Unidade IV

Transações

- Conceitos
- Propriedades
- Concorrência
- Serialização

Conceitos

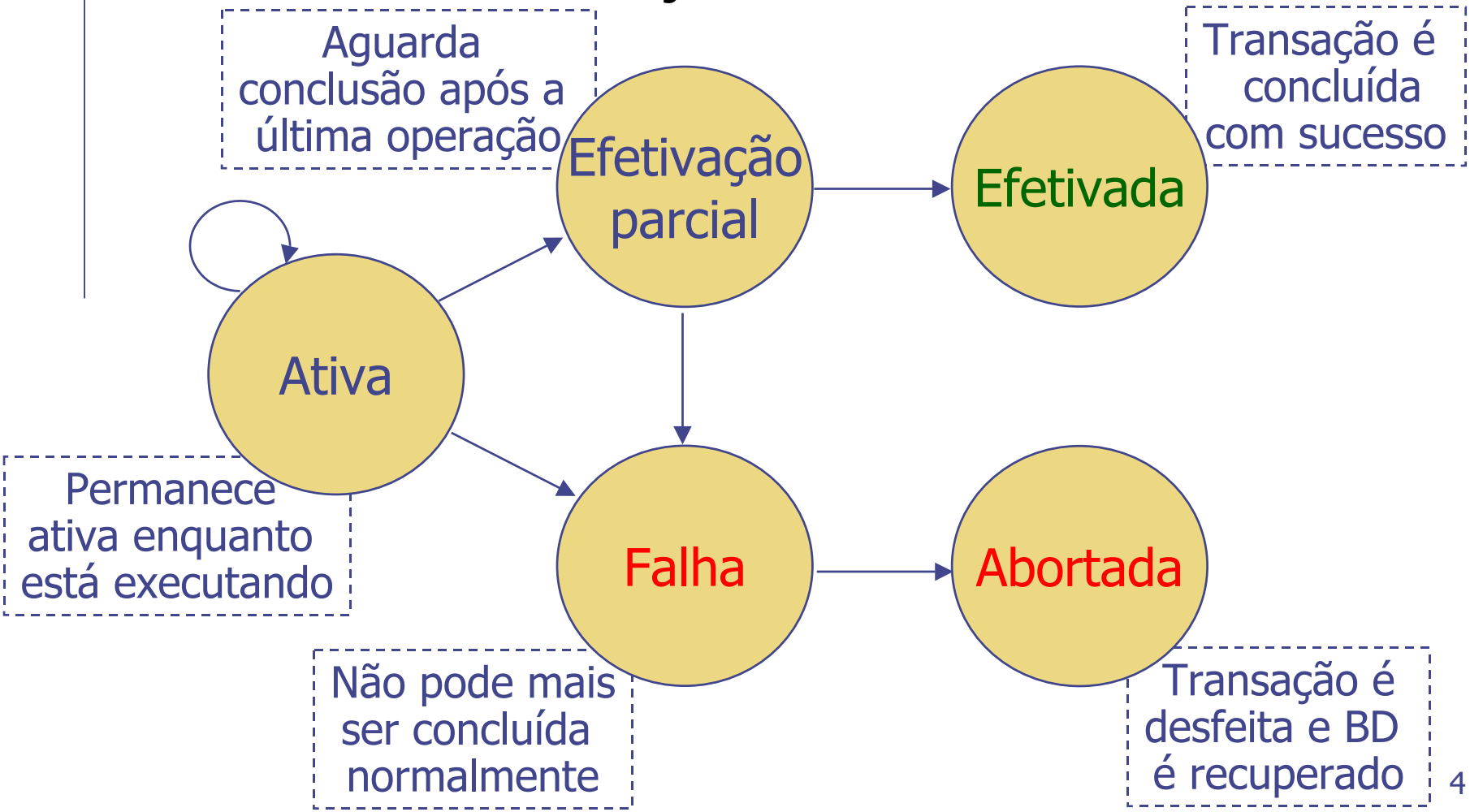
- Transação
  - Atividade indivisível do ponto de vista do usuário
  - Envolve múltiplas operações de acesso a um BD
  - Pode atualizar os dados do BD
- Para executar uma transação com sucesso, precisamos evitar os seguintes problemas:
  - Falhas de hardware e de software podem impedir a que uma transação se complete, interrompendo-a antes da sua efetivação (commit)
  - Uma transação, ao alterar os dados, pode fazer com que outras transações executadas em paralelo (concorrentes) leiam dados inconsistentes (inválidos)

Conceitos

- Exemplos de transações
  - Saque bancário
  - Transferência de fundos
  - Reserva de passagem
  - Venda ao consumidor
  - Comércio entre empresas
  - Cadastro de usuário
  - Mudança de senha
  - Alocação de recurso
  - Controle de estoque
  - etc.

Conceitos

- Estados da Transação



Conceitos

- Tipos de falha
  - Falha de processamento: ocorrida ao processar a transação devido a dois tipos de erro
    - Erro lógico: transação interrompida devido a uma entrada inválida, dado inexistente, overflow, etc.
    - Erro de sistema: transação é abortada ao atingir um estado inadequado (ex.: deadlock)
  - Falha de dispositivo: ocorrida devido a um bloco de disco corrompido ou a um defeito irreparável no HD
  - Falha de sistema: causada por bugs no software de banco de dados ou no sistema operacional, ou por uma falha no hardware

Conceitos

- Uma transação é concluída ao ser efetivada ou abortada pelo sistema
  - Ao ser efetivada, a transação só pode ser desfeita com uma transação de compensação
  - Opções em caso de aborto da transação
    - Reiniciar a transação: se a falha for transitória
    - Matar a transação: se a falha for permanente
- Recuperação de Estado (Rollback)
  - Ocorre quando a transação é abortada
  - Dados retomam os valores que tinham antes do início da transação

Conceitos

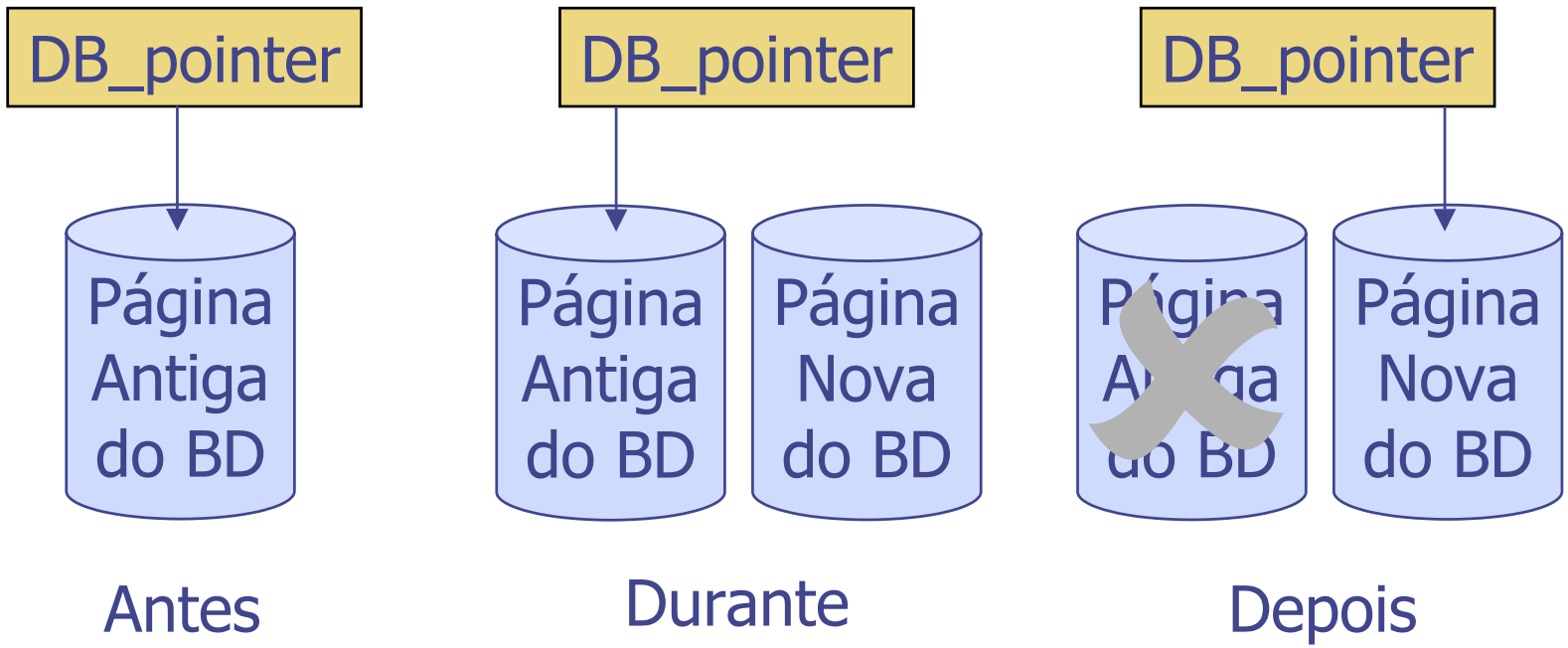
- Recuperação baseada em Logs
  - Todas as operações são registradas
    $<T_i, start>$, $<T_i, commit>$, $<T_i, abort>$
  - No log são identificadas a transação e o dado, além do seu valor antigo e do valor novo
    $<T_i, A_j, V_1, V_2>$
  - Operações podem ser desfeitas em caso de falha
- Recuperação usando Checkpoint
  - Pontos de verificação são criados periodicamente
  - Nos checkpoints, são gravados no disco os dados em memória, os blocos alterados e os registros de log
  - Rollback é feito até o último checkpoint do log

Conceitos

- Recuperação usando Cópia Shadow
  - SGBD mantém um ponteiro para o BD no disco
  - É criada uma nova cópia do BD com as modificações
  - A cópia antiga (ou cópia shadow) é mantida durante a transação
  - No momento da efetivação, o ponteiro passa a apontar para a nova cópia do BD
  - Se transação for abortada, basta manter o ponteiro para a cópia antiga e apagar a cópia nova
  - Otimização: ao invés de copiar o BD inteiro, podemos trabalhar com páginas do BD

Conceitos

- Rollback usando Cópia Shadow (cont.)



Propriedades

- Propriedades das transações
  - **A**tomicidade: todas as operações com os dados são completadas, ou nenhuma o será.
  - **C**onsistência: a execução de transações isoladas (sem concorrência) preserva a integridade do BD
  - **I**solamento: uma transação não toma conhecimento de outras transações concorrentes (ou seja, temos sempre a sensação de que uma transação T_i é anterior ou posterior a T_j, e nunca concorrente)
  - **D**urabilidade: depois de completada a transação, as modificações efetuadas por ela no BD devem ser persistentes, mesmo em caso de falha

Propriedades

- Garantia das propriedades ACID
  - Atomicidade da transação é garantida se as alterações nos valores dos dados forem atômicas (ex.: mudança do ponteiro para cópia shadow)
  - Consistência dos dados é preservada controlando a ordem na qual as operações são efetuadas
  - Isolamento das transações é garantido controlando a concorrência no acesso aos dados (próx. unidade)
  - Durabilidade da transação é garantida gravando as alterações no disco ao concluir a transação

Propriedades

- Exemplo de transação: aplicação financeira

```
Read(Conta);
Conta.Saldo = Conta.Saldo – 100;
Write(Conta);
Read (Aplicação);
Aplicação.Saldo = Aplicação.Saldo + 100;
Write (Aplicação);
```

  - Aplicação das propriedades ACID:
    - Atomicidade: faz todas as operações ou nenhuma
    - Consitência: deve manter o saldo total do cliente
    - Isolamento: controla as operações concorrentes
    - Durabilidade: salva as modificações no BD

Concorrência

- Transações podem ser executadas de modo:
  - Seqüencial: uma transação T_i termina antes do início ou começa após o final de uma transação T_j
  - Concorrente: partes das transações T_i e T_j podem ser processadas simultaneamente
- Concorrência entre transações é necessária para melhorar o desempenho
  - Aproveita o tempo que uma transação permanece bloqueada devido a operações de leitura e escrita no disco para processar outras transações
  - Evita que transações muito longas retardem a execução de transações mais curtas

Concorrência

- Execuções concorrentes implicam que:
  - O processador executa cada transação por um tempo, alternando entre as transações
  - Transações têm sua execução retomada ciclicamente
- Execuções concorrentes precisam garantir as propriedades de consistência e isolamento
  - A definição de escalas de execução (schedules) identifica as operações que podem ser efetuadas concorrentemente sem afetar a consistência do BD
  - Controle de concorrência (visto na próxima unidade) impede que um dado que está sendo alterado por uma transação seja usado por uma outra transação

Concorrência

- Exemplo: suponha duas transações T_1 e T_2
  - T_1: Read (Conta);
    Conta.Saldo = Conta.Saldo – 200;
    Write (Conta);
    Read (Aplic);
    Aplic.Saldo = Aplic.Saldo + 200;
    Write (Aplic);
  - T_2: Read (Conta);
    X = Conta.Saldo * 0.5;
    Conta.Saldo = Conta.Saldo – X;
    Write (Conta);
    Read (Aplic);
    Aplic.Saldo = Aplic.Saldo + X;
    Write (Aplic);

Concorrência

- Escalas para execução seqüenciais do exemplo
  - Suponha que o cliente tem um saldo total de R$2000, sendo R$1000 na conta e R$1000 em aplicações
  - Se T_1 for executada antes de T_2 teremos ao final
    - Conta.Saldo = R$ 400
    - Aplic.Saldo = R$ 1600
  - Já se T_2 for executada antes de T_1, teremos
    - Conta.Saldo = R$ 300
    - Aplic.Saldo = R$ 1700
  - O saldo total é mantido em R$ 2000 nos dois casos

Concorrência

- Exemplo de escala concorrente equivalente à execução seqüencial $T_1 \rightarrow T_2$

T_1	T_2
Read (Conta); Conta.Saldo = Conta.Saldo – 200; Write (Conta);	Bloqueado
Bloqueado	Read (Conta); X = Conta.Saldo * 0.5; Conta.Saldo = Conta.Saldo - X; Write (Conta);
Read (Aplic); Aplic.Saldo = Aplic.Saldo + 200; Write (Aplic);	Bloqueado
Bloqueado	Read (Aplic); Aplic.Saldo = Aplic.Saldo + X; Write (Aplic);

Concorrência

- Exemplo de escala concorrente que não respeita o isolamento → saldo total não é preservado

T_1	T_2
Read (Conta); Conta.Saldo = Conta.Saldo – 200;	Bloqueado
Bloqueado	Read (Conta); X = Conta.Saldo * 0.5; Conta.Saldo = Conta.Saldo - X; Write (Conta);
Write (Conta); Read (Aplic); Aplic.Saldo = Aplic.Saldo + 200; Write (Aplic);	Bloqueado
Bloqueado	Read (Aplic); Aplic.Saldo = Aplic.Saldo + X; Write (Aplic);

Concorrência

- A recuperação de estado é dificultada quando há concorrência entre transações
  - Na escala concorrente do slide 17, se T_1 falha, T_2 tem que ser refeita, pois leu um valor modificado por T_1
  - Nestes casos, a recuperação é feita em cascata → cancelar as operações que leram valores inválidos
  - Rollback em cascata pode ser evitado obrigando que T_2 leia o valor escrito por T_1 só após o commit de T_1
  - Se a efetivação de T_2 ocorrer antes do commit de T_1 será impossível fazer o rollback em cascata

Serialização

- Isolamento obriga que uma escala concorrente seja equivalente a uma escala seqüencial
- Uma escala concorrente equivalente a uma escala seqüencial é dita serializável
  - Considere apenas as operações de leitura e escrita para verificar a serialização
  - As transações podem executar operações com os dados em memória entre operações **Read** e **Write**
- Formas de serialização de escalas de execução
  - Serialização de Conflito
  - Serialização de Visão

Serialização

- Serialização de Conflito
  - Instruções das transações T_i e T_j são conflitantes se ambas acessam um mesmo dado Q e se pelo menos uma delas modifica o valor de Q

Instrução de T_i	Instrução de T_j	Há conflito?
Read(Q)	**Read**(Q)	Não
Read(Q)	**Write**(Q)	Sim
Write(Q)	**Read**(Q)	Sim
Write(Q)	**Write**(Q)	Sim

  - Se duas instruções não são conflitantes, sua ordem pode ser trocada na escala

Serialização

- A escala do slide 17 é equivalente em conflito à escala seqüencial $T_1 \rightarrow T_2$
- A escala do slide 18 não é equivalente em conflito a nenhuma escala seqüencial

T_1	T_2
Read (Conta); Write (Conta);	Bloqueado
Bloqueado	Read (Conta); Write (Conta);
Read (Aplic); Write (Aplic);	Bloqueado
Bloqueado	Read (Aplic); Write (Aplic);

T_1	T_2
Read (Conta);	Bloqueado
Bloqueado	Read (Conta); Write (Conta);
Write (Conta); Read (Aplic); Write (Aplic);	Bloqueado
Bloqueado	Read (Aplic); Write (Aplic);

Serialização

- Serialização de Visão
  - As seguintes regras devem ser respeitadas para que duas escalas S e S’ sejam equivalentes em visão
    - Se na escala S a transação T_i lê o valor inicial de um dado, o mesmo deve ocorrer na escala S’
    - Se na escala S a transação T_i lê o valor de um dado escrito por T_j, o mesmo deve ocorrer em S’
    - Se na escala S a transação T_i escreve o valor final de Q, o mesmo deve ocorrer em S’

Serialização

- Escalas serializáveis em conflito são sempre serializáveis em visão, mas não o contrário
  - O exemplo abaixo é serializável em visão, mas não é serializável em conflito
  - Isto ocorre sempre que são feitas escritas às cegas (blind writes)

<table>
<tr><th>T_3</th><th>T_4</th><th>T_5</th></tr>
<tr><td>Read (Conta);</td><td>Bloqueado</td><td rowspan="3">Bloqueado</td></tr>
<tr><td>Bloqueado</td><td>Write (Conta);</td></tr>
<tr><td>Write (Conta);</td><td rowspan="2">Bloqueado</td></tr>
<tr><td>Bloqueado</td><td>Write (Conta);</td></tr>
</table>

Serialização

- Equivalência por serialização não é precisa
  - Escalas concorrentes equivalentes a seqüenciais podem não ser serializáveis em conflito nem em visão
  - Neste caso, precisamos analisar todas as operações

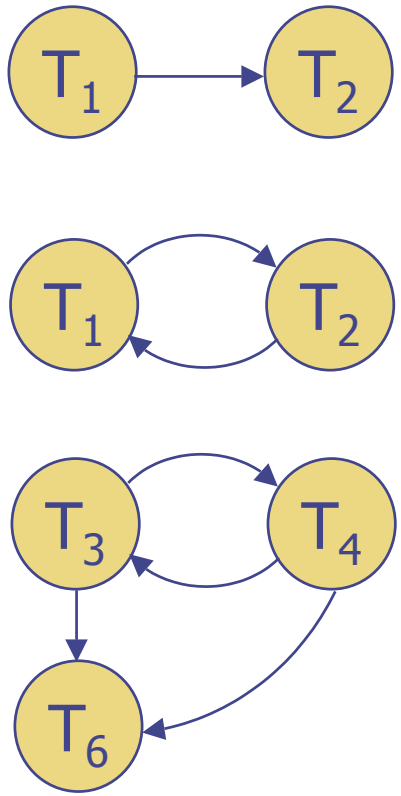
T_1	T_6
Read (Conta); Conta.Saldo = Conta.Saldo – 200; Write (Conta);	Bloqueado
Bloqueado	Read (Aplic); Aplic.Saldo = Aplic.Saldo – 100; Write (Aplic);
Read (Aplic); Aplic.Saldo = Aplic.Saldo + 200; Write (Aplic);	Bloqueado
Bloqueado	Read (Conta); Conta.Saldo = Conta.Saldo + 100; Write (Conta);

Serialização

- Testes de serialização com grafo de precedência
- Teste de serialização de conflito
  - Os nós representam as transações
  - Os arcos indicam a precedência de ações conflitantes
    - T_i executa Write(Q) antes de T_j executar Read(Q)
    - T_i executa Read(Q) antes de T_j executar Write(Q)
    - T_i executa Write(Q) antes de T_j executar Write(Q)
  - Escala é serializável em conflito se o grafo for acíclico
- Teste de serialização de visão
  - Arcos são rotulados, e não devem estar em ciclo
  - Teste complexo e oneroso (não-polinomial)

Serialização

- Exemplos de teste de serialização de conflito
  - Grafo da escala do slide 17
    - Serializável em conflito
  - Grafo da escala do slide 18
    - Não-serializável em conflito (nem em visão)
  - Grafo da escala do slide 24
    - Não-serializável em conflito (mas serializável em visão)



Serialização

- Serialização, Recuperação e Isolamento
  - Escalas devem ser serializáveis em conflito ou visão
  - Alguns SGBD aceitam escalas serializáveis em visão, mesmo que haja conflito; outros permitem apenas escalas serializáveis em conflito
  - SGBDs podem ser configurados para aceitar menores níveis de consistência (ex.: permitir leitura de dados ainda não efetivados)
  - Escalas devem ser recuperáveis, preferencialmente sem a necessidade de recuperação em cascata
  - Quanto maior a concorrência, maior a possibilidade de termos que fazer recuperação em cascata
  - Concorrência pode ser controlada (próxima unidade)