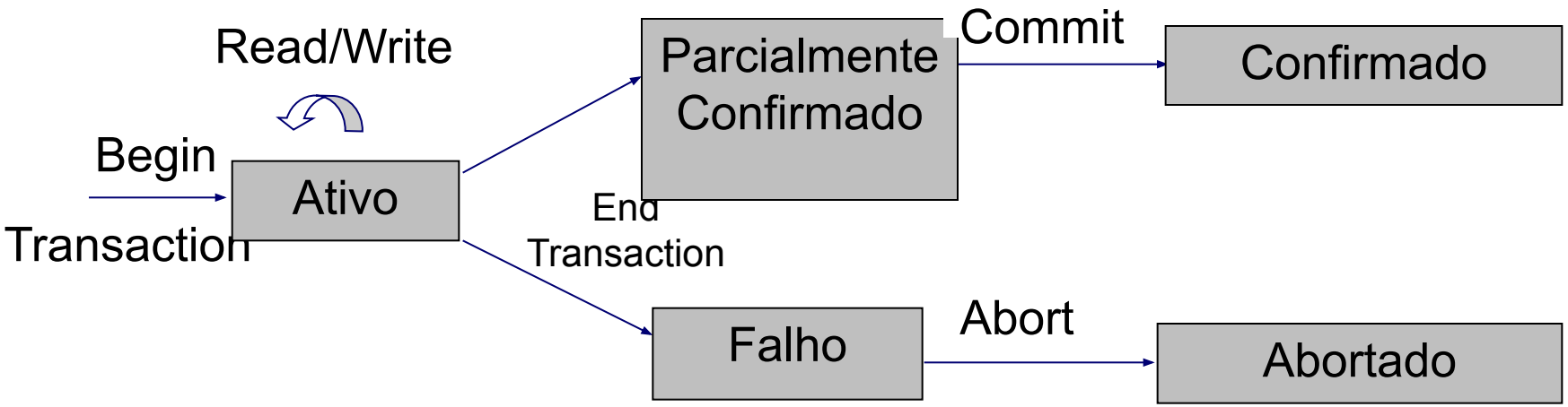
Universidade Federal
de Campina Grande

Banco de Dados I
Unidade 11: Recuperação de Falhas

Prof. Cláudio de Souza Baptista, Ph.D.
Laboratório de Sistemas de Informação – LSI
UFCG

Introdução

Diagrama de estados de uma transação



Classificação das Falhas

- **Tipos de falhas:**
  1. Falha na transição detectada pelo próprio código
     Ex.: "Fundos insuficientes"
  2. Falha na transição não prevista no código
     Ex.: Overflow, divisão por zero
  3. Falha no sistema que não danifica o BD
     Ex.: System crash, deadlock
  4. Falha nos periféricos que danifica o BD
     Ex.: Falha no disco, cabeçote
- **Quando uma Ti aborta:**
  - Pode ser reiniciada;
  - Pode ser abortada

Classificação das Falhas

- **Tipos de falhas:**
  1. Falha na transação detectada pelo próprio código
     Ex.: "Fundos insuficientes"
  2. Falha na transição não prevista no código
     Ex.: Overflow, divisão por zero
  3. Falha no sistema que não danifica o BD
     Ex.: System crash, deadlock
  4. Falha nos periféricos que danifica o BD
     Ex.: Falha no disco, cabeçote
- **Quando uma Ti aborta:**
  - Pode ser reiniciada;
  - Pode ser abortada

Classificação das Falhas

- **Tipos de falhas:**
  1. Falha na transição detectada pelo próprio código
     Ex.: "Fundos insuficientes"
  2. Falha na transição não prevista no código
     Ex.: Overflow, divisão por zero
  3. Falha no sistema que não danifica o BD
     Ex.: System crash, deadlock
  4. Falha nos periféricos que danifica o BD
     Ex.: Falha no disco, cabeçote
- **Quando uma Ti aborta:**
  - Pode ser reiniciada;
  - Pode ser abortada

Classificação das Falhas

- **Tipos de falhas:**
  1. Falha na transição detectada pelo próprio código
     Ex.: "Fundos insuficientes"
  2. Falha na transação não prevista no código
     Ex.: Overflow, divisão por zero
  3. Falha no sistema que não danifica o BD
     Ex.: System crash, deadlock
  4. Falha nos periféricos que danifica o BD
     Ex.: Falha no disco, cabeçote
- **Quando uma Ti aborta:**
  - Pode ser reiniciada;
  - Pode ser abortada

Classificação das Falhas

- **Tipos de falhas:**
  1. Falha na transação detectada pelo próprio código
     Ex.: "Fundos insuficientes"
  2. Falha na transação não prevista no código
     Ex.: Overflow, divisão por zero
  3. Falha no sistema que não danifica o BD
     Ex.: System crash, deadlock
  4. Falha nos periféricos que danifica o BD
     Ex.: Falha no disco, cabeçote
- **Quando uma Ti aborta:**
  - Pode ser reiniciada;
  - Pode ser abortada

Classificação das Falhas

- **Tipos de falhas:**
  1. Falha na transação detectada pelo próprio código
     Ex.: "Fundos insuficientes"
  2. Falha na transação não prevista no código
     Ex.: Overflow, divisão por zero
  3. Falha no sistema que não danifica o BD
     Ex.: System crash, deadlock
  4. Falha nos periféricos que danifica o BD
     Ex.: Falha no disco, cabeçote
- **Quando uma Ti aborta:**
  - Pode ser reiniciada;
  - Pode ser abortada

Classificação das Falhas

- **Tipos de falhas:**
  1. Falha na transação detectada pelo próprio código
     Ex.: "Fundos insuficientes"
  2. Falha na transação não prevista no código
     Ex.: Overflow, divisão por zero
  3. Falha no sistema que não danifica o BD
     Ex.: System crash, deadlock
  4. Falha nos periféricos que danifica o BD
     Ex.: Falha no disco, cabeçote
- **Quando uma Ti aborta:**
  - Pode ser reiniciada;
  - Pode ser abortada

Classificação das Falhas

- **Tipos de falhas:**
  1. Falha na transação detectada pelo próprio código
     Ex.: "Fundos insuficientes"
  2. Falha na transação não prevista no código
     Ex.: Overflow, divisão por zero
  3. Falha no sistema que não danifica o BD
     Ex.: System crash, deadlock
  4. Falha nos periféricos que danifica o BD
     Ex.: Falha no disco, cabeçote
- **Quando uma Ti aborta:**
  - Pode ser reiniciada;
  - Pode ser abortada

Classificação das Falhas

- **Tipos de falhas:**
  1. Falha na transação detectada pelo próprio código
     Ex.: "Fundos insuficientes"
  2. Falha na transação não prevista no código
     Ex.: Overflow, divisão por zero
  3. Falha no sistema que não danifica o BD
     Ex.: System crash, deadlock
  4. Falha nos periféricos que danifica o BD
     Ex.: Falha no disco, cabeçote
- **Quando uma Ti aborta:**
  - Pode ser reiniciada;
  - Pode ser abortada

Classificação das Falhas

- **Tipos de falhas:**
  1. Falha na transação detectada pelo próprio código
     Ex.: "Fundos insuficientes"
  2. Falha na transação não prevista no código
     Ex.: Overflow, divisão por zero
  3. Falha no sistema que não danifica o BD
     Ex.: System crash, deadlock
  4. Falha nos periféricos que danifica o BD
     Ex.: Falha no disco, cabeçote
- **Quando uma Ti aborta:**
  - Pode ser reiniciada;
  - Pode ser abortada

Classificação das Falhas

- **Tipos de falhas:**
  1. Falha na transação detectada pelo próprio código
     Ex.: "Fundos insuficientes"
  2. Falha na transação não prevista no código
     Ex.: Overflow, divisão por zero
  3. Falha no sistema que não danifica o BD
     Ex.: System crash, deadlock
  4. Falha nos periféricos que danifica o BD
     Ex.: Falha no disco, cabeçote
- **Quando uma Ti aborta:**
  - Pode ser reiniciada;
  - Pode ser abortada

Classificação das Falhas

- **Tipos de falhas:**
  1. Falha na transação detectada pelo próprio código
     Ex.: "Fundos insuficientes"
  2. Falha na transação não prevista no código
     Ex.: Overflow, divisão por zero
  3. Falha no sistema que não danifica o BD
     Ex.: System crash, deadlock
  4. Falha nos periféricos que danifica o BD
     Ex.: Falha no disco, cabeçote
- **Quando uma Ti aborta:**
  - Pode ser reiniciada;
  - Pode ser abortada

Classificação das Falhas

- **Tipos de falhas:**
  1. Falha na transação detectada pelo próprio código
     Ex.: "Fundos insuficientes"
  2. Falha na transação não prevista no código
     Ex.: Overflow, divisão por zero
  3. Falha no sistema que não danifica o BD
     Ex.: System crash, deadlock
  4. Falha nos periféricos que danifica o BD
     Ex.: Falha no disco, cabeçote
- **Quando uma Ti aborta:**
  - Pode ser reiniciada;
  - Pode ser abortada

Log (Journal, History)

- É um arquivo que mantém todas as operações que as transações efetuaram no BD;
- É usado para recuperar o sistema na ocorrência de falhas;
- É mantido em disco e em fita (backup).

Log (Journal, History)

- É um arquivo que mantém todas as operações que as transações efetuaram no BD;
- É usado para recuperar o sistema na ocorrência de falhas;
- É mantido em disco e em fita (backup).

Log (Journal, History)

- É um arquivo que mantém todas as operações que as transações efetuaram no BD;
- É usado para recuperar o sistema na ocorrência de falhas;
- É mantido em disco e em fita (backup).

Log (Journal, History)

- Tipos de registro gravados no log:
  1. **[Start_Transaction, T1] :** T1 inicia a execução;
  1. **[Write (X), T1, ValorAntigo, ValorNovo]**
  1. **[Read (X), T1]**
  1. **[Commit, T1]**

Obs.: Como veremos a seguir, alguns protocolos de recuperação não requerem que **[Read (X), T]** seja gravado no log => menor overhead!

Checkpoint

- **Motivação:**
  - Quando ocorrer uma falha, o sistema não precisa pesquisar todo o log, nem refazer transações que não precisem de **REDO**
  - Diminuir overhead de recuperação

- **Solução:** Checkpoints

- Periodicamente, um registro de checkpoint é gravado no log

Checkpoint

- Cada gravação requer as seguintes ações:
  1. Forçar a gravação do LOG em disco
  1. Gravar os buffers do BD em disco
  1. Gravar um registro de checkpoint no LOG
  1. Gravar o endereço do último checkpoint num arquivo de restart.

Ex.: Seja um conjunto de **Tis** $\{T_0, T_1, \ldots, T_{100}\}$ executando na ordem dos índices. Suponha que ocorreu um checkpoint (último) durante a transação $\mathbf{T_{68}}$.

Recuperar {Ti / 68 <= i <= 100}

Checkpoint

- Cada gravação requer as seguintes ações:
  1. Forçar a gravação do LOG em disco
  1. Gravar os buffers do BD em disco
  1. Gravar um registro de checkpoint no LOG
  1. Gravar o endereço do último checkpoint num arquivo de restart.

Ex.: Seja um conjunto de **Tis** $\{T_0, T_1, \ldots, T_{100}\}$ executando na ordem dos índices. Suponha que ocorreu um checkpoint (último) durante a transação $\mathbf{T_{68}}$.

Recuperar {Ti / 68 <= i <= 100}

Checkpoint

- Cada gravação requer as seguintes ações:
  1. Forçar a gravação do LOG em disco
  1. Gravar os buffers do BD em disco
  1. Gravar um registro de checkpoint no LOG
  1. Gravar o endereço do último checkpoint num arquivo de restart.

Ex.: Seja um conjunto de **Tis** $\{T_0, T_1, \ldots, T_{100}\}$ executando na ordem dos índices. Suponha que ocorreu um checkpoint (último) durante a transação $\mathbf{T_{68}}$.

Recuperar {Ti / 68 <= i <= 100}

Checkpoint

- Cada gravação requer as seguintes ações:
  1. Forçar a gravação do LOG em disco
  1. Gravar os buffers do BD em disco
  1. Gravar um registro de checkpoint no LOG
  1. Gravar o endereço do último checkpoint num arquivo de restart.

Ex.: Seja um conjunto de **Tis** $\{T_0, T_1, \ldots, T_{100}\}$ executando na ordem dos índices. Suponha que ocorreu um checkpoint (último) durante a transação $\mathbf{T_{68}}$.

Recuperar {Ti / 68 <= i <= 100}

Checkpoint

- Cada gravação requer as seguintes ações:
  1. Forçar a gravação do LOG em disco
  1. Gravar os buffers do BD em disco
  1. Gravar um registro de checkpoint no LOG
  1. Gravar o endereço do último checkpoint num arquivo de restart.

Ex.: Seja um conjunto de **Tis** $\{T_0, T_1, \ldots, T_{100}\}$ executando na ordem dos índices. Suponha que ocorreu um checkpoint (último) durante a transação $\mathbf{T_{68}}$.

Recuperar {Ti / 68 <= i <= 100}

Checkpoint

- Cada gravação requer as seguintes ações:
  1. Forçar a gravação do LOG em disco
  1. Gravar os buffers do BD em disco
  1. Gravar um registro de checkpoint no LOG
  1. Gravar o endereço do último checkpoint num arquivo de restart.

Ex.: Seja um conjunto de **Tis** $\{T_0, T_1, \ldots, T_{100}\}$ executando na ordem dos índices. Suponha que ocorreu um checkpoint (último) durante a transação $\mathbf{T_{68}}$.

Recuperar {Ti / 68 <= i <= 100}

Checkpoint

- Cada gravação requer as seguintes ações:
  1. Forçar a gravação do LOG em disco
  1. Gravar os buffers do BD em disco
  1. Gravar um registro de checkpoint no LOG
  1. Gravar o endereço do último checkpoint num arquivo de restart.

Ex.: Seja um conjunto de **Tis** $\{T_0, T_1, \ldots, T_{100}\}$ executando na ordem dos índices. Suponha que ocorreu um checkpoint (último) durante a transação $\mathbf{T_{68}}$.

Recuperar {Ti / 68 <= i <= 100}

Recuperação com Atualizações Adiadas

- **Motivação:**
  - Considere a seguinte transação:

    T = R(X); W(X); R(Y); W(Y)
  - Valores dos objetos antes: **X** = 100, **Y** = 200
  - Transação: transferir 50 da conta **X** para a conta **Y**
  - Problema: falha ocorre após **W(X)**
  - Possíveis procedimentos de recuperação:
    - Reexecutar **T** → **X = 0, Y = 250**
    - Não reexecutar **T** → **X = 50, Y = 200**
  - Modificou-se o BD sem a garantia que T iria confirmar.

Recuperação com Atualizações Adiadas

- **Solução:**
  - Recuperar com Atualizações Adiadas.
- Todos os **Writes** serão atrasados até que **Ti** confirme parcialmente;
- Todos os **Writes** serão gravados num arquivo de log;
- Quando **commit** ocorre → BD é atualizado.

Recuperação com Atualizações Adiadas

- **Log:** mantém todas as atualizações do BD. Entradas no log para cada **id,transaction**.

1. <Ti, início>
1. < Ti, W, X, Valor.novo >
1. < Ti, commit >

- Usando o log, o sistema pode resolver qualquer falha que não danifique o BD físico.

Recuperação com Atualizações Adiadas

- **Algoritmo RAA (com checkpoint)**
  1. Iniciar da última entrada do log, voltando em direção ao checkpoint mais recente ;
  2. Montar uma lista de Tis_Confirmadas → possuem no log, <Ti, Início> e <Ti, commit>;
  3. Refazer (**REDO**) todos os writes confirmados na ordem em que foram escritos no log.

- **REDO (Ti)**: determina o valor de todos os objetos atualizados por Ti para novos valores.

  REDO (X) = REDO (REDO (REDO (X)))

  ↓

  Idempotente

Recuperação com Atualizações Adiadas

- **Algoritmo RAA (com checkpoint)**
  1. Iniciar da última entrada do log, voltando em direção ao checkpoint mais recente ;
  2. Montar uma lista de Tis_Confirmadas → possuem no log, <Ti, Início> e <Ti, commit>;
  3. Refazer (**REDO**) todos os writes confirmados na ordem em que foram escritos no log.

- **REDO (Ti)**: determina o valor de todos os objetos atualizados por Ti para novos valores.

  REDO (X) = REDO (REDO (REDO (X)))

  ↓

  Idempotente

Recuperação com Atualizações Adiadas

- **Algoritmo RAA (com checkpoint)**
  1. Iniciar da última entrada do log, voltando em direção ao checkpoint mais recente ;
  2. Montar uma lista de Tis_Confirmadas → possuem no log, <Ti, Início> e <Ti, commit>;
  3. Refazer (**REDO**) todos os writes confirmados na ordem em que foram escritos no log.

- **REDO (Ti)**: determina o valor de todos os objetos atualizados por Ti para novos valores.

  REDO (X) = REDO (REDO (REDO (X)))

  ↓

  Idempotente

Recuperação com Atualizações Adiadas

- **Algoritmo RAA (com checkpoint)**
  1. Iniciar da última entrada do log, voltando em direção ao checkpoint mais recente ;
  2. Montar uma lista de Tis_Confirmadas → possuem no log, <Ti, Início> e <Ti, commit>;
  3. Refazer (**REDO**) todos os writes confirmados na ordem em que foram escritos no log.

- **REDO (Ti)**: determina o valor de todos os objetos atualizados por Ti para novos valores.

  REDO (X) = REDO (REDO (REDO (X)))

  ↓

  Idempotente

Recuperação com Atualizações Adiadas

- **Algoritmo RAA (com checkpoint)**
  1. Iniciar da última entrada do log, voltando em direção ao checkpoint mais recente ;
  2. Montar uma lista de Tis_Confirmadas → possuem no log, <Ti, Início> e <Ti, commit>;
  3. Refazer (**REDO**) todos os writes confirmados na ordem em que foram escritos no log.

- **REDO (Ti)**: determina o valor de todos os objetos atualizados por Ti para novos valores.

  REDO (X) = REDO (REDO (REDO (X)))

  ↓

  Idempotente

Recuperação com Atualizações Adiadas

- **Algoritmo RAA (com checkpoint)**
  1. Iniciar da última entrada do log, voltando em direção ao checkpoint mais recente ;
  2. Montar uma lista de Tis_Confirmadas → possuem no log, <Ti, Início> e <Ti, commit>;
  3. Refazer (**REDO**) todos os writes confirmados na ordem em que foram escritos no log.

- **REDO (Ti)**: determina o valor de todos os objetos atualizados por Ti para novos valores.

  REDO (X) = REDO (REDO (REDO (X)))

  ↓

  Idempotente

Recuperação com Atualizações Adiadas

Exemplo: Sejam duas transações

T1: transferência de fundos de $50 da conta **A** para **B**

T2 : saque de $100 da conta **C**

Implementadas como segue:

T1: Begin_Transaction_T1	T1: Begin_Transaction_T2
Read(A, a1);	Read(C, c1);
a1 := a1 – 50;	c1 := c1 – 100;
Write (A, a1);	Write (C, c1);
Read (B, b1);	End_Transaction_T2;
b1 := b1 + 50;	
Write (B, b1);	
End_Transaction_T1;	

Valores inicias das contas: A=$1000, B=$2000, C=$700.

Suponha que T2 é executada depois de T1.

Recuperação com Atualizações Adiadas

Log para execução completa de **T1** e **T2**:

< T1 , Início >

< T1 , A, 950 >

< T1 , B, 2050 >

< T2 , Início >

< T1 , commit >

→ A = 950 e B = 2050

< T2 , C, 600 >

< T2 , commit >

→ C = 600

Recuperação com Atualizações Adiadas

Vamos simular alguns casos de falhas:

1) Falha ocorre após **< T1, B, 2050 >** ser gravado no log. O log na hora da falha é:

< T1 , Início >

< T1 , A, 950 >

< T1 , B, 2050 >

- No restart não precisa recuperação, pois não há commit.

Recuperação com Atualizações Adiadas

Vamos simular alguns casos de falhas:

1) Falha ocorre após **< T1, B, 2050 >** ser gravado no log. O log na hora da falha é:

< T1 , Início >

< T1 , A, 950 >

< T1 , B, 2050 >

- No restart não precisa recuperação, pois não há commit.

Recuperação com Atualizações Adiadas

2) Falha ocorre após **< T2, C, 600 >** ser gravado no log. Log na hora da falha:

< T1 , Início >

< T1 , A, 950 >

< T1 , B, 2050 >

< T1 , commit >

< T2 , Início >

< T2 , C, 600 >

- Quando restart ocorre → **REDO (T1)**

Recuperação com Atualizações Adiadas

2) Falha ocorre após **< T2, C, 600 >** ser gravado no log. Log na hora da falha:

< T1 , Início >

< T1 , A, 950 >

< T1 , B, 2050 >

< T1 , commit >

< T2 , Início >

< T2 , C, 600 >

- Quando restart ocorre → **REDO (T1)**

Recuperação com Atualizações Adiadas

3) Falha ocorre após **< T2, commit >** ser gravado no log. Log na hora da falha:

< T1 , Início >

< T1 , A, 950 >

< T1 , B, 2050 >

< T1 , commit >

< T2 , Início >

< T2 , C, 600 >

< T2 , commit >

- Quando restart ocorre → **REDO (T1)** e **REDO(T2)**

Recuperação com Atualizações Adiadas

3) Falha ocorre após **< T2, commit >** ser gravado no log. Log na hora da falha:

< T1 , Início >

< T1 , A, 950 >

< T1 , B, 2050 >

< T1 , commit >

< T2 , Início >

< T2 , C, 600 >

< T2 , commit >

- Quando restart ocorre → **REDO (T1)** e **REDO(T2)**

Recuperação com Atualizações Adiadas

4) Segunda falha ocorre durante recuperação da 1ª falha.

   - Algumas mudanças podem ter sido feitas no BD como consequência de **REDO**, mas todas as mudanças foram feitas.

- **Solução:** executar o algoritmo de recuperação novamente.

Recuperação com Atualizações Adiadas

4) Segunda falha ocorre durante recuperação da 1ª falha.

   - Algumas mudanças podem ter sido feitas no BD como consequência de **REDO**, mas todas as mudanças foram feitas.

- **Solução:** executar o algoritmo de recuperação novamente.

Recuperação com Atualizações Adiadas

- Vantagens / Desvantagens RAA
  - **Vantagem:** Ti não precisa de UNDO
  - **Desvantagem:** limitação de concorrência

Recuperação com Atualização Imediata

- **Motivação:**
  - Visa promover uma maior concorrência de Ti's
  - **Ideia:** quando uma Ti produz um **write**, o BD é atualizado imediatamente, sem esperar pelo COMMIT_PARCIAL
  - **Log:** mesma idéia anterior, com as seguintes entradas:
    1. **<Ti, inicio>**
    2. **< Ti, W, X, Valor_Antigo, Valor_Novo >**
    3. **<Ti, commit>**

Recuperação com Atualização Imediata

- **Algoritmo RAI (com checkpoint)**

1. Iniciar da última entrada do log para trás, voltando em direção do checkpoint (último)
1. Montar 2 listas: transações confirmadas (< Ti, COMMIT > no log após o checkpoint) e transações não confirmadas (com < Ti, Início >, mas sem < Ti, commit > no log)
1. Desfazer todas as operações de **write** das Tis não confirmadas, usando o procedimento UNDO (ordem reversa à ordem de escrita no log)
1. Refazer todos os **writes** das Tis confirmadas, na ordem que foram escritas no log (REDO)

Recuperação com Atualização Imediata

Obs.:

- Gravar BD
- Grava Log

> Susceptível a falhas entre elas

- **Solução:** Protocolo de gravação no log primeiro.
- Não se permite que uma Ti grave um objeto no BD, até que pelo menos a parcela desfazer esteja no log.
- Não se permite que uma Ti confirme, até que ambas as parcelas UNDO e REDO estejam no log.

Recuperação com Atualização Imediata

Procedimento UNDO

- Retorna o valor de todos os objetos atualizados pela Ti para valores antigos.
- Uma Ti precisa ser desfeita se o log contém **<Ti, início>**, mas não contém o registro **<Ti, commit>**

UNDO (X) = UNDO (UNDO (UNDO (X)))

↓

Idempotente

Recuperação com Atualização Imediata

Exemplo:

Considere as transações **T1** e **T2** do exemplo anterior.

O log com a execução completa de **T1** e **T2** seria:

< T1 , Início >

< T1 , A, 1000, 950 > **A = 950**

< T1 , B, 2000, 2050 > **B = 2050**

< T1 , commit >

< T2 , Início >

< T2 , C, 700, 600 > **C = 600**

< T2 , commit >

Recuperação com Atualização Imediata

Simulando algumas falhas:

1) Falha ocorre após **< T1, B, 2000, 2050 >** ser gravado no log. Log na hora da falha:

    < T1, início >

    < T1, A, 1000, 950 >

    < T1, B, 2000, 2050 >

- Quando restart ocorre → UNDO(T1)

Recuperação com Atualização Imediata

Simulando algumas falhas:

1) Falha ocorre após **< T1, B, 2000, 2050 >** ser gravado no log. Log na hora da falha:

    < T1, início >

    < T1, A, 1000, 950 >

    < T1, B, 2000, 2050 >

- Quando restart ocorre → **UNDO(T1)**

Recuperação com Atualização Imediata

2) Falha ocorre após **< T2, C, 700, 600 >** ser gravado no log. Log na hora da falha:

< T1, início >

< T1, A, 1000, 950 >

< T1, B, 2000, 2050 >

< T1, commit >

< T2, início >

< T2, C, 700, 600 >

- Quando restart ocorre → **REDO(T1)** e **UNDO(T2)**

Recuperação com Atualização Imediata

2) Falha ocorre após **< T2, C, 700, 600 >** ser gravado no log. Log na hora da falha:

< T1, início >

< T1, A, 1000, 950 >

< T1, B, 2000, 2050 >

< T1, commit >

< T2, início >

< T2, C, 700, 600 >

- Quando restart ocorre → **REDO(T1)** e **UNDO(T2)**

Recuperação com Atualização Imediata

3) Falha ocorre após **< T2, commit >** ser gravado no log. Log na hora da falha:

< T1, início >

< T1, A, 1000, 950 >

< T1, B, 2000, 2050 >

< T1, commit >

< T2, início >

< T2, C, 700, 600 >

< T2, commit >

- Quando restart ocorre → **REDO(T1)** e **REDO(T2)**

Recuperação com Atualização Imediata

3) Falha ocorre após **< T2, commit >** ser gravado no log.
   Log na hora da falha:

   < T1, início >

   < T1, A, 1000, 950 >

   < T1, B, 2000, 2050 >

   < T1, commit >

   < T2, início >

   < T2, C, 700, 600 >

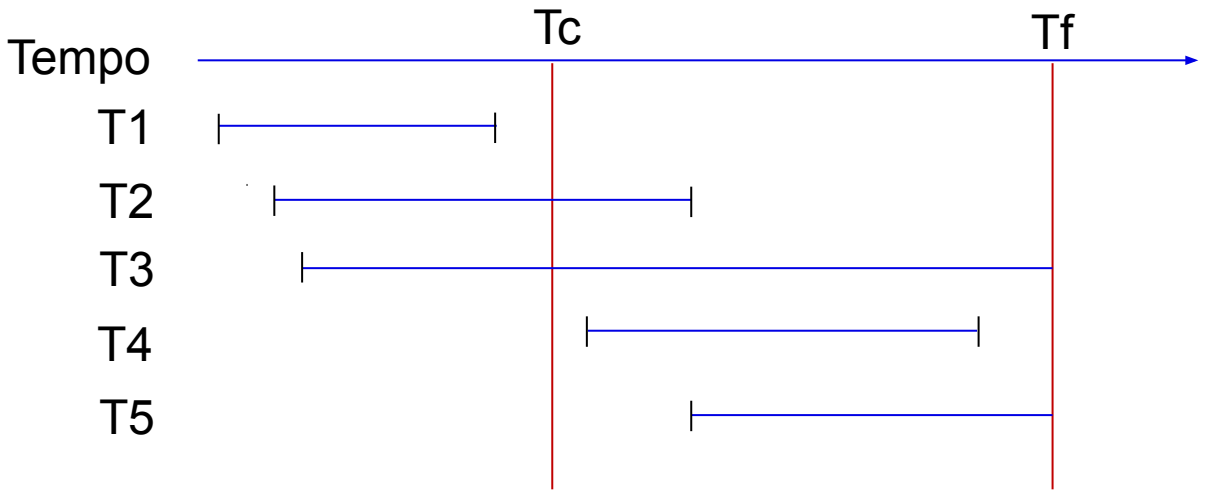
   < T2, commit >

- Quando restart ocorre → **REDO(T1)** e **REDO(T2)**

Recuperação com Atualização Imediata

- Vantagens / Desvantagens RAI
  - **Vantagem:** prover maior concorrência
  - **Desvantagens:**
    - UNDO é necessário
    - Possibilidade de aborto em cascate

Recuperação a Falhas



T_i = transações

T_c = tempo de checkpoint

T_f = tempo de falha

- **REDO(X):** refaz a transação; uma T_i que sofreu C_i antes da falha, mas nem todas as atualizações foram gravadas no BD.

  Ex.: T2 e T4

- **UNDO(X):** desfaz a transação; uma T_i que começou a executar, mas não sofreu commit antes da falha, considerando que algumas atualizações foram feitas → desfazê-las.

  Ex.: T3 e T5