Banco de Dados

Prof. Dr. Ronaldo Celso Messias Correia
ronaldo.correia@unesp.br

Sistema de Recuperação

Introdução

- Um Sistema Computador está sujeito a falhas:
  - Quebra de disco, falha de energia, erro de software, fogo na sala de equipamento, sabotagem, etc.
- Em cada um destes casos, dados podem ser perdidas
- O sistema de banco de dados deve precaver-se para garantir que as propriedades de atomicidade e durabilidade de uma transação sejam preservadas
- Uma parte integrante de um sistema de banco de dados é o esquema de recuperação que é responsável pela restauração do banco de dados para um estado consistente que havia antes da ocorrência da falha

Classificação de falha

- O tipo de falha mais simples – não resulta na perda da informação. Falhas mais difíceis de tratar – resultam em perda de informação.
- Falha de transação:
  - **Erros lógicos** : a transação não pode completar devido a alguma condição de erro interna
  - **Erros do sistema** : o sistema de banco de dados precisa terminar uma transação ativa devido a uma condição de erro (por exemplo, deadlock)
- **Falha do sistema** : uma falta de energia ou outra falha do hardware ou software faz com que o sistema falhe.
  - Suposição falhar-parar: conteúdo do armazenamento não volátil é considerado como não sendo corrompido por falha do sistema
    - Sistemas de banco de dados possuem diversas verificações de integridade para impedir adulteração de dados do disco
- **Falha do disco** : uma falha de cabeça ou falha de disco semelhante destrói todo ou parte do armazenamento de disco
  - A destruição é considerada como detectável: unidades de disco usam somas de verificação para detectar falhas

Algoritmos de recuperação

- Algoritmos de recuperação são técnicas para garantir a consistência do banco de dados e a atomicidade e durabilidade da transação apesar das falhas.
- Algoritmos de recuperação têm duas partes:
  - Ações tomadas durante o processamento normal da transação para garantir que existem informações suficiente para recuperação de falhas
  - Ações tomadas após uma falha para recuperar o conteúdo do banco de dados a um estado que garante atomicidade, consistência e durabilidade

Estrutura de armazenamento

- Armazenamento volátil:
  - não sobrevive a falhas do sistema
  - exemplos: memória principal, memória cache
- Armazenamento não volátil:
  - sobrevive a falhas do sistema
  - exemplos: disco, fita, memória flash, RAM não-volátil (alimentada por bateria)
- Armazenamento estável:
  - a informação residente em armazenamento estável nunca é perdida. Uma forma mítica de armazenamento que sobrevive a todas as falhas
  - aproximado mantendo-se várias cópias em meios não voláteis distintos

Implementação do armazenamento estável

- Mantenha várias cópias de cada bloco em discos separados
  - as cópias podem estar em sites remotos para proteger contra desastres como incêndio ou inundação
- A falha durante a transferência de dados ainda pode resultar em cópias inconsistentes: a transferência em bloco pode resultar em
  - Término bem-sucedido
  - Falha parcial: bloco de destino possui informações incorretas
  - Falha total: bloco de destino nunca foi atualizado
- Proteção do meio de armazenamento contra falha durante a transferência de dados (uma solução):
  - Executar a operação de saída da seguinte forma (considerando duas cópias de cada bloco):
    - Escreva a informação no primeiro bloco físico.
    - Quando a primeira escrita terminar com sucesso, escreva a mesma informação no segundo bloco físico.
    - A saída só é completada depois que a segunda escrita for completada com sucesso.

Implementação de armazenamento estável (cont.)

- Proteção do meio de armazenamento contra falha durante a transferência de dados (cont.):
- As cópias de um bloco podem diferir devido a falhas durante a operação de saída. Para recuperar-se da falha:
  - Primeiro, encontre blocos inconsistentes:
    - Solução simples: Compare as duas cópias de cada bloco do disco.
    - Solução melhor:
      - Registre as escritas em disco em andamento no armazenamento não volátil (RAM não volátil ou área especial do disco).
      - Use essa informação durante a recuperação para encontrar blocos que podem ser inconsistentes, e só compare cópias destes.
      - Usada em sistemas RAID de hardware
  - Se uma cópia de um bloco inconsistente for detectada com um erro (soma de verificação defeituosa), escreva a outra cópia em cima dela. Se ambos não têm erro, mas forem diferentes, escreva o primeiro bloco sobre o segundo.

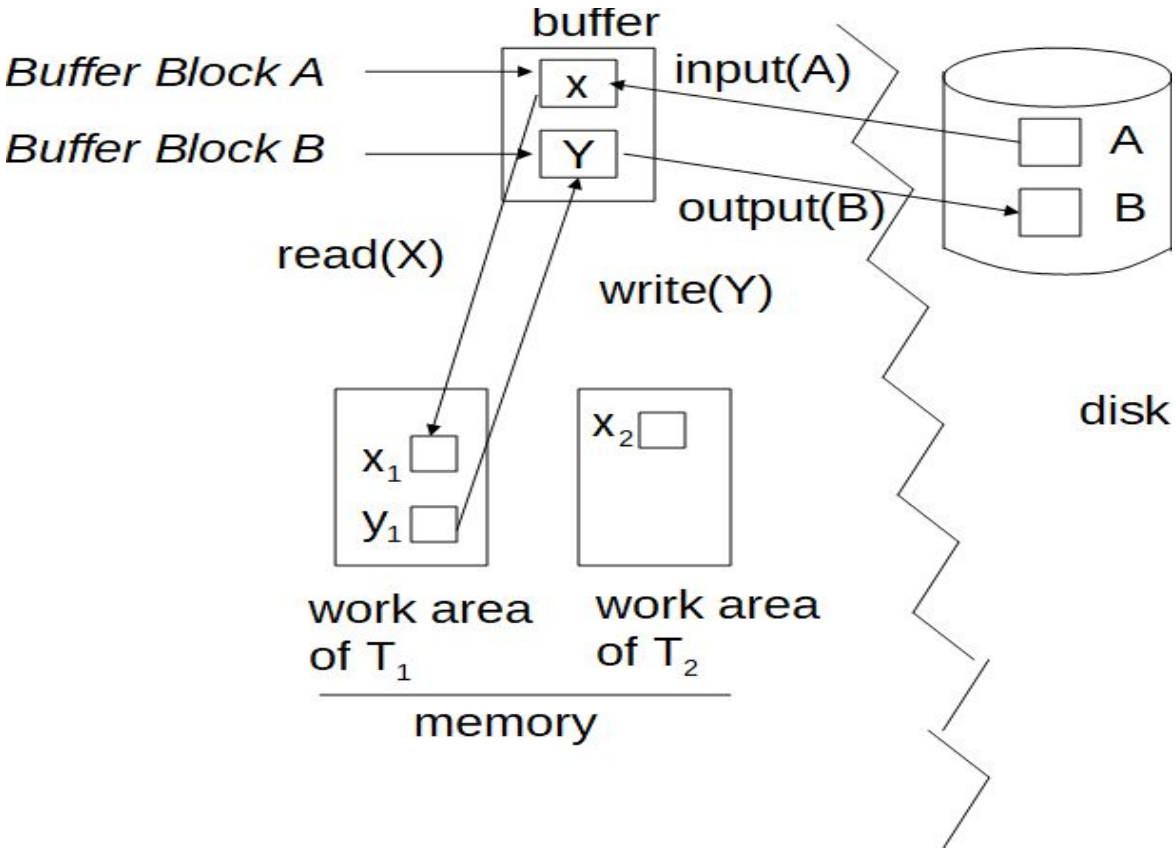
Acesso aos dados

- Blocos físicos são aqueles blocos residindo no disco.
- Blocos de buffer são os blocos residindo temporariamente na memória principal.
- Movimentos de bloco entre disco e a memória principal são iniciados por meio das duas operações a seguir:
  - input(B) transfere o bloco físico B para a memória principal.
  - output(B) transfere o bloco de buffer B para o disco, e substitui o bloco físico apropriado lá.
- Cada transação Ti possui sua área de trabalho privada, onde são mantidas as cópias locais de todos os itens de dados acessados e atualizados por ela.
  - A cópia local de Ti de um item de dados X é chamada de xi.
- Consideramos, por simplicidade, que cada item de dados cabe e é armazenado dentro de um único bloco.

Acesso aos dados (cont.)

- A transação transfere itens de dados entre os blocos de buffer do sistema e sua área de trabalho privada usando as seguintes operações:
  - read(X) atribui o valor do item de dados X à variável local xi.
  - write(X) atribui o valor da variável local xi ao item de dados {X} no bloco de buffer.
  - esses dois comandos podem precisar da emissão de uma instrução input(BX) antes da atribuição, se o bloco BX em que X reside ainda não estiver na memória.
- Transações
  - Execute read(X) enquanto acessa X pela primeira vez;
  - Todos os acessos subsequentes são para a cópia local.
  - Após o último acesso, a transação executa write(X).
- output(BX) não precisa vir imediatamente após write(X). O sistema pode realizar a operação output quando julgar necessário.

Exemplo de acesso aos dados



Recuperação e atomicidade

- Modificar o banco de dados sem garantir que a transação será confirmada pode levar o banco de dados a um estado inconsistente.
- Considere a transação Ti que transfere $50 da conta A para a conta B; o objetivo é realizar todas as modificações do banco de dados feitas por Ti ou nenhuma delas.
- Várias operações de saída podem ser exigidas para Ti (para gerar A e B). Uma falha pode ocorrer após uma dessas modificações ter sido feita, mas antes que todas elas sejam feitas.

Recuperação e atomicidade (cont.)

- Para garantir a atomicidade apesar das falhas, primeiro geramos informações descrevendo as modificações no armazenamento estável sem modificar o próprio banco de dados.
- Estudamos duas técnicas:
  - recuperação baseada em log, e
  - paginação de sombra
- Consideramos (inicialmente) que as transações serão executadas em série, ou seja, uma após a outra.

Recuperação baseada em log

- Um log é mantido no armazenamento estável.
  - O log é uma sequência de registros de log, e mantém um registro das atividades de atualização no banco de dados.
- Quando a transação Ti inicia, ela se registra escrevendo um registro de log <Ti start>
- Antes que Ti execute write(X), um registro de log <Ti, X, V1, V2> é escrito, onde V1 é o valor de X antes do write, e V2 é o valor a ser escrito em X.
  - O registro de log observa que Ti realizou uma escrita no item de dados Xj. Xj tinha o valor V1 antes da escrita, e terá o valor V2 após a escrita.
- Quando Ti termina sua última instrução, o registro de log <Ti commit> é escrito.
- Consideramos, por enquanto, que os registros de log são escritos diretamente no armazenamento estável (ou seja, eles não são mantidos em buffer)
- Duas técnicas usando logs
  - Modificação de banco de dados adiada
  - Modificação de banco de dados imediata

Modificação de banco de dados adiada

- O esquema de modificação de banco de dados adiada registra todas as modificações no log, mas adia todas as escritas para depois da confirmação parcial.
- Suponha que as transações são executadas serialmente
- A transação começa escrevendo o registro <Ti start> no log.
- Uma operação write(X) resulta em um registro de log <Ti, X, V> sendo escrito, onde V é o novo valor para X
  - Nota: o valor antigo não é necessário para esse esquema
- A escrita não é realizada em X neste momento, mas é adiada.
- Quando Ti confirma parcialmente, <Ti commit> é escrito no log
- Finalmente, os registros de log são lidos e usados para realmente executar as escritas previamente adiadas.

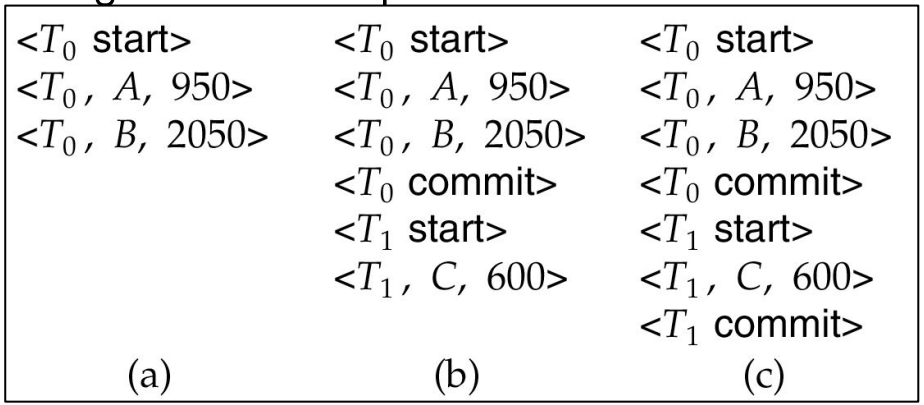
Modificação de banco de dados adiada (cont.)

- Durante a recuperação após uma falha, uma transação precisa ser refeita se e somente se tanto <Ti start> quanto <Ti commit> existirem no log.
- Refazer uma transação Ti (redo Ti) define o valor de todos os itens de dados atualizado pela transação como os novos valores.
- Falhas podem ocorrer enquanto
  - a transação estiver executando as atualizações originais, ou
  - enquanto a ação de recuperação estiver sendo tomada
- transações de exemplo T0 e T1 (T0 executa antes de T1):

T0: read (A)	T1 : read (C)
A: - A - 50	C:- C- 100
Write (A)	write (C)
read (B)	
B:- B + 50	
write (B)	

Modificação de banco de dados adiada (cont.)

- A seguir mostramos o log conforme aparece em três instâncias de tempo.



- Se o log no armazenamento estável no momento da falha for como neste caso:
  - (a) Nenhuma ação de rede precisa ser tomada
  - (b) redo(T0) precisa ser realizado, pois <T0 commit> está presente
  - (c) redo(T0) precisa ser realizado seguido por redo(T1), pois <T0 commit> e

  <Ti commit> estão presentes

Modificação de banco de dados imediata

- O esquema de modificação de banco de dados imediata permite que atualizações de banco de dados de uma transação não confirmada sejam feitas enquanto as escritas são emitidas
  - como pode ser preciso desfazer, os logs de atualização precisam ter valor antigo e valor novo
- O registro de log de atualização precisa ser escrito antes que o item do banco de dados seja escrito
  - Consideramos que o registro de log é enviado diretamente ao armazenamento estável
  - Pode ser estendido para adiar a saída do registro de log, desde que, antes da execução de uma operação output(B) para um bloco de dados B, todos os registros de log correspondentes aos itens B sejam esvaziados para o armazenamento estável
- A saída dos blocos atualizados pode ocorrer a qualquer momento antes ou depois do commit da transação
- A ordem em que os blocos são enviados pode ser diferente da ordem em que são escritos.

Exemplo de modificação de banco de dados imediata

Log	Write	Output
<T0 start>		
<T0, A, 1000, 950>		
To, B, 2000, 2050		
	A = 950	
	B = 2050	
<T0 commit>		
<T1 start>		
<T1, C, 700, 600>		
	C = 600	
		BB, BC
<T1 commit>		
		BA

Nota: BX indica bloco contendo X.

Modificação de banco de dados imediata (cont.)

- O procedimento de recuperação possui duas operações em vez de uma:
  - undo(Ti) restaura o valor de todos os itens de dados atualizados por Ti aos seus valores antigos, indo para trás a partir do último registro para Ti
  - redo(Ti) define o valor de todos os itens de dados atualizados por Ti aos novos valores, indo para frente a partir do primeiro registro para Ti
- As duas operações precisam ser idempotentes
  - Ou seja, mesmo que a operação seja executada várias vezes, o efeito é o mesmo que se fosse executada uma vez
    - Necessário porque as operações podem ser novamente executadas durante a recuperação
- Ao recuperar-se após a falha:
  - A transação Ti precisa ser desfeita (undo) se o log tiver o registro <Ti start>, mas não contém o registro <Ti commit>.
  - A transação Ti precisa ser refeita (redo) se o log tiver o registro <Ti start> e o registro <Ti commit>.
- Operações de undo são realizadas primeiro, depois as operações de redo.

Exemplo de recuperação de modificação de BD imediata

- A seguir mostramos o log conforme aparece em três instâncias de tempo.

(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>

- As ações de recuperação em cada um destes são:
  - (a) undo (T0): B é restaurado para 2000 e A para 1000.
  - (b) undo (T1) e redo (T0): C é restaurado para 700, e depois A e B são definidos para 950 e 2050, respectivamente.
  - (c) redo (T0) e redo (T1): A e B são definidos para 950 e 2050 respectivamente. Depois, C é definido para 600

Pontos de verificação

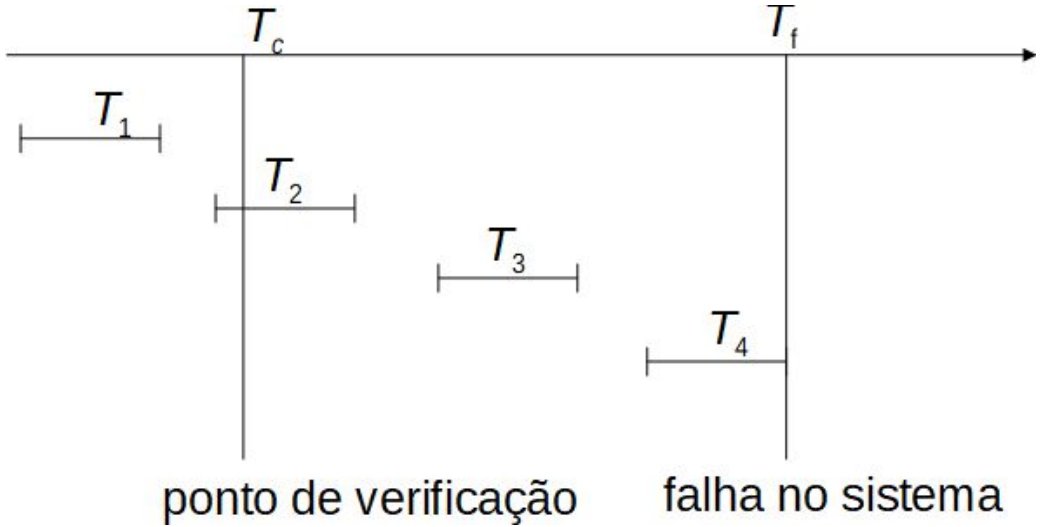
- Problemas no procedimento de recuperação, conforme discutimos:
  - pesquisar o log inteiro é demorado
  - poderíamos desnecessariamente refazer transações que já emitiram sua saída no banco de dados.
- Facilite o procedimento de recuperação realizando periodicamente o ponto de verificação
  - Envie todos os registros de log atualmente residindo na memória principal para o armazenamento estável.
  - Envie todos os blocos de buffer modificados para o disco.
  - Escreva um registro de log < checkpoint> no armazenamento estável.

Pontos de verificação (cont.)

- Durante a recuperação, temos que considerar apenas a transação mais recente Ti que foi iniciada antes do ponto de verificação, e as transações que começaram após Ti.
  - Varra para trás a partir do final do log para encontrar o registro <checkpoint> mais recente.
  - Continue varrendo para trás até um registro <Ti start> ser encontrado.
  - Só precisa considerar a parte do log vindo após o registro start. A parte inicial do log pode ser ignorada durante a recuperação, e pode ser apagada sempre que for desejado.
  - Para todas as transações (começando de Ti ou mais) sem <Ti commit>, execute undo(Ti). (Feito apenas no caso de modificação imediata.)
  - Varrendo para frente no log, para todas as transações começando a partir de Ti ou depois com um <Ti commit>, execute redo(Ti).

Exemplo de pontos de verificação

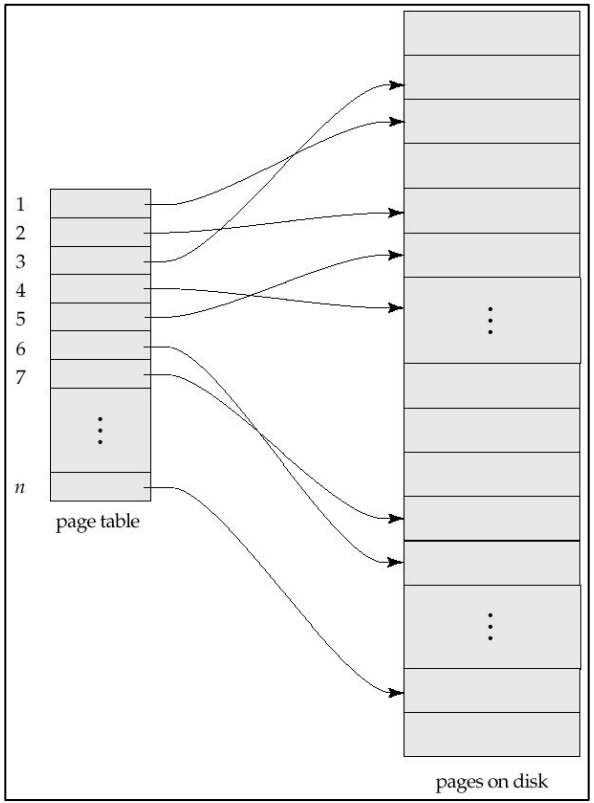
- T1 pode ser ignorada (atualizações já enviadas ao disco devido ao ponto de verificação)
  - T2 e T3 refeitos
  - T4 refeito



Paginação de sombra

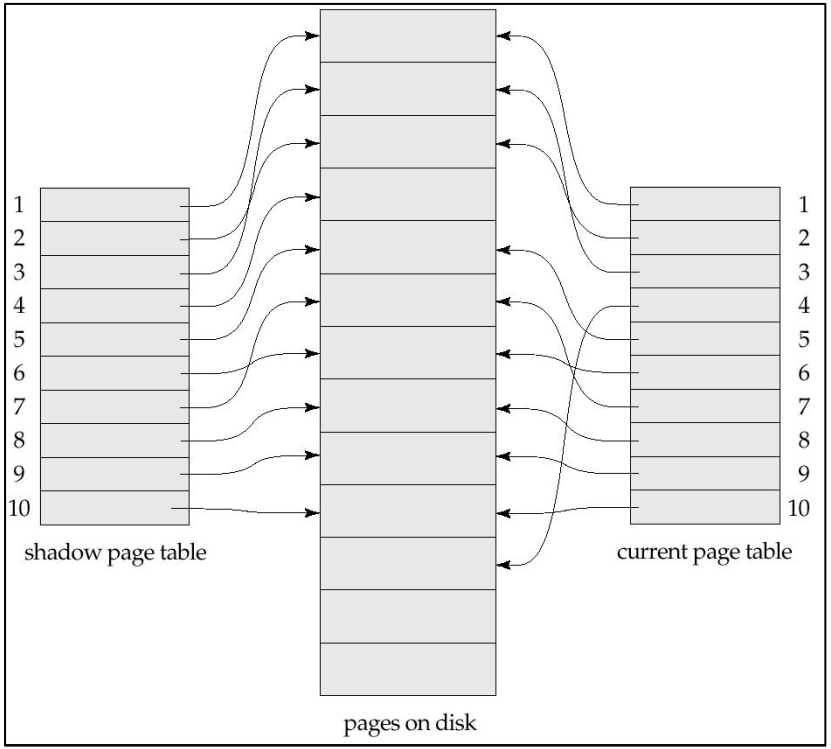
- A paginação de sombra é uma alternativa à recuperação baseada em log; esse esquema é útil se as transações forem executadas em série
- Idéia: manter duas tabelas de página durante o tempo de vida de uma transação - a tabela de página atual, e a tabela de página de sombra
- Armazene a tabela de página de sombra no armazenamento não volátil, de modo que o estado do banco de dados antes da execução da transação possa ser recuperado.
  - A tabela de página de sombra nunca é modificada durante a execução
- Para começar, as duas tabelas de página são idênticas. Somente a tabela de página atual é usada para os acessos ao item de dados durante a execução da transação.
- Sempre que qualquer página estiver para ser escrita pela primeira vez
  - Uma cópia dessa página é feita em uma página não usada.
  - A tabela de página atual aponta para a cópia
  - A atualização é realizada na cópia

Exemplo de tabela de página



Exemplo de paginação de sombra

- Tabelas de sombra e página atual após escrita na página 4



Paginação de sombra (cont.)

- Para confirmar uma transação:
  - 1. Esvazie todas as páginas modificadas na memória principal para o disco
  - 2. Envie a tabela de página atual para o disco
  - 3. Torne a tabela de página atual a nova tabela de página de sombra, da seguinte maneira:
    - mantenha um ponteiro para a tabela de página de sombra em um local fixo (conhecido) no disco.
    - para tornar a tabela de página atual a nova tabela de página de sombra, basta atualizar o ponteiro para apontar para a tabela de página atual no disco
- Quando o ponteiro para a tabela de página de sombra tiver sido escrito, a transação está confirmada.
- Nenhuma recuperação é necessária após uma falha - novas transações podem começar imediatamente, usando a tabela de página de sombra.
- As páginas não apontada de/para a tabela de página atual/sombra devem ser liberadas (com coleta de lixo).

Paginação de sombra (cont.)

- Vantagens da página de sombra em relação aos esquemas baseados em log
  - a recuperação é trivial
  - nenhuma sobrecarga de escrita de registros de log
- Desvantagens:
  - Copiar a tabela de página inteira é muito dispendioso
    - Pode ser reduzido usando uma tabela de página estruturada como uma árvore B+
      - Não é preciso copiar árvore inteira, somente os caminhos na árvore que levam a nós de folha atualizados
  - A sobrecarga do commit é alta mesmo com a extensão acima
    - Precisa esvaziar cada página atualizada, e tabela de página
  - Os dados são fragmentados (páginas relacionadas ficam separadas no disco)
  - Após o término de cada transação, as páginas do banco de dados contendo versões antigas de dados modificados precisam passar pela coleta de lixo
  - Difícil de estender o algoritmo para permitir que transações sejam executadas simultaneamente
    - Mais fácil para estender esquemas baseados em log

Recuperação com transações concorrentes

- Modificamos os esquemas de recuperação baseados em log para permitir que várias transações sejam executadas simultaneamente.
  - Todas as transações compartilham um único buffer de disco e um único log
  - Um bloco de buffer pode ter itens de dados atualizados por uma ou mais transações
- Consideramos o controle de concorrência usando um bloqueio estrito em duas fases;
  - ou seja, as atualizações de transações não confirmadas não devem ser visíveis a outras transações
    - Caso contrário, como realizar o undo se T1 atualiza A, depois T2 atualiza A e confirma, e finalmente T1 precisa abortar?
- O logging é feito conforme descrevemos anteriormente.
  - Os registros de log de diferentes transações podem ser intercalados no log.
- A técnica de ponto de verificação e as ações tomadas na recuperação precisam ser alteradas
  - pois várias transações podem estar ativas quando um ponto de verificação é realizado.

Recuperação com transações concorrentes (cont.)

- Os pontos de verificação são realizados como antes, exceto que o registro de log do ponto de verificação agora tem a forma < checkpoint L> onde L é a lista de transações ativas no momento do ponto de verificação
  - Consideramos que nenhuma atualização está em andamento enquanto o ponto de verificação é executado (isso será aliviado mais tarde)
- Quando o sistema se recupera de uma falha, ele primeiro faz o seguinte:
  - Inicializa a lista de undo e lista de redo para vazio
  - Varre o log para trás a partir do fim, parando quando o primeiro registro <checkpoint L> for encontrado. Para cada registro encontrado durante a varredura:
    - se o registro for <Ti commit>, acrescenta Ti à lista de redo
    - se o registro for <Ti start>, então se Ti não está na lista de redo, acrescenta Ti à lista de undo
  - Para cada Ti em L, se Ti não estiver na lista de redo, acrescenta Ti à lista de undo

Recuperação com transações concorrentes (cont.)

- Neste ponto, lista de undo consiste em transações incompletas, que precisam ser desfeitas, e lista de redo consiste em transações acabadas, que precisam ser refeitas.
- A recuperação agora continua da seguinte forma:
  - Varra o log para trás a partir do registro mais recente, parando quando registros <Ti start> tiverem sido encontrados para cada Ti na lista de undo.
    - Durante a varredura, realize undo para cada registro de log que pertence a uma transação na lista de undo.
  - Localize o registro <checkpoint L> mais recente.
  - Varra o log para frente a partir do registro <checkpoint L> até o final do log.
    - Durante a varredura, realize redo para cada registro de log que pertence a uma transação na lista de redo.

Exemplo de recuperação

- Percorra as etapas do algoritmo de recuperação no log a seguir:
  - <T0 start>
  - <T0, A, 0, 10>
  - <T0 commit>
  - <T1 start>
  - <T1, B, 0, 10>
  - <T2 start> /* Varredura na etapa 4 pára aqui */
  - <T2, C, 0, 10>
  - <T2, C, 10, 20>
  - <checkpoint {T1, T2}>
  - <T3 start>
  - <T3, A, 10, 20>
  - <T3, D, 0, 10>
  - <T3 commit>

Buffering de registro de log

- Buffering de registro de log: os registros de log são mantidos na memória principal, em vez de serem enviados diretamente para o armazenamento estável.
  - Registros de log são enviados ao armazenamento estável quando um bloco de registros de log no buffer estiver cheio, ou uma operação de log forçado for executada.
- O log forçado é realizado para confirmar uma transação forçando todos os seus registros de log (incluindo o registro de commit) para o armazenamento estável.
- Vários registros de log, portanto, podem ser enviados por meio de uma única operação de saída, reduzindo o custo da E/S.

Buffering de registro de log (cont.)

- As regras a seguir precisam ser seguidas se os registros de log forem colocados em buffer:
  - Os registros de log são enviados para o armazenamento estável na ordem em que são criados.
  - A transação Ti só entra no estado de commit quando o registro de log <Ti commit> tiver sido enviado ao armazenamento estável.
  - Antes que um bloco de dados na memória principal seja enviado ao banco de dados, todos os registros de log pertencentes aos dados nesse bloco precisam ter sido enviados ao armazenamento estável.
    - Essa regra é chamada logging de escrita antecipada ou regra WAL (Write Ahead Logging)
      - Estritamente falando, WAL só requer que informações de undo sejam enviadas

Buffering de banco de dados

- O banco de dados mantém um buffer na memória dos blocos de dados
  - Quando um novo bloco é necessário, se o buffer estiver cheio, um bloco existente precisa ser removido do buffer
  - Se o bloco escolhido para remoção tiver sido atualizado, ele terá que ser enviado para o disco
- Como resultado da regra de logging de escrita antecipada, se um bloco com atualizações não confirmadas for enviado ao disco, os registros de log com informações de undo para as atualizações são enviados ao log no armazenamento estável primeiro.
- Nenhuma atualização deve estar em progresso em um bloco quando for enviada ao disco. Pode ser garantido da seguinte forma.
  - Antes de escrever um item de dados, a transação adquire bloqueio exclusivo sobre o bloco contendo o item de dados
  - O bloqueio pode ser liberado quando a escrita terminar.
    - Tais bloqueios mantidos por uma curta duração software chamados de latches.
  - Antes que um bloco seja enviado ao disco, o sistema adquire um latch exclusivo sobre o bloco
    - Garante que nenhuma atualização pode estar em andamento no bloco

Gerenciamento de buffer (cont.)

- O buffer de banco de dados pode ser implementado
  - em uma área de memória principal real reservada para o banco de dados, ou
  - na memória virtual
- Implementar o buffer na memória principal reservada tem desvantagens:
- A memória é particionada de antemão entre o buffer de banco de dados e as aplicações, limitando a flexibilidade.
  - As necessidades podem mudar, e embora o sistema operacional saiba melhor como a memória deve ser dividida a qualquer momento, ele não pode mudar o particionamento da memória.

Gerenciamento de buffer (cont.)

- Os buffers de banco de dados geralmente são implementados na memória virtual apesar de algumas desvantagens:
  - Quando o sistema operacional precisa expulsar uma página que foi modificada, para criar espaço para outra página, a página é escrita no espaço de swap no disco.
  - Quando o banco de dados decide escrever a página de buffer no disco, a página de buffer pode estar no espaço de swap e pode ter que ser lida do espaço de swap no disco e enviada ao banco de dados no disco, resultando em E/S extra!
    - Conhecido como problema de paginação dual.
  - O ideal é que, quando houver um swap de uma página de buffer do banco de dados, o sistema operacional passe o controle ao banco de dados, que por sua vez envia a página ao banco de dados, ao invés do espaço de swap (cuidado para enviar primeiro os registros de log)
    - Assim, a paginação dual pode ser evitada, mas os sistemas operacionais comuns não admitem tal funcionalidade.

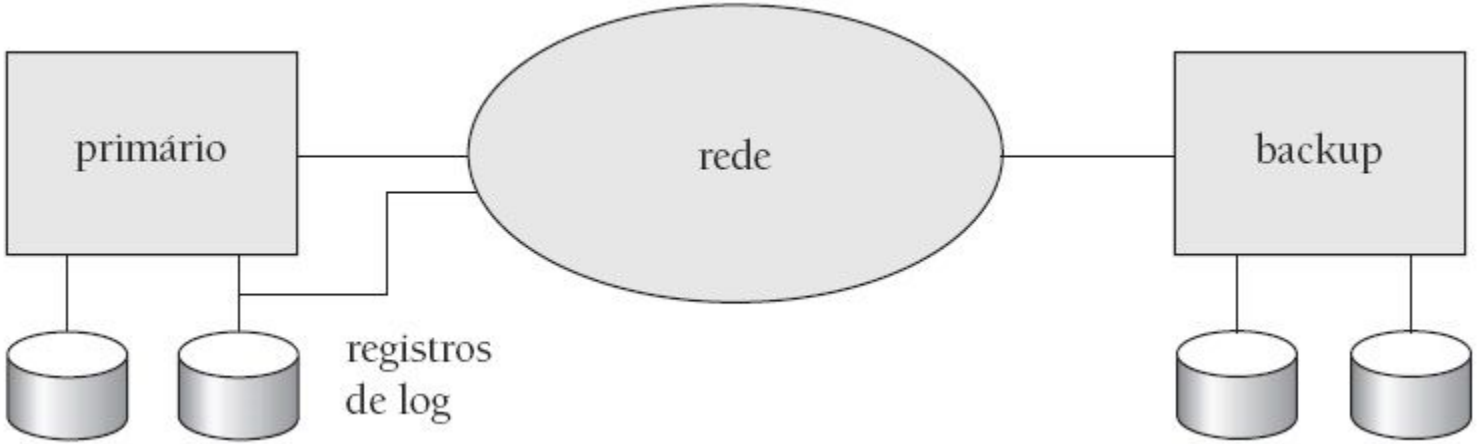
Falha com perda de armazenamento não volátil

- Até aqui, não assumimos perda de armazenamento não volátil
- Técnica semelhante ao ponto de verificação usada para lidar com perda de armazenamento não volátil
  - Periodicamente faz o dump do conteúdo inteiro do banco de dados para armazenamento estável
  - Nenhuma transação pode estar ativa durante o procedimento de dump; um procedimento semelhante ao ponto de verificação precisa ocorrer
    - Envie todos os registros de log atualmente residindo na memória principal para o armazenamento estável.
    - Envie todos os blocos de buffer para o disco.
    - Copie o conteúdo do banco de dados para o armazenamento estável.
    - Envie um registro <dump> para log no armazenamento estável.
  - Para se recuperar da falha de disco
    - restaurar o banco de dados do dump mais recente
    - Consulte o log e refaça todas as transações que foram confirmadas após o dump
- Pode ser estendido para permitir que as transações estejam ativas durante o dump; conhecido como dump difuso ou dump online
  - Estudaremos o ponto de verificação difuso mais adiante

Sistemas de backup remoto

Sistemas de backup remoto

- Os sistemas de backup remoto oferecem alta disponibilidade, permitindo que o processamento de transação continue mesmo que o site primário seja destruído.



Sistemas de backup remoto (cont.)

- Detecção de falha: o site de backup precisa detectar quando o site primário falhou
  - para distinguir a falha do site primário da falha do enlace, mantenha vários links de comunicação entre o site primário e o backup remoto.
- Transferência de controle:
  - Para assumir o controle, o site de backup primeiro realiza a recuperação usando sua cópia do banco de dados e todos os registros de log que recebeu do site primário.
    - Assim, transações completadas são refeitas e transações incompletas são descartadas.
  - Quando um site de backup assume o processamento, ele se torna o novo site primário
  - Para transferir o controle de volta ao antigo primário quando se recuperar, o antigo primário precisa receber logs de redo do backup antigo e aplicar todas as atualizações localmente.

Sistemas de backup remoto (cont.)

- Tempo para recuperação: Para reduzir o atraso na retomada, o site de backup periodicamente processa os registros de log de redo (com efeito, realizando a recuperação do estado anterior do banco de dados), realiza um ponto de verificação e pode então excluir partes mais antigas do log.
- A configuração hot-spare permite a retomada muito rápida:
  - O backup processa continuamente o registro de log de rede enquanto chega, aplicando as atualizações localmente.
  - Quando a falha do site primário é detectada, o backup reverte transações incompletas e está pronto para processar novas transações.
- Alternativa para o backup remoto: banco de dados distribuído com dados replicados
  - O backup remoto é mais rápido e mais barato, porém menos tolerante a falhas

Sistemas de backup remoto (cont.)

- Garante a durabilidade das atualizações adiando o commit da transação até que a atualização seja registrada no backup; evite esse atraso permitindo menores graus de durabilidade.
- One-safe: confirme assim que o registro de log de commit da transação for escrito no site primário
  - Problema: as atualizações podem não chegar no backup antes que ele assuma.
- Two-very-safe: confirme quando o registro de log de commit da transação for escrito no site primário e no backup
  - Reduz a disponibilidade, pois as transações não podem confirmar se um site falhar.
- Two-safe: prossegue como no two-very-safe se o site primário e de backup estiverem ativos. Se apenas o primário estiver ativo, a transação confirma assim que seu registro de log de commit for escrito no primário.
  - Melhor disponibilidade do que two-very-safe; evita problema de transações perdidas no one-safe.