INSTITUTO FEDERAL
Paraíba
Campus João Pessoa

Aula
9

Banco de Dados II

Atualizada em 21/11/2019

Transações e Tolerância a Falhas

Professor:

Dr. Alex Sandro da Cunha Rêgo

Email *alex@ifpb.edu.br*

Conteúdo

- **Transações**
  - Introdução
  - Propriedades A.C.I.D.
  - Diagrama de estado
- LOG
  - Registros de Controle de Transações
- Checkpoint
- Algoritmo R.A.A
- Algoritmo R.A.I.
- Paginação Sombra

Transações

- **Definição**
  - Coleção de várias operações em dados de um SGBD que formam uma **única unidade lógica de trabalho**
  - Uma unidade de execução do programa que **acessa** e possivelmente **atualiza** vários itens de dados.
- Por que utilizar **Transações**?
  - Garantir a execução apropriada de um conjunto de operações SQL **mesmo na ocorrência de falhas**
- **Exemplo clássico**
  - Transferência bancária de valores de uma **conta de origem** para uma **conta de destino**

Transações

- **Como uma transação é "ativada"?**
  - Geralmente é iniciada por um programa de usuário escrito em **linguagem de alto nível** ou **linguagem de manipulação de dados**
- Estrutura Genérica de Declaração

```
BEGIN TRANSACION
   operações PL/SQL;
   ...
   COMMIT;
   ...
   ROLLBACK;
END TRANSACTION
```

  - A **transação** consiste em todas as operações executadas entre o **begin** e **end transaction**

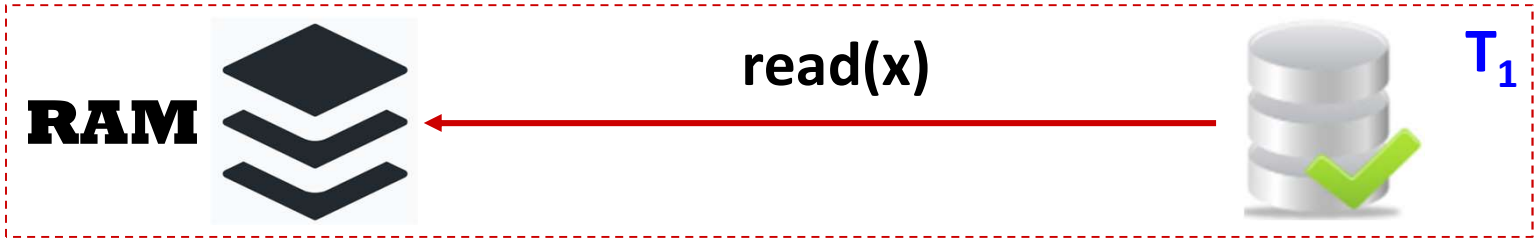
Transação

- **Propriedades de uma Transação**
  - **Atomicidade:**
    - ✓ Todas as operações são efetivadas ou nenhuma será!
  - **Consistência:**
    - ✓ Isoladamente, sua execução deve preservar a consistência dos dados do banco de dados
  - **Isolamento:**
    - ✓ Embora várias transações estejam executando **simultaneamente,** o sistema garante que as transações se comportem como se fosse a única em execução
  - **Durabilidade:**
    - ✓ As mudanças realizadas por uma transação no BD persistem, **mesmo no caso de falhas**

A C I D

Transação: ACID em ação

- **Instruções de acesso aos dados**
  - `READ(x)`: **recupera** o item de dados **x** do BD para um buffer local pertencente à transação



  - `WRITE(x)`: **grava** o item de dados **x** do buffer local da transação de volta para o BD

    - ✓ A operação **write(x)** não necessariamente resulta na atualização imediata dos dados no disco

Transação: ACID em ação

- **Cenário:**
  - Seja T_i uma transação que transfere R$ 50,00 da conta A (saldo R$ 1.000,00) para a conta B (saldo R$ 2.000,00)

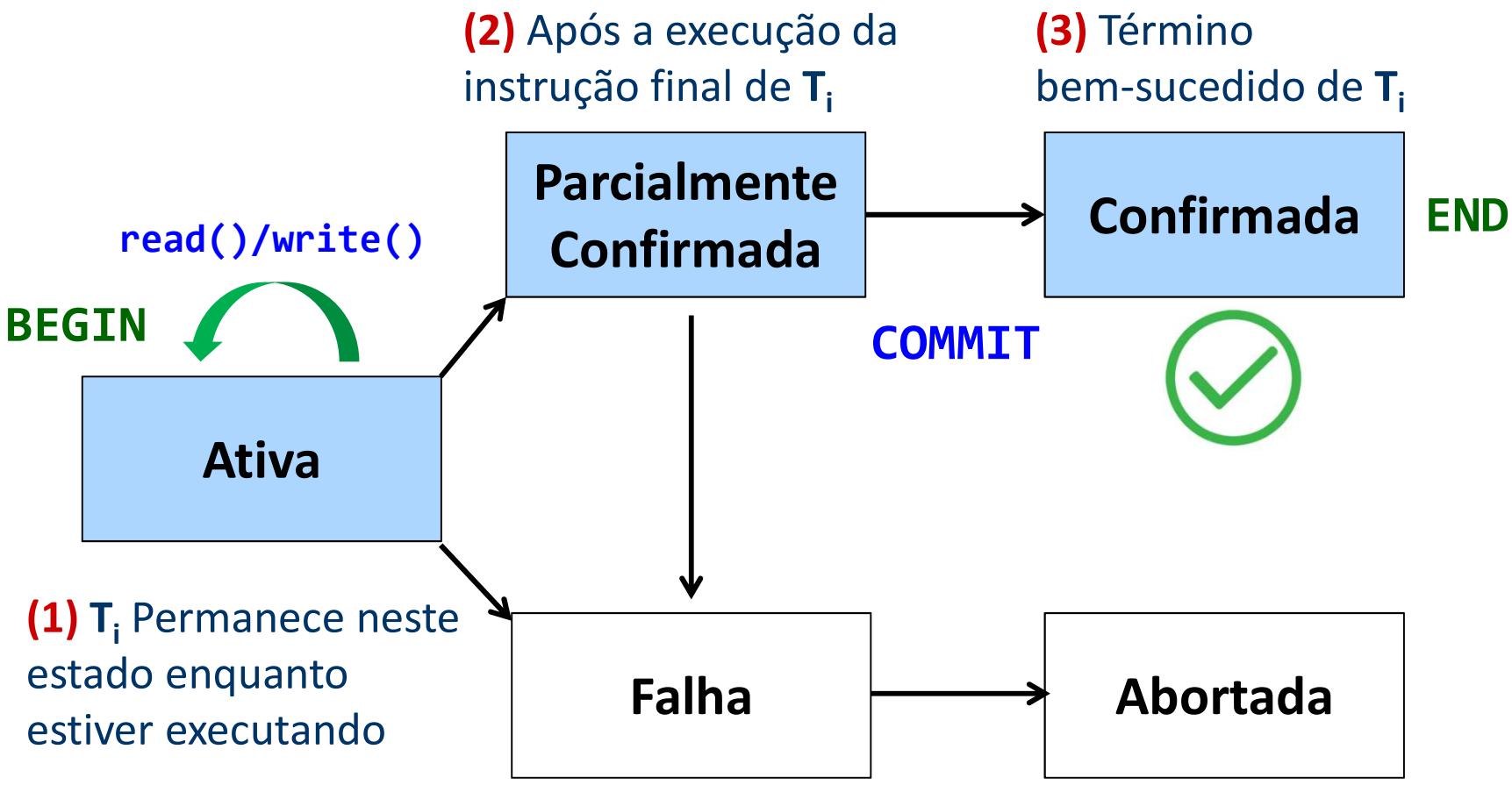
```
BEGIN TRANSACTION Ti
    read(A)
    A := A - 50.00
    write(A)
    read(B)
    B := B + 50.00
    write(B)
END TRANSACTION
```

Transação: ACID em ação

- **Atomicidade:**
  - Em caso de falha no transcorrer da transação, todas as operações deverão ser revertidas
  - Caso contrário, **todas** as operações são **efetivadas**
- **Consistência:**
  - A soma de **A** com **B** seja **inalterada** pela execução da transação
- **Isolamento**
  - Garantia de serialização de transações concorrentes
- **Durabilidade:**
  - T_i realizado com sucesso, todas as operações persistem no BD, mesmo em caso de falha (reconstrução das atualizações)

Transações

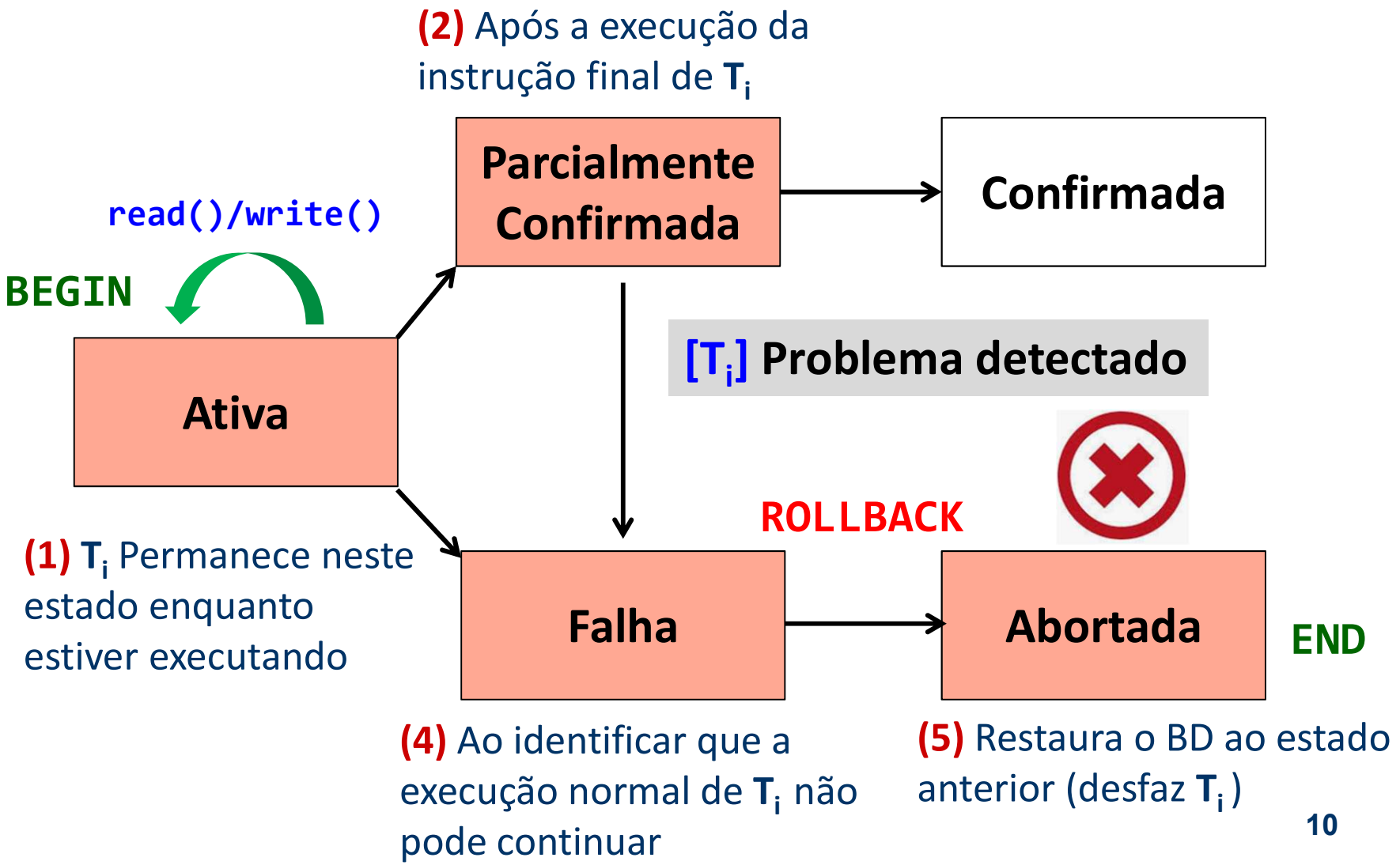
- **Diagrama de Estado**



[T_i] Ausência de falhas: dados garantidos no BD

Transações

- **Diagrama de Estado**



Transações

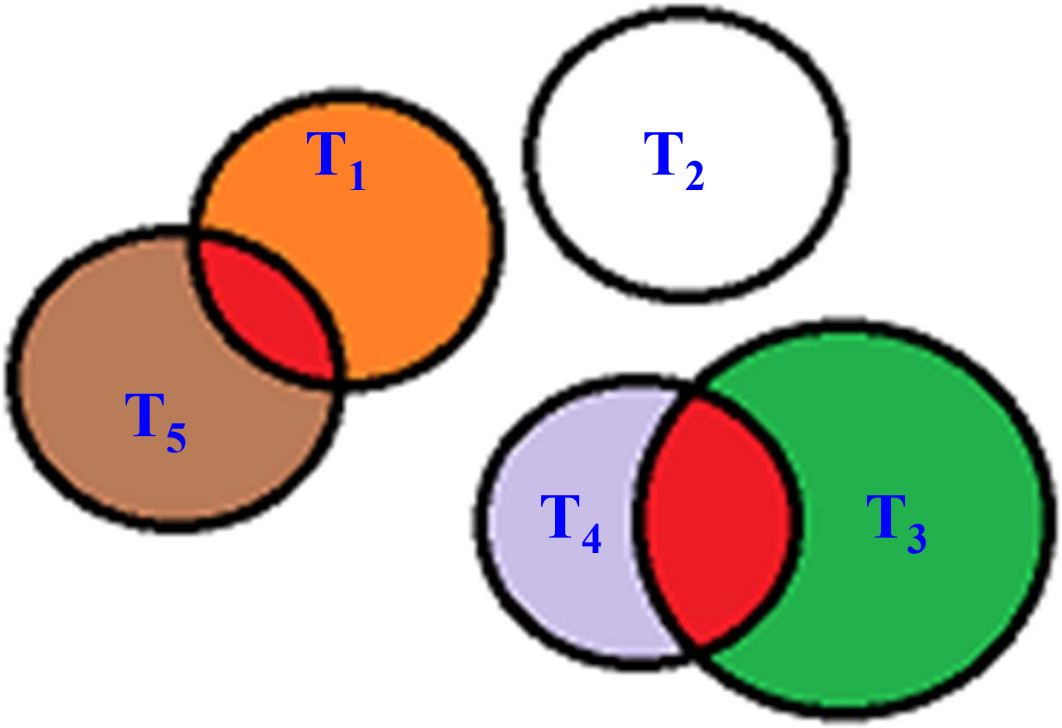
- Posso desfazer uma operação **confirmada?**
  - Não há como desfazer seus efeitos abortando-a

Solução:

- Executar uma transação de compensação
  - Se uma transação somou R$ 200.00 a uma conta, a **transação de compensação** subtrairia R$ 200.00 desta mesma conta
  - É de responsabilidade do **usuário** de escrever e executar a transação de compensação no BD
    - ✓ Nâo é tratada pelo SGBD

Execução Simultânea

- **Concorrência de transações**



Transações Concorrentes

- **Execução Simultânea**
  - Uma implementação desejada aos SGBDs
    - ✓ Melhor **throughput** e utilização de recursos
    - ✓ Tempo de espera reduzido
  - O controle de transações deve ser efetivo para evitar problemas de **inconsistência dos dados**
    - ✓ **SOLUÇÃO**: lidar com as transações concorrentes de forma que sejam executadas "**serialmente**"
  - **Schedule**
    - ✓ Nome dado à **ordem cronológica** em que as instruções da(s) transação(ões) são executadas no Sistema
    - ✓ A instrução corrente em execução pode alternar entre as transações partícipes da **schedule**

Transações Concorrentes

- **Exemplo: Schedule Serial**

T_1	S_1	T_2
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)

Seja T_1 uma transação que transfere R$ 50,00 da conta **A** (saldo R$ 1.000,00) para a conta **B** (saldo R$ 2.000,00) e uma transação T_2 transfere 10% do saldo da conta **A** para a conta **B**

(1) Qual o saldo final das contas **A** e **B**?

(2) A consistência dos dados foi preservada?

(3) E se invertermos a ordem para T_2 seguido de T_1?

A=$855 e B=$2.145

Transações Concorrentes

- **Exemplo: Schedule Intercalado**

T_1	S_2	T_2
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)

Seja T_1 uma transação que transfere R$ 50,00 da conta **A** (saldo R$ 1.000,00) para a conta **B** (saldo R$ 2.000,00) e uma transação T_2 transfere 10% do saldo da conta **A** para a conta **B**

(1) A sequência de instruções desta schedule preserva a consistência da soma A + B?

A=$855 e B=$2.145

Transações Concorrentes

- **Exemplo: Schedule Intercalado**

T_1	S_3	T_2
read(A)		
A:= A - 50		
write(A)		
		read(A)
		temp:= A * 0.1
		A:= A – temp
		write(A)
		read(B)
read (B)		
B:= B + 50		
write(B)		
		B:= B + temp
		write(B)

Seja T_1 uma transação que transfere R$ 50,00 da conta **A** (saldo R$ 1.000,00) para a conta **B** (saldo R$ 2.000,00) e uma transação T_2 transfere 10% do saldo da conta **A** para a conta **B**

(1) A sequência de instruções desta schedule preserva a consistência da soma A + B?

A=$950 e B=$2.100

Transações Concorrentes

- **Perguntas:**
  1. Qual o problema que ocasionou inconsistência na **schedule** anterior?
  2. Na sua opinião, como o problema poderia ser resolvido?

É tarefa do SGBD garantir que qualquer schedule executado deixe o banco de dados em um estado consistente!
Atribuição dada ao **Componente de Controle de Concorrência**

Controle de Concorrência

- Protocolo baseado em **Bloqueio**
  - Quando uma transação está acessando um item de dados, nehuma outra transação pode modificar esse item de dados
    - ✓ **Não** é um bloqueio de transação completa.
  - Modos
    - ✓ **COMPARTILHADO**: Se T_i obteve um bloqueio no modo compartilhado (denotado por **S**) sobre o item **P**, então T_i pode ler mas não pode escrever **P**

      `lock-s(P)`
    - ✓ **EXCLUSIVO**: Se T_i obteve um bloqueio no modo exclusivo (denotado por **X**) sobre o item **P**, então T_i pode ler e escrever **P**

      `lock-x(P)`

Controle de Concorrência

- Protocolo baseado em **Bloqueio**
  - Matriz de compatibilidade: **modos de bloqueio**

	S	X
S	**True**	False
X	False	False

  - O modo **S** é compatível com o modo **S**, mas não com o modo **X**
  - Para acessar um item de dados, a transação T_i primeiro precisa bloquear o item de dados **P**
    - ✓ Se o item de dado já estiver bloqueado por outra transação em um **modo incompativel**, o gerenciador de concorrência não concederá o bloqueio até que todos os bloqueios incompatíveis forem liberados
  - Uma transação pode desbloquear um item de dado

Transações Concorrentes

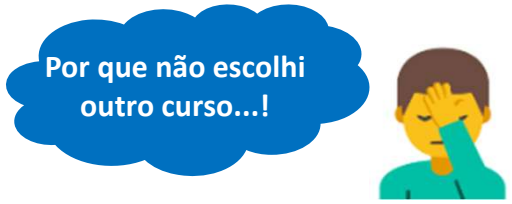
- **Exemplo: Schedule Intercalado**

T_1	S_4	T_2
lock-x(A)		
read(A)		
A:= A - 50		
write(A)		
unlock-x(A)		
		lock-S(A)
		read(A)
		unlock-s(A)
		lock-S(B)
		read(B)
		unlock-s(B)
		display(A+B)
lock-x(B)		
read (B)		
B:= B + 50		
write(B)		
unlock-x(B)		

Seja T_1 uma transação que transfere R$ 50,00 da conta **A** (saldo R$ 1000,00) para a conta **B** (saldo R$ 2000,00) e uma transação T_2 que exibe o total do saldo **A** + **B**

(1) A sequência de instruções desta schedule preserva a consistência da soma A + B?

`display(A+B)=$2950`



Transações Concorrentes

- **Exemplo: Schedule Intercalado**

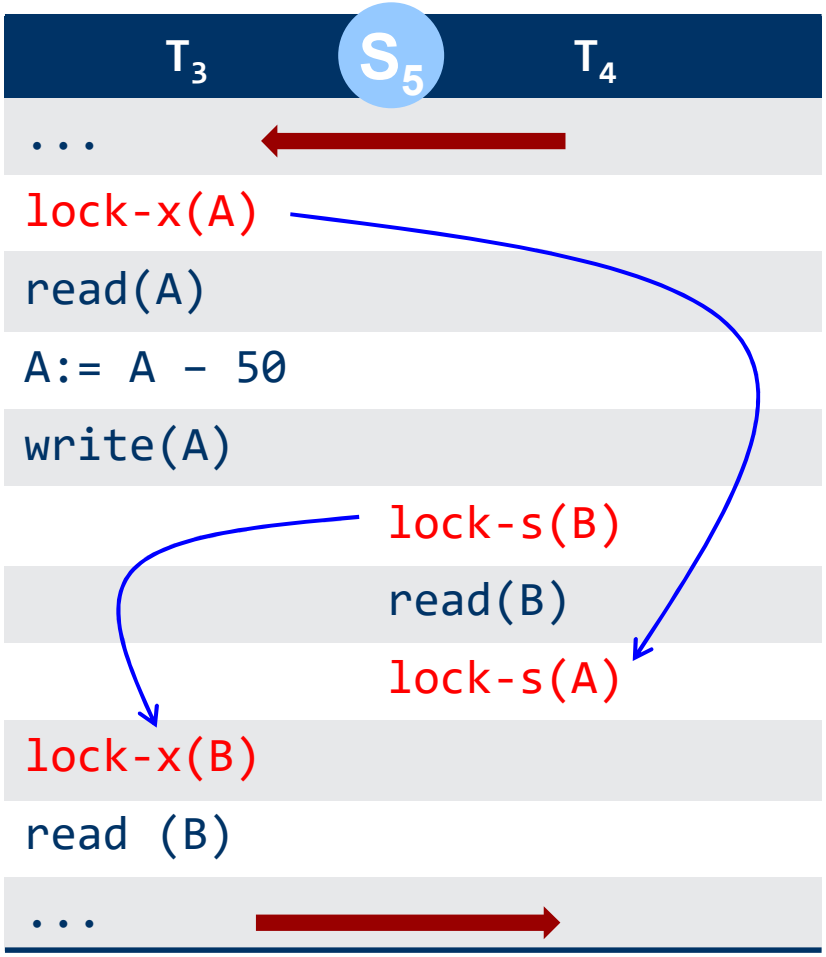
T_1	S_4	T_2
lock-x(A)		
read(A)		
A:= A - 50		
write(A)		
unlock-x(A)		
		lock-S(A)
		read(A)
		unlock-s(A)
		lock-S(B)
		read(B)
		unlock-s(B)
		display(A+B)
lock-x(B)		
read (B)		
B:= B + 50		
write(B)		
unlock-x(B)		

(1) Deduza o momento em que os bloqueios deverão ser concedidos

`display(A+B)=$3000`

Transações Concorrentes

- O Protocolo Baseado em **Bloqueio** pode levar a uma **situação indesejável**...



Schedule parcial T_3 e T_4

A = R$ 1000,00

B = R$ 2000,00

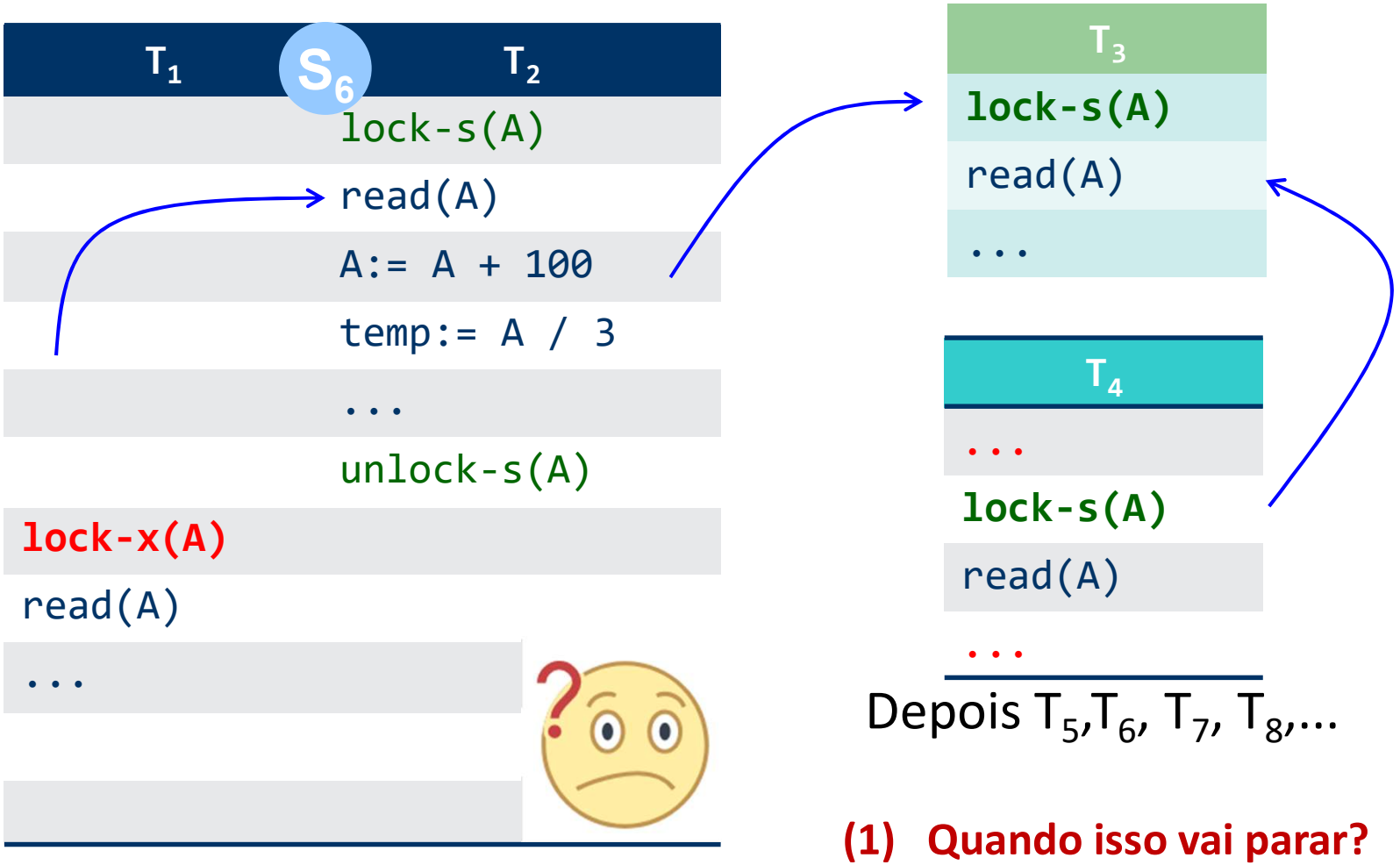
(1) A sequência de instruções desta schedule preserva a consistência da soma A + B?

Controle de Concorrência

- O problema do ***Deadlock*** **(impasse)**
  - Um **ciclo de dependência** entre duas ou mais transações
  - As transações envolvidas bloqueiam os recursos **permanentemente**, pois cada transação detém o bloqueio do recurso que a outra precisa
- Solução: **reverter uma das transações**
  - Se T_3 for revertida, os itens de dados que estavam bloqueados por essa transação serão desbloqueados
  - T_4 pode, então, continuar sua execução
- Os SGDBs implementam mecanismos próprios para evitar o **deadlock**

Transações Concorrentes

- Analisemos outro cenário...



(1) Quando isso vai parar?

Controle de Concorrência

- O problema ***Starvation*** **(inanição)**
  - Situação em que uma transação que requer acesso no modo exclusivo **aguarda indefinidamente** pelo desbloqueio de um item de dado que fôra bloqueado no modo conflitante
    - ✓ Outras transações concorrentes se apossam do item de dados por requererem bloqueio compatível ao mesmo item
- **Solução**: se T_1 solicita **lock-X(A),** o gerenciador concede o bloqueio desde que:
  - Não haja outra transação mantendo um bloqueio sobre **A** em modo conflitante
  - Não existe outra $\mathbf{T_j}$ que esteja esperando por um bloqueio sobre A e que fez sua solicitação de bloqueio antes de $\mathbf{T_j}$

Tolerância a Falhas

commit

rollback

Tolerância a Falhas

- **Classificação das Falhas**

Falha de Transação

- **Detectadas pelo próprio código** [Erro Lógico]
  - ✓ **Exemplo:** (1) saldo insuficiente para saque, (2) solicitar seguro-desemprego sem ter trabalhado o mínimo necessário para o benefício, (3) dados de entrada incorretos.

Falha de Sistema

- **Instabilidade no Sistema** [Erro do Sistema]
  - ✓ **Exemplo:** deadlock, starvation
- **Bugs no SGBD ou No Sistema Operacional**
  - ✓ **Exemplo:** access violation, system crash

Falha de Disco

- **Falha de hardware que não danifica o BD**
  - ✓ **Exemplo:** Memory Error
- **Falha em periféricos que danificam o BD**
  - ✓ **Exemplo:** Problema no disco rígido (**restore backup**)

Tolerância a Falhas

- **Recuperação baseada em LOG**
  - Estrutura mais utilizada para registrar a s modificações realizadas no banco de dados
  - O **LOG** armazena uma sequência de registros especiais que registram todas as atividades de atualização no banco de dados
  - Mantido em disco ou unidade de armazenamento secudário
  - O **LOG** sempre é consultado na ocorrência de falha de uma transação

No caso de **falha,** a transação pode ser **reiniciada** (erro de hardware ou software) ou **cancelada** (erro lógico interno, dados não encontrados no BD)

Tolerância a Falhas

- **O registro de LOG**
  - Um registro de log de atualização descreve uma única escrita no SGBD

Identificador exclusivo da transação que realizou um write

R_1 $\langle T_i,$ start$\rangle$ -- Ti foi iniciada

Identificador exclusivo do item de dados escrito

V_o: valor antigo **V_n: novo valor**

R_2 $\langle T_i,$ write, data, V_o, $V_n\rangle$
-- Ti escreveu sobre o item de dados "data". Possuía V_o
-- antes da escrita e o valor V_n depois da escrita

R_3 $\langle T_i,$ commit$\rangle$ -- A transação T_i foi confirmada

R_4 $\langle T_i,$ abort$\rangle$ -- A transação T_i foi abortada

Tolerância a Falhas

- **Recuperação baseada em LOG**
  - Todas as alterações realizadas no banco de dados **são gravadas primeiramente no LOG de transação**, antes que qualquer outra coisa possa ocorrer
  - A leitura/gravação no LOG é uma atribuição do modulo de recuperação do SGBD
  - Consideremos que cada registro de LOG é escrito **no final** do arquivo de LOG

(a) Por default, o PostgreSQL armazena seus logs de transações na pasta **pg_wal** do diretório de dados.
(b) Se o diretório **pg_wal** for preenchido e nenhum novo arquivo de log puder ser criado, o SGBD provavelmente paralisará suas atividades

Tolerância a Falhas

- **Checkpoint (**marcas de controle**)**
  - Em caso de falha no sistema, é preciso acessar o LOG para determinas as transações que precisam ser **refeitas** ou **desfeitas**
  - A princípio, o LOG inteiro deve ser pesquisado. **Dificuldades**:
    - ✓ Processo de busca é **demorado**
    - ✓ Recuperação consome **mais tempo**
  - O mecanismo de **checkpoint** reduz o tempo de "sobrecarga" para garantir a atomicidade
    - ✓ Pontos de verificação dentro do LOG
    - ✓ O percurso no LOG começa a partir do **checkpoint**

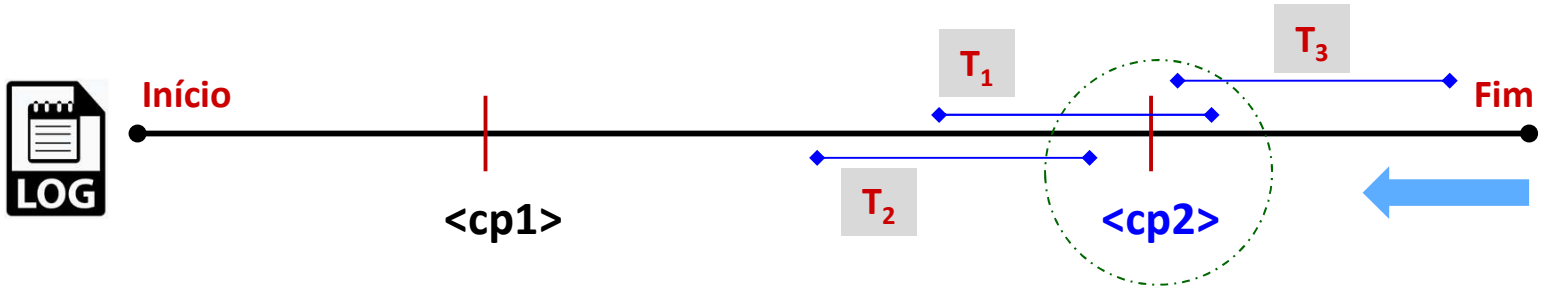
```
<checkpoint> -- Registro de checkpoint no LOG
```

Tolerância a Falhas

- **Checkpoint**
  - Os registros de **checkpoint** são adicionados periodicamente
  - O registro de **checkpoint** implica:
    - ✓ Força a gravação dos buffers do BD de $\mathbf{T_i}$ em disco
    - ✓ Adiciona um **registro de checkpoint** no LOG (disco)
    - ✓ Grava o endereço do **último checkpoint** em um arquivo de reinicialização (restart_log)
  - Transações (finalizadas ou não) podem aparecer antes ou depois do **checkpoint**
    - ✓ Ações específicas para cada uma das situações são realizadas

Tolerância a Falhas

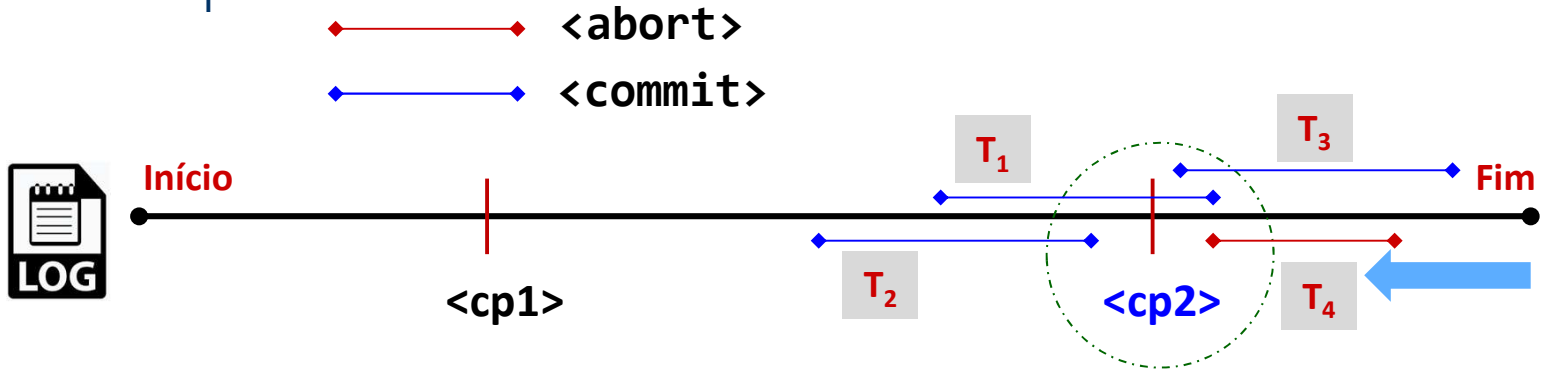
- **<checkpoint> após falha**
  1. Determinar a transação T_i mais recente no LOG que iniciou a execução antes que ocorreu o **checkpoint**
     - ✓ Do final para o início, em direção ao primeiro registro de **checkpoint**



  2. Encontrar o registro <T_i, start> mais recente
     - ✓ Iniciado **antes** do **checkpoint** <cp2>

Tolerância a Falhas

- **<checkpoint> após falha**
  3. Ao identificar a transação T_i, aplicar as operações **redo** e **undo** em T_i e transações que iniciaram depois de T_i



  - A parte anterior do LOG, relacionado ao checkpoint, pode ser **ignorada** (ou até mesmo apagada!)

Quais transações serão **refeitas** e quais serão **desfeitas**?

Tolerância a Falhas

As operações de recuperação exatas a serem realizadas dependem da **técnica de modificação** adotada

Algoritmo R.A.I.

- **Recuperação com Atualização Imediata**
  - As modificações (`write`) feitas por T_i são enviadas ao banco de dados enquanto a transação está no **estado ativo**.
    - ✓ As modificações são chamadas de **modificações não confirmadas**
  - LOG e BD, nesta ordem, são atualizados de imediato
    - ✓ Não se permite que uma T_i grave um dado no BD, até que pelo menos a parcela desfazer esteja no log.
  - No evento de falha, o sistema precisa guardar o valor anterior do item de dado modificado, no registro do LOG

Tolerância a Falhas

- **Recuperação com Atualização Imediata**
  - Antes de iniciar a execução de Ti, escreve o registro no LOG

Identificador exclusivo da transação que inicia a execução

R_1

```
<Ti, start> -- Ti foi iniciada
```

Identificador exclusivo do item de dados escrito

V_o: valor antigo **V_n: novo valor**

R_2

```
<Ti, write, data, Vo, Vn>
-- qualquer write é precidido pela entrada do registro no
-- LOG
```

R_3

```
<Ti, commit> -- A transação Ti foi PARCIALMENTE
             -- confirmada
```

Tolerância a Falhas

- **Algoritmo RAI com checkpoint**
  1. Localizar o **checkpoint** mais recente a partir do final do LOG
  2. Montar duas listas de transações:
     - ✓ **Transações Confirmadas**: <T_i, `commit`> escrito no LOG após o checkpoint
     - ✓ **Transações NÃO Confirmadas**: com <T_i, `start`> mas sem <T_i, `commit`> no LOG
  3. Desfazer **`writes`** das transações não confirmadas, restaurando o valor antigo: **`undo(x)`**
  4. Refazer todos os **`writes`** **`redo(x)`** das transações confirmadas, **na ordem em que foram escritas no LOG**

Tolerância a Falhas

- **Exemplo:**
  - Considere as transações T_0 e T_1 executadas uma após a outra (série). Valores iniciais das contas: A=$1000, B=$2000 e C = $700.

(a)	(b)	(c)
$<T_0$, start>	$<T_0$, start>	$<T_0$, start>
$<T_0$, A, 1000, 950>	$<T_0$, A, 1000, 950>	$<T_0$, A, 1000, 950>
$<T_0$, B, 2000, 2050>	$<T_0$, B, 2000, 2050>	$<T_0$, B, 2000, 2050>
	$<T_0$, commit>	$<T_0$, commit>
	$<T_1$, start>	$<T_1$, start>
	$<T_1$, C, 700, 600>	$<T_1$, C, 700, 600>
		$<T_1$, commit>

(1) Falha ocorre após $<T_0$, B, 2000, 2050> ser escrito no log. Qual o procedimento?

Tolerância a Falhas

- **Exemplo:**
  - Considere as transações T_0 e T_1 executadas uma após a outra (série). Valores iniciais das contas: A=$1000, B=$2000 e C = $700.

(a)	(b)	(c)
$<T_0$, start>	$<T_0$, start>	$<T_0$, start>
$<T_0$, A, 1000, 950>	$<T_0$, A, 1000, 950>	$<T_0$, A, 1000, 950>
$<T_0$, B, 2000, 2050>	$<T_0$, B, 2000, 2050>	$<T_0$, B, 2000, 2050>
	$<T_0$, commit>	$<T_0$, commit>
	$<T_1$, start>	$<T_1$, start>
	$<T_1$, C, 700, 600>	$<T_1$, C, 700, 600>
		$<T_1$, commit>

(2) Falha ocorre após $<T_1$, C, 700, 600> ser escrito no log. Qual o procedimento?

Tolerância a Falhas

- **Exemplo:**
  - Considere as transações T_0 e T_1 executadas uma após a outra (série). Valores iniciais das contas: A=$1000, B=$2000 e C = $700.

(a)	(b)	(c)
<T_0, start>	<T_0, start>	<T_0, start>
<T_0, A, 1000, 950>	<T_0, A, 1000, 950>	<T_0, A, 1000, 950>
<T_0, B, 2000, 2050>	<T_0, B, 2000, 2050>	<T_0, B, 2000, 2050>
	<T_0, commit>	<T_0, commit>
	<T_1, start>	<T_1, start>
	<T_1, C, 700, 600>	<T_1, C, 700, 600>
		<T_1, commit>

(3) Falha ocorre após <T_1, commit> ser escrito no log. Qual o procedimento?

Tolerância a Falhas

- **Conclusão:**
  - Algoritmo R.A.I. e Momento de Escrita no BD

Log	Banco de Dados
$<T_0$, start>	
$<T_0$, A, 1000, 950>	
	A = 950
$<T_0$, B, 2000, 2050>	
	B = 2050
$<T_0$, commit>	
$<T_1$, start>	
$<T_1$, C, 700, 600>	
	C = 600
$<T_1$, commit>	

Tolerância a Falhas

- **Recuperação com Atualização Adiada**
  - Antes de T_i iniciar sua execução, um registro **<T_i, start>** é escrito no LOG
  - Uma operação de escrita feita por (**write**) T_i resulta na inserção de um registro no LOG
  - Quando T_i é **confirmada parcialmente**, um registro **<T_i, commit>** é escrito no LOG.
  - Com T_i no estado de **confirmada,** os registros de T_i no LOG são usados para executar as **escritas adiadas** no banco de dados
  - Se o sistema "cair" antes de T_i completar sua execução (**commit**) ou se a transação **abortar**, então as informações no LOG serão **ignoradas**!

Tolerância a Falhas

- **Recuperação com Atualização Adiada**
  - Registros de controle R.A.A

Identificador exclusivo da transação que inicia a execução

R_1 `<Ti, start> -- Ti foi iniciada`

  - Uma operação **`write`** implica na escrita de um registro no LOG

V_n: novo valor

R_2 `<Ti, write, data, Vn>`

Não tem o valor antigo!

  - Se T_i é **parcialmente efetivada**:

R_3 `<Ti, commit> -- registro inserido no LOG`

    - ✓ Se T_i é efetivado, registros de T_i no LOG atualizam BD

Tolerância a Falhas

- **Algoritmo RAA com checkpoint**
  1. Localizar o **checkpoint** mais recente a partir do final do LOG
  2. Montar uma lista para as transações confirmadas: `<T_i, start> ... <T_i, commit>`
     - ✓ Registros de transações que iniciaram **após** o checkpoint
     - ✓ Registros de informações que iniciaram **antes** do checkpoint, **mas que não foram finalizadas** no momento do checkpoint
  3. Executar `redo()` dos itens de dados das transações confirmadas
     - ✓ **redo()** é **idempotente**: **redo(x)** = **redo**(**redo**(**redo**(x)))
  4. Registros de Transações incompletas podem ser excluídas do log

Tolerância a Falhas

- **Exemplo:**
  - Considere as transações T_0 e T_1 executadas uma após a outra. Valores iniciais das contas: A = $1000, B=$2000 e C = $700.

(a)	(b)	(c)
<T_0, start>	<T_0, start>	<T_0, start>
<T_0, A, 950>	<T_0, A, 1000, 950>	<T_0, A, 1000, 950>
<T_0, B, 2050>	<T_0, B, 2000, 2050>	<T_0, B, 2000, 2050>
	<T_0, commit>	<T_0, commit>
	<T_1, start>	<T_1, start>
	<T_1, C, 700, 600>	<T_1, C, 700, 600>
		<T_1, commit>

(1) Falha ocorre após <T_0, B, 2050> ser escrito no log. Qual o procedimento?

Tolerância a Falhas

- **Exemplo:**
  - Considere as transações T_0 e T_1 executadas uma após a outra. Valores iniciais das contas: A = $1000, B=$2000 e C = $700.

(a)	(b)	(c)
<T_0, start>	<T_0, start>	<T_0, start>
<T_0, A, 950>	<T_0, A, 950>	<T_0, A, 1000, 950>
<T_0, B, 2050>	<T_0, B, 2050>	<T_0, B, 2000, 2050>
	<T_0, commit>	<T_0, commit>
	<T_1, start>	<T_1, start>
	<T_1, C, 600>	<T_1, C, 700, 600>
		<T_1, commit>

(2) Falha ocorre após <T_1, C, 600> ser escrito no log. Qual o procedimento?

Tolerância a Falhas

- **Exemplo:**
  - Considere as transações T_0 e T_1 executadas uma após a outra. Valores iniciais das contas: A = $1000, B=$2000 e C = $700.

(a)	(b)	(c)
<T_0, start>	<T_0, start>	<T_0, start>
<T_0, A, 950>	<T_0, A, 950>	<T_0, A, 950>
<T_0, B, 2050>	<T_0, B, 2050>	<T_0, B, 2050>
	<T_0, commit>	<T_0, commit>
	<T_1, start>	<T_1, start>
	<T_1, C, 600>	<T_1, C, 600>
		<T_1, commit>

(3) Falha ocorre após <T_1, commit> ser escrito no log. Qual o procedimento?

Tolerância a Falhas

- **Conclusão:**
  - Algoritmo R.A.A. e Momento de Escrita no BD

Log	Banco de Dados
<T_0, start>	
<T_0, A, 950>	
<T_0, B, 2050>	
<T_0, commit>	
	A = 950
	B = 2050
<T_1, start>	
<T_1, C, 600>	
<T_1, commit>	
	C = 600

Tolerância a Falhas

- **Algoritmo RAI com checkpoint**
  - Provê maior concorrência

  - Necessidade de realizar **undo()**
  - Possibilidade de aborto em cascata

- **Algoritmo RAA com checkpoint**
  - Não necessita de **undo()**

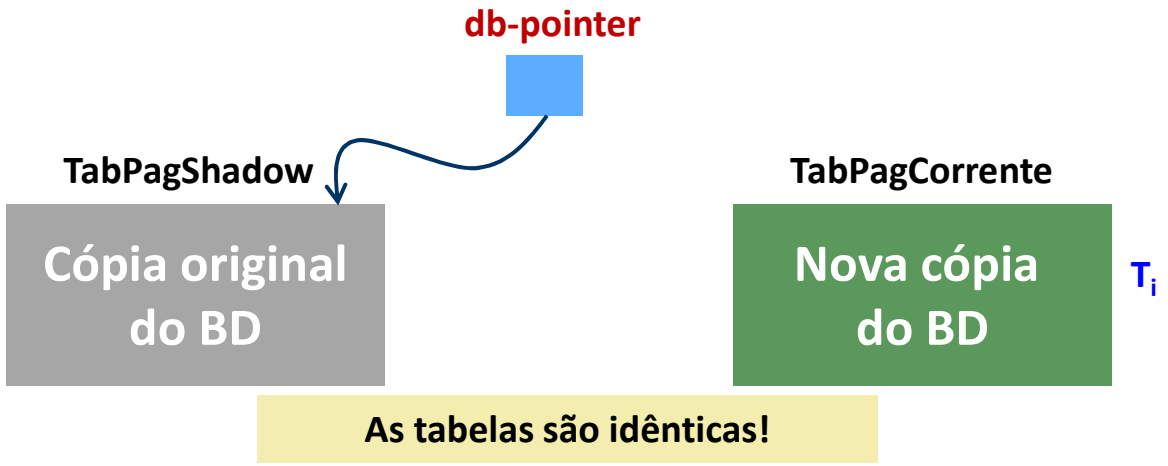
  - Limitação da concorrência de transações

Tolerância a Falhas

- **Paginação Sombra**
  - Outro componente do Sistema de Gerenciamento de Recuperação do BD para suporte à **atomicidade** e **durabilidade**
  - Consiste em realizar uma cópia do BD (cópia de **sombra**) e particioná-la em **páginas**
  - A **tabela de páginas** é constituída por **n** entradas, que aponta para a página **i** no disco rígido.
    - ✓ A **tabela de páginas** reside na memória RAM
  - A ideia básica é manter duas **tabelas de páginas**: a **atual** (TabPagCorrente) a **sombra** (TabPagShadow)
    - ✓ **Sombra**: disco rígido. **Atual**: RAM
  - Um ponteiro denominado db-pointer, mantido em disco, aponta para **cópia atual**

Tolerância a Falhas

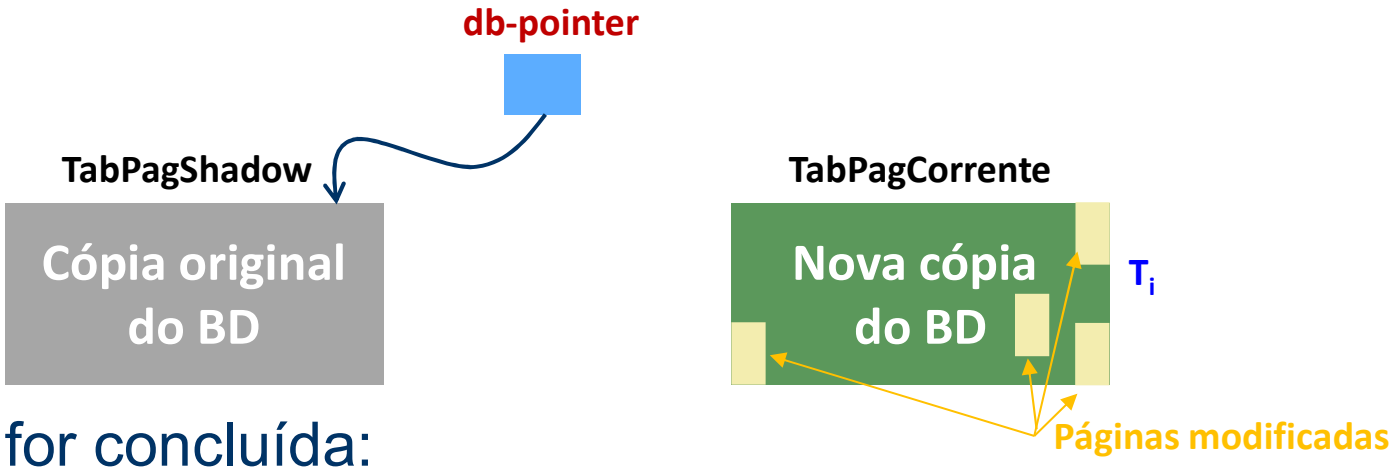
- **Paginação Sombra: Funcionamento**
  - Uma transação T_i que inicia sua execução, provoca a cópia da **TabPagCorrente** para **TabPagShadow**



  - Durante a execução, **TabPagShadow** não muda
    - ✓ Todas as atualizações são feitas sobre a nova cópia (**TabPagCorrente**)

Tolerância a Falhas

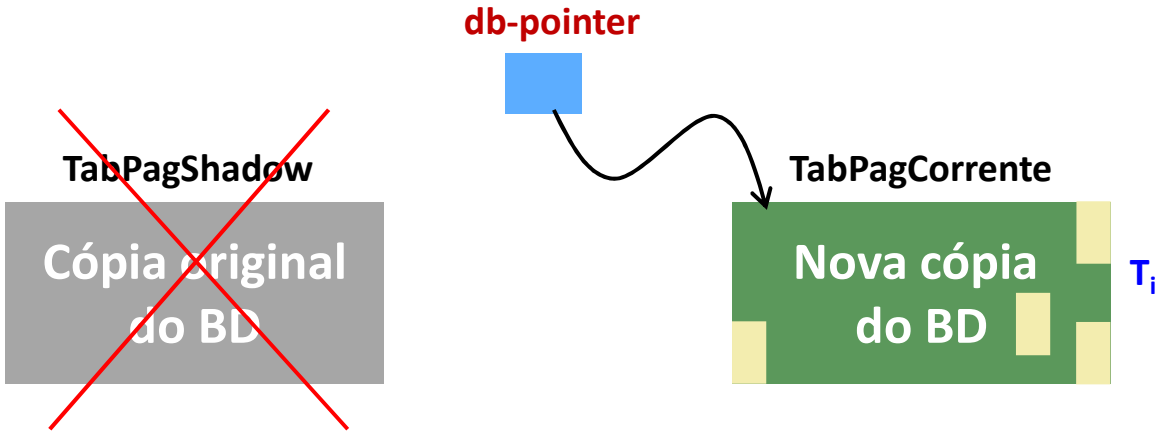
- **Paginação Sombra: Funcionamento**
  - Quando ocorrer um **`write`**, uma nova cópia da página modificada e passa a ser apontada por **TabPagCorrente**



  - Se T_i for concluída:
    - ✓ O S.O. confirma se as **novas páginas** foram gravadas em disco
    - ✓ O SGBD atualiza **db-pointer** para que aponte para a **nova cópia** do banco de dados

Tolerância a Falhas

- **Paginação Sombra: Funcionamento**
  - T_i concluída: a cópia antiga do banco de dados (shadow) é descartada



  - Em caso de **falha,** basta liberar as páginas modificadas e **descartar TabPagCorrente.** Assim, o Sistema se recupera usando a **shadow**

Tolerância a Falhas

- **Vantagens**
  - **`undo(x)` e `redo(x)` não são necessários**
  - Menos acessos ao disco que o método baseado em log (**não existe** LOG na paginação sombra!!!)
- **Desvantagens**
  - Mudanças de locações de páginas no disco: **complexo** gerenciador de armazenamento
  - *Overhead* por copiar a tabela de página de sombra para disco quando T_i confirma é maior se **TabPagCorrente** é grande
  - Dificil aplicação em transações concorrentes

Transações no PostgreSQL

- **Sintaxe**

```
BEGIN [TRANSACTION]; -- Iniciando a transação
-- comandos SQL da transação
COMMIT [TRANSACTION]; -- confirmar
```

  - Se tudo ocorrer bem, os dados serão efetivados no banco de dados após a execução do COMMIT
  - A instrução ROLLBACK desfaz as modificações realizadas pela transação

```
BEGIN [TRANSACTION];
  -- comandos SQL da transação
ROLLBACK [TRANSACTION]; -- reverter modificações
```

  - A estrutura com **BEGIN; ... COMMIT/ROLLBACK;** não aceita estruturas de controle de fluxo em seu interior

Transações no PostgreSQL

- **Considerações**
  - Cada operação SQL de modificação isolado é tratado pelo PostgreSQL como uma transação diferente

T_1 / T_2

```
UPDATE produto SET estatual = estatual + 8
WHERE codProd = 15;                                    1

DELETE FROM itensPedido WHERE codPed = 25
AND codProd = 15;                                      2
```

BEGIN implícito e COMMIT efetivado!

T_1

```
BEGIN;
   UPDATE produto SET estatual = estatual + 8
   WHERE codProd = 15;

   DELETE FROM itensPedido WHERE codPed = 25
   AND codProd = 15;
COMMIT;
```

Transações no PostgreSQL

- **Exemplo:** transferir $100 da conta **101** para a conta **505**

```
BEGIN;
UPDATE conta SET saldo = saldo – 100
WHERE numConta = 101;

UPDATE conta SET saldo = saldo + 100
WHERE numConta = 505;

COMMIT;
```

Até aqui, as UPDATES executados não são visíveis entre Sessões distintas de acesso ao BD

Transações no PostgreSQL

- **Savepoint**
  - Instrução que possibilita o descartar parte da transação, enquanto o resto recebe COMMIT
  - Funciona como um ponto de controle para reverter alterações efetuadas depois do registro de SAVEPOINT
    - ✓ Savepoint's são rotulados dentro do código

```
BEGIN;
UPDATE produto SET estatual = estatual + 8
WHERE codProd = 15;
SAVEPOINT spoint1;
DELETE FROM itensPedido WHERE codPed = 25
AND codProd = 15;
ROLLBACK TO spoint1;
INSERT INTO produto VALUES (33, 'lapis',0);
COMMIT;
```

Transações no PostgreSQL

- **Manejo de Transações – Stored Procedure**
  - Tanto em **stored procedures** quando em **blocos anônimos** (DO $$), é possível finalizar transações utilizando os comandos COMMIT ou ROLLBACK
  - O próprio BEGIN da estrutura da **stored procedure** realiza o *start* da transação

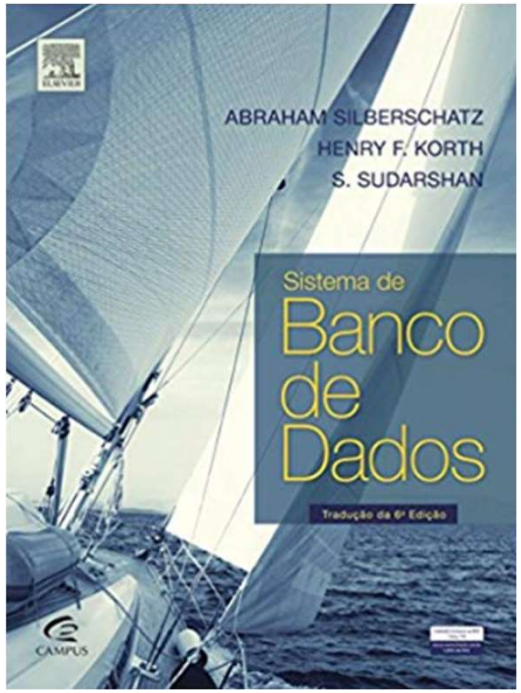
```
CREATE PROCEDURE teste_transação()
LANGUAGE plpgsql;
AS $$
BEGIN
   FOR i IN 1..5 LOOP
      INSERT INTO conta VALUES (i,‘X’,0);
      IF i%2 = 0 THEN
          COMMIT;
      ELSE
          ROLLBACK;
      END IF
   END LOOP;
END $$;
```

Transações no PostgreSQL

- **Considerações**
  - Tente manter **transações curtas**
    - ✓ Não acrescente muitas instruções SQL entre o **BEGIN** e o **COMMIT**
  - Tratamento de exeções podem ser previstas dentro do bloco de código da transação, quando definidos dentro de uma stored procedure
    - ✓ Um **RAISE EXCEPTION** gerando dentro do bloco da transação causará um **ROLLBACK** automático
  - A lógica do negócio escrita pela transação é quem vai definir o momento de alcançar um COMMIT ou um ROLLBACK
  - Instruções de uma transação em andamento NÃO são visíveis em sessões diferentes de acesso ao BD

Referências Bibliográficas

- Silberchatz, A.; Korth, H.; Sudarshan, S. **Sistemas de Banco de Dados.** Campus: Rio de Janeiro, 5ª edição, 2006.



Referências Bibliográficas

- PostgreSQL Transaction. Disponível em: http://www.postgresqltutorial.com/postgresql-transaction/
- PostgreSQL Transaction (TutorialsPoint). https://www.tutorialspoint.com/postgresql/postgresql_transactions.htm
- Transactions (PostgreSQL documentation). https://www.postgresql.org/docs/8.3/tutorial-transactions.html
- Transaction Management (PostgreSQL documentation). https://www.postgresql.org/docs/11/plpgsql-transactions.html