porém T_1 abortou (**t=7**), assim T_2 está com um valor "sujo" da conta **A**.

Problemas

- Sumarização incorreta* (aplicação reserva de assentos em um vôo) – phantom tuples

t	T₁	T₃
1.		Soma=0
2.		Read (P)
3.		Soma+=P
4.	Read (X)	
5.	X=X-N	
6.	Write (X)	
7.		Read (X)
8.		Soma+=X
9.		Read (Y)
10.		Soma+=Y
11.	Read (Y)	
12.	Y=Y+N	
13.	Write (Y)	

* conhecido como registro fantasma

Problemas

■ Sumarização incorreta (aplicação reserva de assentos em um voo)

t	T_1	T_3
1.		Soma=0
2.		Read (P)
3.		Soma+=P
4.	Read (X)	
5.	X=X-N	
6.	Write (X)	
7.		Read (X)
8.		Soma+=X
9.		Read (Y)
10.		Soma+=Y
11.	Read (Y)	
12.	Y=Y+N	
13.	Write (Y)	

A transação T_3 está calculando o número total de reservas em todos os voos da cia aérea. Durante a execução de T_3 , T_1 é disparada ($t=4$). Se acontecer a intercalação de operações como mostrado ao lado, o resultado de T_3 não contabilizará N , pois T_3 leu o valor de X ($t=7$) depois que os N assentos foram subtraídos, mas lerá o valor Y ($t=9$) antes que esses N assentos tenham sido adicionados a Y ($t=12$).

Problemas

Leitura não repetida (non-repeatable read) –
conflito **RW**

(suponha $A=10$)

t	T_1	T_2
1.	Read (A)	
2.		Read (A)
3.		$A=A+10$
4.		Write (A)
5.	:	
6.	:	
7.	Read (A)	
8.	:	
9.	:	

T_1 lê o valor de $A=10$ ($t=1$) quando tenta ler A de novo ($t=7$) obterá **20**
pois A foi alterado por T_2 ($t=2$, $t=3$ e $t=4$)

Propriedades

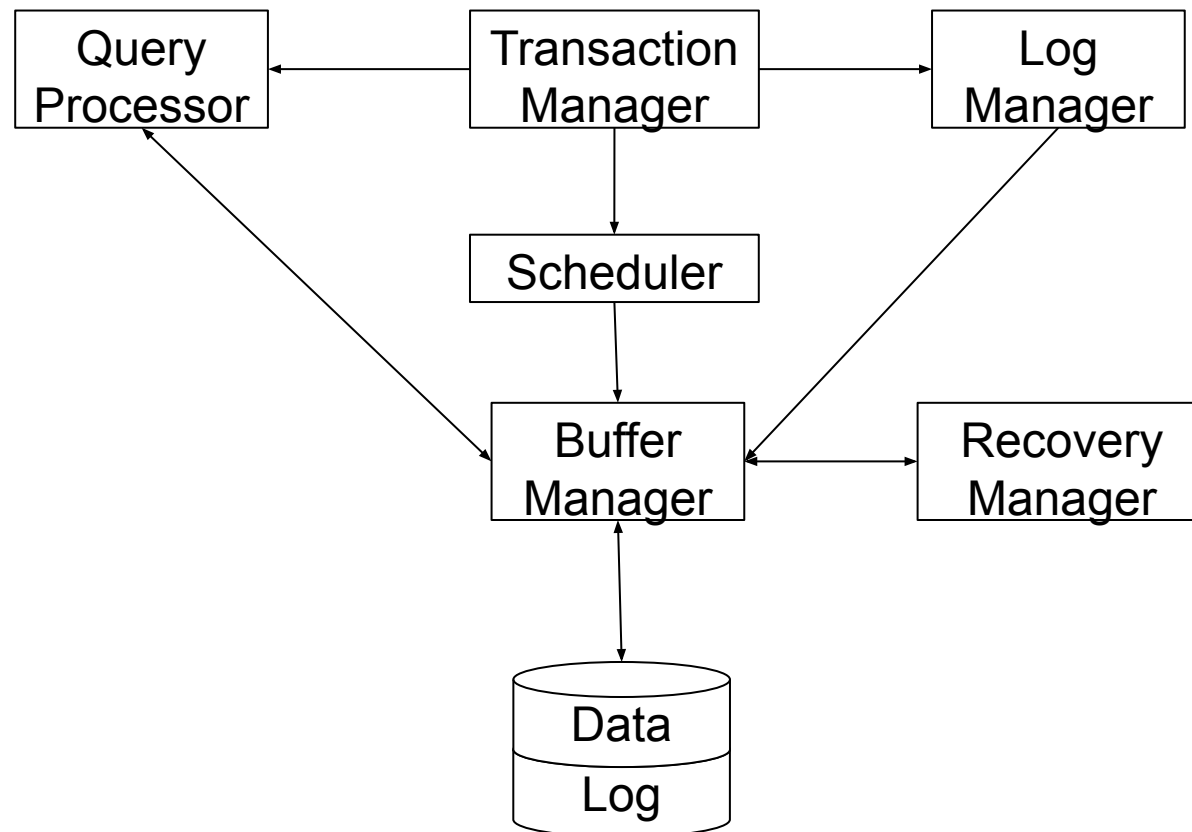
- Tais problemas não ocorrerão se as transações respeitarem a propriedade ACID:
 - **Atomicidade**: a transação não pode ser dividida, ou é executada por completo (operações refletidas no BD) ou não é executada.
 - **Consistência**: a transação ao finalizar deve manter a consistência dos dados do banco, ou seja, o estado corrente do banco de dados deve respeitar o conjunto de restrições de integridade definido.

Propriedades

- As transações para resolver tais problemas devem respeitar a propriedade ACID:
 - **Isolamento**: a execução da transação deve ocorrer sem interferência, como se a mesma estivesse executando sozinha no processador.
 - **Durabilidade**: as alterações nos dados feitas por uma transação terminada com sucesso deve ser persistido no banco de dados "para sempre".

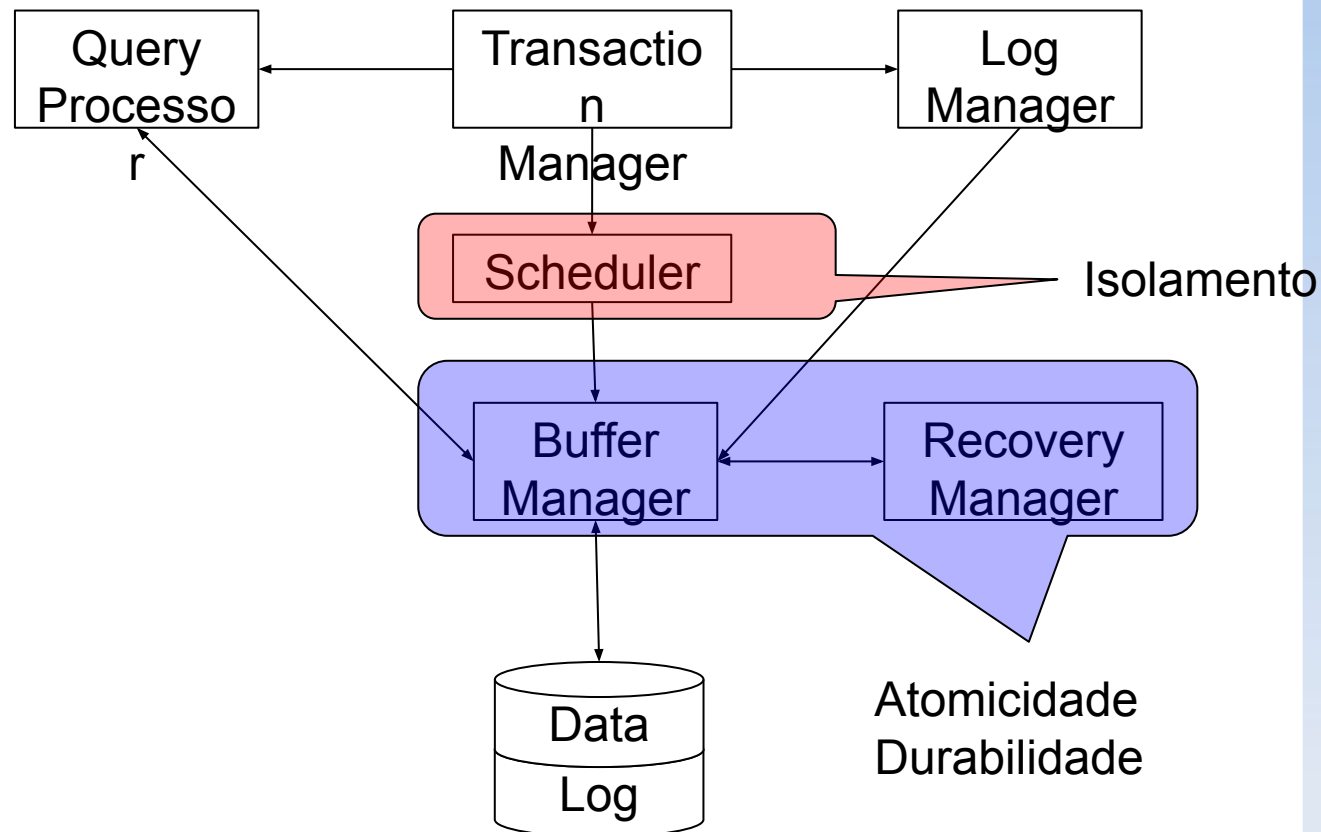
Gerenciador de Transações

- Responsável por manter a propriedade ACID



Gerenciador de Transações

- Responsável por manter a propriedade ACID



Gerenciador de Transações

- Garantia de ACID pelo GT:

t T_1

1. Read(A)
2. A=A-50
3. Write(A)
4. Read(B)
5. B=B+50
6. Write(B)

- Atomicidade: Se a transação falhar após o tempo 3 e antes do tempo 6, o GT deve garantir que as atualizações feitas não seja refletidas no banco. Essa característica é garantida pelo gerenciador de transação.

Gerenciador de Transações

- Garantia de ACID pelo GT:

t T_1

1. Read(A)
2. $A = A - 50$
3. Write(A)
4. Read(B)
5. $B = B + 50$
6. Write(B)

- Consistência:** após a transação finalizar, o item B deve ter 50 a mais e o item A , 50 a menos, conforme definido em T_1 , além de a soma de $A+B$ após o término de T_1 ser igual à antes do início de T_1 .

Na maioria das vezes a consistência é garantida pelo programador da aplicação, pois se ele fizer $A-30$ e $B+40$, a consistência desses dois itens não pode ser garantida.

Gerenciador de Transações

- Garantia de ACID pelo GT:

t T_1

1. Read(A)
2. $A = A - 50$
3. Write(A)
4. Read(B)
5. $B = B + 50$
6. Write(B)

- Isolamento:** Se entre os tempos **3** e **6**, uma outra transação T_2 receber permissão de acessar o banco de dados parcialmente atualizado, ele verá um banco de dados inconsistente (a soma $A + B$ será menor do que deveria ser). O isolamento poderia ser assegurado executando as transações serialmente. Porém essa característica subutilizaria o poder de processamento da máquina hospedeira. Assim, executar múltiplas transações simultaneamente oferece vantagens significativas (a forma de orquestrar essa execução será vista mais tarde). Garantida pelo gerenciador de transação e de bloqueios.

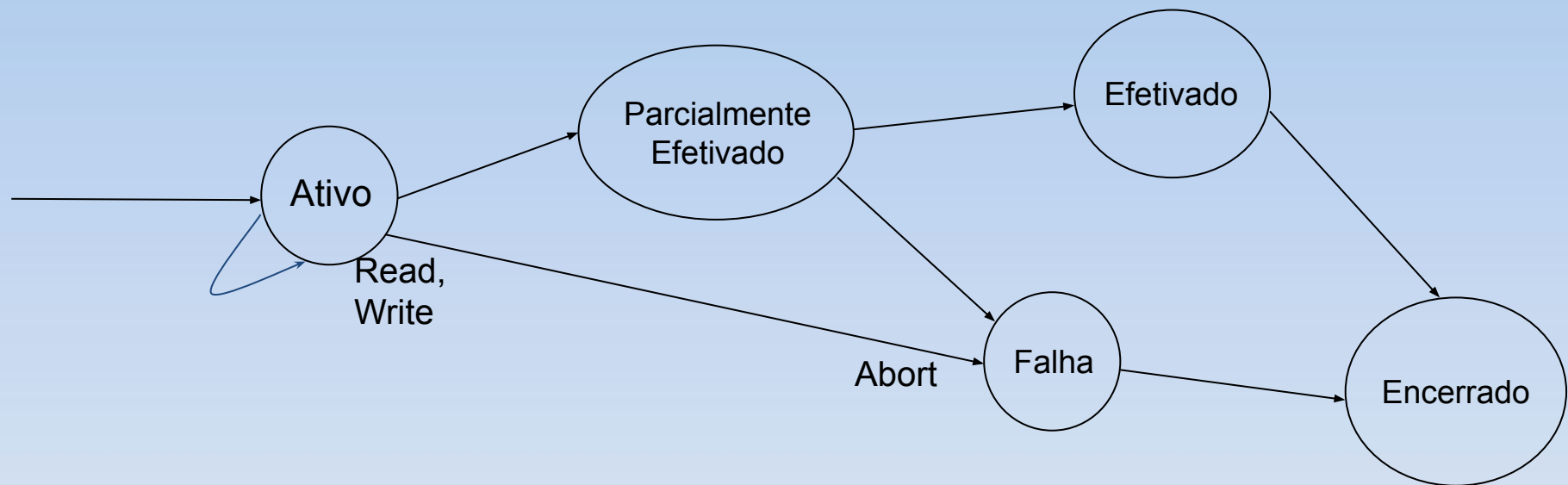
Estados

- Comandos transacionais e estados:
 - **BeginTransaction**: marca o início da transação
 - Transação passa para o estado **Ativo**
 - **EndTransaction**: especifica que todas as operações da transação terminaram.
 - Marca o fim da transação.
 - Verifica se as operações devem ser persistidas (caso **Commit**) ou abortadas (caso **Rollback**). Passa o estado **Parcialmente Efetivado**

Estados

- Comandos transacionais e estados:
 - **Commit:** indica o término com sucesso da transação (alterações devem ser efetivadas)
 - Passa para o estado **Efetivado**
 - **Rollback:** indica o término sem sucesso da transação (alterações devem ser ignoradas)
 - Passa para o estado **Falha**
 - **Encerrado:** transação deixa de existir

Estados



Transações Concorrentes

- Uma transação possui diversos passos (conjuntos de Reads, Writes e processamentos)
- Instruções de entrada/saída (E/S) e de processador podem operar em paralelo
 - Uma transação faz I/O, outra é processada pela CPU etc.
 - Processador e disco ficam menos tempo inativos: mais transações por tempo
 - Reduz o tempo médio de resposta: várias transações podem ser executadas concorrentemente

Transações Concorrentes

- **Concorrência** versus **consistência**
 - O sistema de banco de dados deve controlar a interação entre transações concorrentes para garantir a consistência dos dados
 - Identificar quais ordens de execução (escalas de execução) podem garantir a manutenção da consistência
- As operações das transações são executadas de forma intercalada definida pelo *escalador*

Transações Concorrentes

- O escalanador deve garantir o isolamento (e a consistência)
 - Mesmo que as operações estejam intercaladas, a transação deve executar como se estivesse sozinha
- O maior problema do escalonador é criar um escalonamento das operações das diversas transações de modo a garantir o *isolamento* (e a *consistência*).

Escalonamentos

- S_1 : garante o isolamento e a consistência pois as transações estão executando sequencialmente

tempo	T_1	T_2
0	Read(A)	
1	A=A-50	
2	Write(A)	
3	Read(B)	
4	B=B+50	
5	Write(B)	
6		Read(A)
7		temp=A*0.1
8		A=A-temp
9		Write(A)
10		Read(B)
11		B=B+temp
12		Write(B)

Escalonamentos

- S_2 : também garante o isolamento e a consistência
 - De novo as transações estão executando sequencialmente

tempo	T_1	T_2
0		Read(A)
1		temp=A*0.1
2		A=A-temp
3		Write(A)
4		Read(B)
5		B=B+temp
6		Write(B)
7	Read(A)	
8	A=A-50	
9	Write(A)	
10	Read(B)	
11	B=B+50	
12	Write(B)	

Escalonamentos

- S_3 : escalonamento concorrente que mantém o isolamento e a consistência.
 - Equivale a qual escalonamento serial?

tempo	T_1	T_2
0	Read(A)	
1	A=A-50	
2	Write(A)	
3		Read(A)
4		temp=A*0.1
5		A=A-temp
6		Write(A)
7	Read(B)	
8	B=B+50	
9	Write(B)	
10		Read(B)
11		B=B+temp
12		Write(B)

Escalonamentos

- S_3 : escalonamento concorrente que mantém o isolamento e a consistência.
 - Equivale a S_1

tempo	T_1	T_2
0	Read(A)	
1	A=A-50	
2	Write(A)	
3		Read(A)
4		temp=A*0.1
5		A=A-temp
6		Write(A)
7	Read(B)	
8	B=B+50	
9	Write(B)	
10		Read(B)
11		B=B+temp
12		Write(B)

Escalonamentos

- S_4 : escalonamento concorrente porém não consistente.
 - Por quê?

tempo	T_1	T_2
0	Read(A)	
1	A=A-50	
2		Read(A)
3		temp=A*0.1
4		A=A-temp
5		Write(A)
6		Read(B)
7	Write(A)	
8	Read(B)	
9	B=B+50	
10	Write(B)	
11		B=B+temp
12		Write(B)

Conclusões

- Os escalonamentos devem ser feitos de tal forma que as transações pareçam ter sido executadas serialmente: escalonamentos serializáveis
- Ordem
 - Se duas transações **apenas lêem** itens a ordem não é importante
 - Se duas transações **escrevem** itens **diferentes** a ordem também não é importante
 - Se uma transação **escreve** um item enquanto outra **lê** ou escreve o **mesmo item**, a ordem é importante

Conclusões

- Nível de isolamento das transações
 - Mede a independência de uma transação em relação às alterações nos dados por ela lidos feitas por outras transações.
 - Uma transação tem um elevado nível de isolamento se for absolutamente imune a essas alterações.
 - Pelo contrário, será pouco isolada se os seus resultados perderem integridade com as alterações feitas por outras transações.

Conclusões

- O SQL-92 define 4 níveis de isolamento que consideram os três problemas que podem ocorrer na execução concorrente de transações:
 - Dirty reads (leituras sujas)
 - Nonrepeatable reads (leituras não repetidas)
 - Phantom reads (leituras fantasmas / sumarização incorreta)

Conclusões

- Níveis de isolamento (SQL-92)
 - **Serializable** (padrão): garante total isolamento
 - **Repeatable read**: leituras sucessivas do mesmo registro durante a transação devolvem o mesmo valor. No entanto, não resolve o problema das tuplas fantasmas/sumarização (phantom records)

Conclusões

- Níveis de isolamento (SQL-92)
 - **Read committed**: somente podem ser lidos registros de transações completadas (committed records). No entanto, leituras sucessivas do mesmo registro durante a transação podem devolver valores diferentes (Problema de unrepeatable read) . Evita problemas do tipo **Dirty Read** e **Lost Update**
 - **Read uncommitted**: registros modificados em uma transação não completada (committed records) podem ser lidos. Somente para leitura, não permite atualizações. Podem ser apresentados problemas do tipo **Dirty Read**

Conclusões

Nível	D Read	NR Read	Ph Read
<i>Read Uncommitted</i>	Pode	Pode	Pode
<i>Read Committed</i>	Não	Pode	Pode
<i>Repeatable read</i>	Não	Não	Pode
<i>Serializable</i>	Não	Não	Não

Conclusões

- Comando para definir níveis de isolamento (PostgreSQL)

```
SET TRANSACTION ISOLATION LEVEL
    {SERIALIZABLE |
     REPEATABLE READ |
     READ COMMITTED |
     READ UNCOMMITTED }
    READ WRITE | READ ONLY
    [ NOT ] DEFERRABLE
```