

Sumário

- 1 Introdução ao Processamento de Consultas
- 2 Otimização de Consultas
- 3 Plano de Execução de Consultas
- 4 Introdução a Transações
- 5 Recuperação de Falhas**
- 6 Controle de Concorrência
- 7 Fundamentos de BDs Distribuídos
- 8 SQL Embutida

Recuperação de Falhas

- Garantia de **atomicidade** e **durabilidade** de Transações
 - requer um SGBD **tolerante a falhas**
- Tolerância a falhas em BDs
 - capacidade de conduzir o BD a um **estado passado consistente**, após a ocorrência de uma falha que o deixou em um estado inconsistente
 - baseia-se em **redundância** de dados
 - não é um mecanismo 100% seguro
 - responsabilidade do **subsistema de recovery** do SGBD

Subsistema de *Recovery*

- Controles
 - durante o funcionamento normal do SGBD
 - manter informações sobre o que foi atualizado no BD pelas transações
 - realizar cópias periódicas do BD
 - após a ocorrência de uma falha
 - executar ações para retornar o BD a um estado consistente
 - ações básicas
 - **UNDO**: desfazer uma atualização no BD
 - **REDO**: refazer uma atualização no BD
- Considerações sobre o seu projeto
 - tipos de falhas a tratar
 - técnica de *recovery* a aplicar

Ações Básicas de *Recovery*

- Transaction UNDO
 - uma transação não concluiu suas operações
 - as modificações realizadas por esta transação no BD são desfeitas
- Global UNDO
 - uma ou mais transações não concluíram as suas operações
 - as modificações realizadas por todas estas transações no BD são desfeitas
- Partial REDO
 - na ocorrência de uma falha, algumas transações podem ter concluído suas operações (*committed*), mas suas ações podem não ter se refletido no BD
 - as modificações realizadas por estas transações são refeitas no BD
- Global REDO
 - no caso de um comprometimento do BD, todas as transações *committed* no BD são perdidas
 - as modificações realizadas por todas estas transações no BD são refeitas

Tipos de Falhas

- Falha de Transação

- uma transação ativa termina de forma anormal
- causas
 - violação de RI, lógica da transação mal definida, *deadlock*, cancelamento pelo usuário, ...
- não compromete a memória principal e a memória secundária (disco, em geral)
- falha com maior probabilidade de ocorrência
- seu tempo de recuperação é pequeno
 - ação: [Transaction UNDO](#)

Tipos de Falhas

- Falha de sistema

- o SGBD encerra a sua execução de forma anormal
- causas
 - interrupção de energia, falha no SO, erro interno no SW do SGBD, falha de HW, ...
- compromete a memória principal e não compromete o disco
- falha com probabilidade média de ocorrência
- seu tempo de recuperação é médio
 - ações: [Global UNDO](#) e [Partial REDO](#)

Tipos de Falhas

- Falha de meio de armazenamento
 - o BD torna-se total ou parcialmente inacessível
 - causas
 - setores corrompidos no disco, falha no cabeçote de leitura/gravação, ...
 - não compromete a memória principal e compromete o disco
 - falha com menor probabilidade de ocorrência
 - seu tempo de recuperação é grande
 - ação: Global REDO

Técnicas de *Recovery*

- Baseadas em *Log*
 - modificação imediata do BD
 - técnica UNDO/REDO
 - técnica UNDO/NO-REDO
 - modificação postergada do BD
 - técnica NO-UNDO/REDO
 - recuperação de meio de armazenamento
 - técnica ARCHIVE/DUMP/REDO
 - Baseadas em *Shadow Pages*
 - técnica NO-UNDO/NO-REDO
- recuperação de falhas de transação e de sistema
- recuperação de falhas de transação e de sistema

Técnicas Baseadas em Log

- Técnicas mais comuns de *recovery*
- Utilizam um **arquivo de Log** (ou *Journal*)
 - registra seqüencialmente as atualizações feitas por transações no BD
 - é consultado em caso de falhas para a realização de UNDO e/ou REDO de transações
 - mantido em uma ou mais cópias em memória secundária (disco, fita, ...)
 - tipos de *log*
 - **log de UNDO**
 - mantém apenas o valor antigo do dado (*before image*)
 - **log de REDO**
 - mantém apenas o valor atualizado do dado (*after image*)
 - **log de UNDO/REDO** (mais comum)
 - mantém os valores antigo e atualizado do dado

Tipos de Registro no Log

- Supõe-se que toda transação possui um identificador único gerado pelo SGBD
- Para fins de recuperação de falhas, operações *read* não precisam ser gravadas
 - úteis apenas para outros fins (auditoria, estatísticas, ...)
- Principais tipos de registro
 - **início de transação**: `<start Tx>`
 - **commit de transação**: `<commit Tx>`
 - **atualização**: `<write Tx, X, beforeImage, afterImage>`
 - não é necessário em log REDO ←
 - não é necessário em log UNDO ←

Exemplo de Log

Log

```
<start T3>
<write T3,B,15,12>
<start T2>
<write T2,B,12,18>
<start T1>
<write T1,D,20,25>
<commit T1>
<write T2,D,25,26>
<write T3,A,10,19>
<commit T3>
<commit T2>
...
```

T₁

```
read(A)
read(D)
write(D)
```

T₂

```
read(B)
write(B)
read(D)
write(D)
```

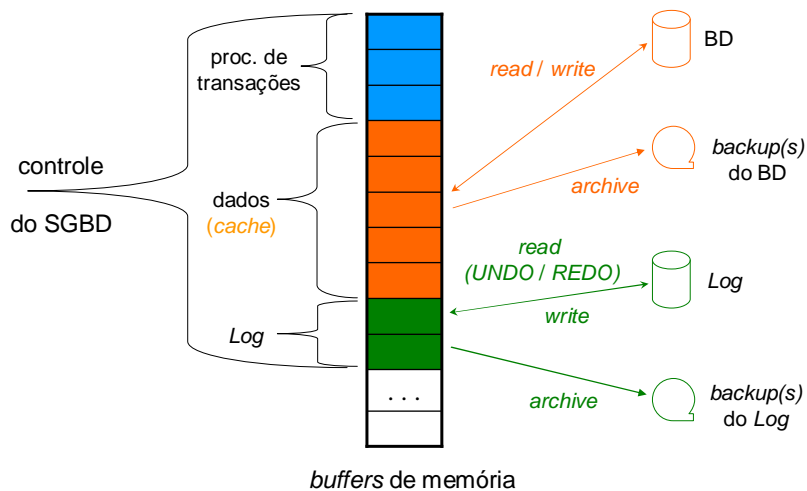
T₃

```
read(C)
write(B)
read(A)
write(A)
```

Gerenciamento de *Buffer*

- *Buffer*
 - conjunto de blocos da memória principal
 - considera-se **bloco** e **página** conceitos sinônimos
- O SGBD é responsável pela gerência de alguns *buffers*
 - *buffers* para dados, para processamento de transações e para o *Log*
 - ele assume o controle desses *buffers*, ao invés do SO, requisitando apenas serviços de leitura/escrita de blocos ao SO

Gerenciamento de *Buffer*



Gerenciamento de *Buffer*

- Técnicas de *recovery* devem sincronizar os *buffers* de *log* e de dados
 - princípio básico
 - um bloco atualizado na *cache* só pode ser gravado no BD após o histórico dos dados atualizados neste bloco ter sido gravado no *Log* em disco
 - *Write-Ahead-Log (WAL)*
 - uma transação T_x só pode passar para o estado *efetivada (committed)* após todas as suas atualizações terem sido gravadas no BD segundo o princípio *WAL*
- O SGBD aplica técnicas de gerenciamento de *buffer*
 - estas técnicas influenciam as técnicas de *recovery*

Técnicas de Gerência de *Buffer*

- *NOT-STEAL*

- um bloco na *cache* utilizado por uma transação T_x *não pode* ser gravado antes do *commit* de T_x
 - bloco possui um *bit de status* indicando se foi (1) ou não (0) modificado
 - *vantagem*: processo de *recovery* mais simples - evita dados de transações inacabadas sendo gravadas no BD

- *STEAL*

- um bloco na *cache* utilizado por uma transação T_x *pode* ser gravado antes do *commit* de T_x
 - necessário se algum dado é requisitado do BD por outra transação e não há blocos disponíveis na *cache*
 - o bloco “vítima” é escolhido através de alguma técnica de SO
 - LRU, FIFO, ...
 - *vantagem*: não há necessidade de manter blocos bloqueados por transações

Técnicas de Gerência de *Buffer*

- *FORCE*

- os blocos que mantêm dados atualizados por uma transação T_x *são* imediatamente gravados no BD quando T_x alcança o *commit*
 - deve-se saber quais os blocos que T_x atualizou dados
- *vantagem*: garante a durabilidade de T_x o mais cedo possível – garante mais o REDO de T_x em caso de falha

- *NOT-FORCE*

- os blocos que mantêm dados atualizados por T_x *não são* imediatamente gravados no BD quando T_x alcança o *commit*
- *vantagem*: blocos atualizados podem permanecer na *cache* e serem utilizados por outras transações, após o *commit* de T_x (reduz custo de acesso a disco)

Exercício 1

- a) Considere o *Log* abaixo após a ocorrência de uma falha de sistema. Apresente os valores resultantes dos dados X e Y para cada alternativa de execução de operações UNDO e REDO proposta abaixo. Qual das alternativas mantém o BD consistente?

```
<start T1>
<start T2>
<write T1,X,1,7>
<start T3>
<write T2,X,7,70>
<commit T1>
<start T4>
<write T3,X,70,700>
<write T3,Y,10,100>
<write T4,Y,100,55>
<commit T4>
<write T2,Y,55,550>
```

- 1) UNDO de T₂ e T₃ + REDO de T₁ e T₄ ↓
- 2) UNDO de T₂ e T₃ + REDO de T₁ e T₄ ↑
- 3) 1ª passada: UNDO de T₂ e T₃ ↑;
2ª passada: REDO de T₁ e T₄ ↓
- 4) 1ª passada: REDO de T₁ e T₄ ↓;
2ª passada: UNDO de T₂ e T₃ ↑

Observações:

- 1) ↓ significa varredura *forward* do *Log*
- 2) ↑ significa varredura *backward* do *Log*

- b) Qual das seguintes combinações de técnicas de gerenciamento de *buffer* requer um gerenciamento mais complexo por parte do SGBD? Por quê?
- a) STEAL + NOT-FORCE
 - b) NOT-STEAL + FORCE

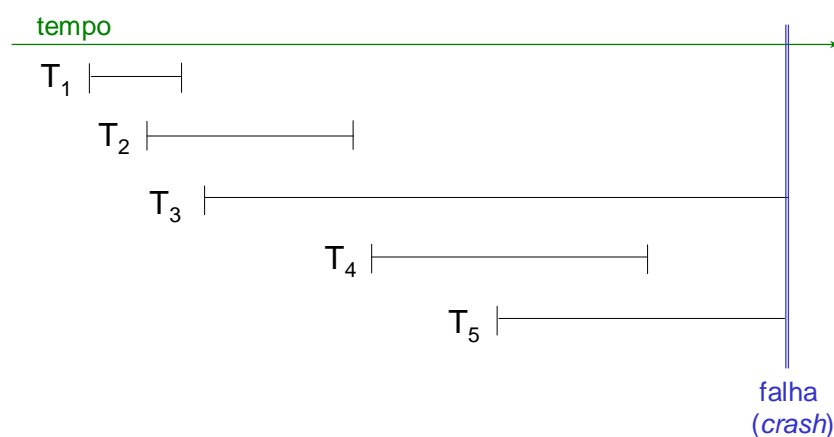
Modificação Imediata do BD

- Abordagem na qual dados atualizados por uma transação T_x **podem ser gravados** no BD **antes** do *commit* de T_x
- Abordagem mais comum de *recovery*
 - gerenciamento de *buffer* mais simples
 - utiliza técnica **STEAL**
- Duas técnicas
 - **UNDO/REDO**
 - técnica mais comum de *recovery*
 - **UNDO/NO-REDO**

Técnica UNDO/REDO

- Grava o *commit* de T_x no Log *depois de* todas as atualizações de T_x terem sido gravadas no Log, e *antes* dessas atualizações serem gravadas no BD
 - requer um Log de UNDO/REDO
- Utiliza 2 listas de transações
 - lista-REDO: IDs de transações *committed*
 - possuem *commit* gravado no Log
 - lista-UNDO: IDs de transações ativas
- Procedimento (em caso de falha)
 1. faz uma varredura *backward do Log*, realizando UNDO das transações na lista-UNDO
 2. faz uma varredura *forward do Log*, realizando REDO das transações na lista-REDO

Técnica UNDO/REDO - Exemplo



lista-UNDO: T₃, T₅ (devem sofrer UNDO)

lista-REDO: T₁, T₂, T₄ (devem sofrer REDO)

Técnica UNDO/REDO

- A propriedade de **idempotência** de operações UNDO e REDO é válida
 - fazer UNDO ou REDO uma vez ou várias vezes produz o mesmo resultado
 - situações em que ocorrem falhas durante o processo de *recovery*
- Técnica mais trabalhosa de *recovery*
 - tanto UNDO quanto REDO devem ser realizados
 - porém, o gerenciamento de *buffer* é mais simples

Exercício 2

- a) Na técnica UNDO/REDO, suponha que uma varredura inicial seja feita no *Log* para montar a lista-UNDO e a lista-REDO, antes da realização das varreduras *backward* e *forward*. Proponha algoritmos de alto nível para definir essas listas através de:
- Uma varredura *backward* do *Log*;
 - Uma varredura *forward* do *Log*.
- Qual algoritmo é mais simples?
- b) No item anterior, o algoritmo UNDO/REDO realiza 3 varreduras no *Log*. Proponha um algoritmo mais eficiente, que resolva o problema realizando apenas 2 varreduras no *Log* (uma *backward* e outra *forward*)



vídeo do bairdo