Banco de Dados

Pós-Graduação em Ciência da Computação

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

Transações

Transações

- Conceito de transação
- Estado da transação
- Execuções simultâneas
- Seriação
- Facilidade de recuperação
- Implementação do isolamento
- Definição de transação na SQL
- Testando a seriação

Conceito de transações

- Uma transação é uma unidade da execução de programa que acessa e possivelmente atualiza vários itens de dados.
- Uma transação precisa ver um banco de dados consistente.
- Durante a execução da transação, o banco de dados pode ser temporariamente inconsistente.
- Quando a transação é completada com sucesso (é confirmada), o banco de dados precisa ser consistente.
- Após a confirmação da transação, as mudanças que ele faz no banco de dados persistem, mesmo se houver falhas de sistema.
- Várias transações podem ser executadas em paralelo.
- Dois problemas principais para resolver:
  - Falhas de vários tipos, como falhas de hardware e quedas de sistema
  - Execução concorrente de múltiplas transações

Propriedades ACID

- Uma transação é unidade da execução do programa que acessa e possivelmente atualiza vários itens de dados
- Para preservar a integridade dos dados, o banco deve assegurar:
  - **Atomicidade** . Ou todas as operações da transação são refletidas corretamente no banco de dados ou nenhuma delas é.
  - **Consistência** . A execução de uma transação isolada preserva a consistência do banco de dados.
  - **Isolamento** . Embora várias transações possam ser executadas simultaneamente, cada transação precisa estar desinformada das outras transações que estão sendo executadas ao mesmo tempo. Os resultados da transação intermediária precisam estar ocultos das outras transações sendo executadas simultaneamente.
    - Ou seja, para cada par de transações, Ti e Tj, parece para Ti que Tj, terminou a execução antes que Ti começasse ou Tj iniciou a execução depois que Ti terminou.
  - **Durabilidade** . Depois que uma transação for completada com sucesso, as mudanças que ela fez ao banco de dados persistem, mesmo que existem falhas no sistema.

Operações de acesso

- O Acesso ao banco de dados é obtido pelas seguintes operações:
  - read(X) – que transfere o item de dados X do banco de dados para um buffer local alocado à transação que executou a operação **read**
  - write(X) – que transfere o item de dados X do buffer local da transação que executou a **write** de volta ao banco de dados
- Em um sistema de banco de dados real, a operação write (escrita) não resulta necessariamente na atualização imediata dos dados no disco

Exemplo de transferência de fundos

- Transação para transferir $ 50 da conta A para a conta B:
  1. read(A)
  2. A := A – 50
  3. write(A)
  4. read(B)
  5. B := B + 50
  6. write(B)
- Requisito de atomicidade — Se a transação falhar após a etapa 3 e antes da etapa 6, o sistema deve garantir que suas atualizações não sejam refletidas no banco de dados, ou uma inconsistência irá resultar.
- Requisito de consistência — A soma de A e B é inalterada pela execução da transação.

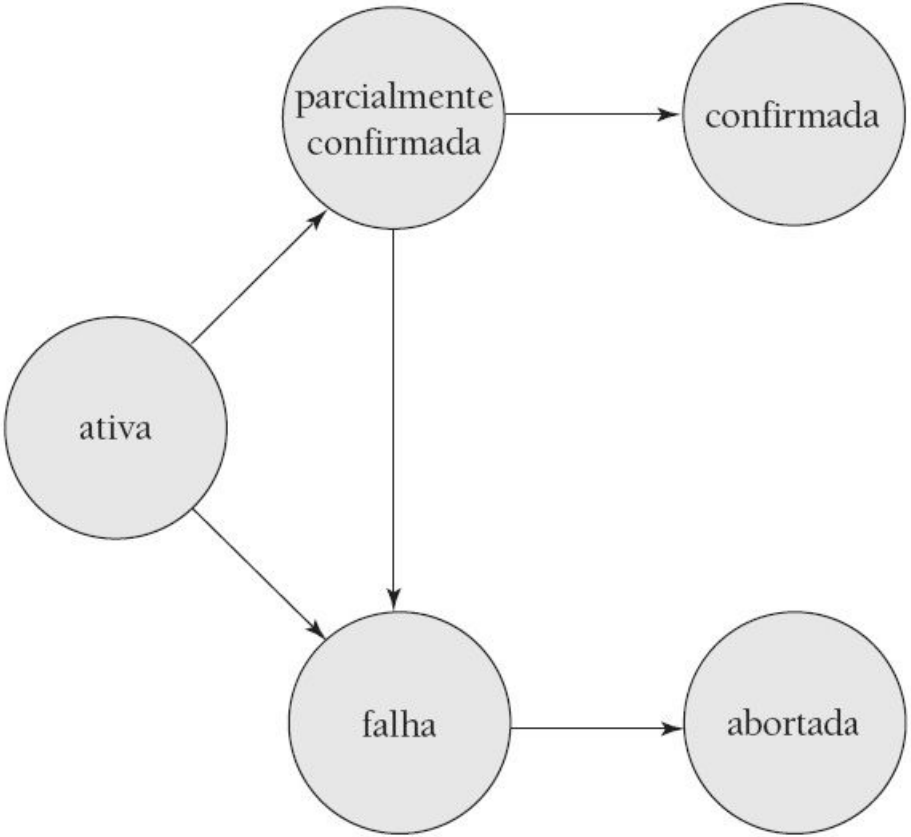
Exemplos de transferência de fundos (cont.)

- Requisito de isolamento — Se entre as etapas 3 e 6, outra transação receber permissão de acessar o banco de dados parcialmente atualizado, ele verá um banco de dados inconsistente (a soma A + B será menor do que deveria ser).
  - Isso pode ser trivialmente assegurado executando transações serialmente (serializadas), ou seja, uma após outra. Entretanto, executar múltiplas transações simultaneamente oferece vantagens significativas, como veremos mais adiante.
- Requisito de durabilidade — Quando o usuário é notificado de que a transação está concluída (ou seja, a transação dos $ 50 ocorreu), as atualizações no banco de dados pela transação precisam persistir apesar de falhas.

Estado da transação

- **Ativa** – O estado inicial; a transação permanece nesse estado enquanto está executando
- **Parcialmente confirmada** – Depois que a instrução final foi executada
- **Falha** – Depois da descoberta de que a execução normal não pode mais prosseguir
- **Abortada** – Depois que a transação foi revertida e o banco de dados foi restaurado ao seu estado anterior ao início da transação. Duas opções após ter sido abortada:
  - Reiniciar a transação; pode ser feito apenas se não houver qualquer erro lógico interno
  - Excluir a transação – erro lógico interno
- **Confirmada** (Commited)– Após o término bem sucedido

Estado da transação (cont.)



Execuções simultâneas (Concorrentes)

- Várias transações podem ser executadas simultaneamente no sistema. As vantagens são:
  - **Melhor utilização do processador e do disco** , levando a um melhor throughput de transação: uma transação pode estar usando a CPU enquanto outra está lendo ou escrevendo no disco
  - **Tempo de médio de resposta reduzido para transações** : as transações curtas não precisam esperar atrás das longas
- **Esquemas de controle de concorrência** – mecanismos para obter isolamento; ou seja, para controlar a interação entre as transações concorrentes a fim de evitar que elas destruam a consistência do banco de dados
- **Throughput** : número de transações executadas em determinada quantidade de tempo

Escalonamento

- Escalonamento – Sequências de instruções que especificam a ordem cronológica em que as instruções das transações concorrentes são executadas
  - Um escalonamento para um conjunto de transações precisa consistir em todas as instruções dessas transações
  - Precisam preservar a ordem em que as instruções aparecem em cada transação individual
- Uma transação que completa com sucesso sua execução terá uma instrução commit como a última instrução (será omitida se for óbvia)
- Uma transação que não completa com sucesso sua execução terá instrução abort como a última instrução (será omitida se for óbvia)

Escalonamento 1

- Suponha que T1 transfere $ 50 de A para B e T2 transfere 10% do saldo de A para B.
- A seguir está um escalonamento serial em que T1 é seguido de T2 .

T_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)

Escalonamento 2

- Sejam T_1 e T_2 as transações definidas anteriormente. O escalonamento a seguir não é um escalonamento serial, mas é equivalente ao escalonamento 1.

T_1	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)

Escalonamento 3

➢ O seguinte escalonamento concorrente não preserva o valor da soma A + B.

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

Seriação/Serialização

- Suposição básica – Cada transação preserva a consistência do banco de dados
- Portanto, a execução serial de um conjunto de transações preserva a consistência do banco de dados
- Um escalonamento (possivelmente simultâneo) é serializável se for equivalente a um escalonamento serial. Diferentes formas de equivalência de escalonamento ensejam as noções de:
  1. Seriação de conflito
  2. Seriação de view
- Ignoramos as operações exceto read e write, e consideramos que as transações podem realizar cálculos arbitrários sobre dados em buffers locais entre reads e writes. Nossos escalonamentos simplificados consistem apenas em instruções read e write.

Instruções conflitantes

- As instruções Ii e Ij das transações Ti e Tj respectivamente, estão em conflito se e somente se algum item Q acessado por Ii e por Ij e pelo menos uma destas instruções escreveram Q.
  1. Ii = read(Q), Ij = read(Q). Ii e Ij não estão em conflito
  2. Ii = read(Q), Ij = write(Q). Estão em conflito
  3. Ii = write(Q), Ij = read(Q). Estão em conflito
  4. Ii = write(Q), Ij = write(Q). Estão em conflito
- Intuitivamente, um conflito entre Ii e Ij força uma ordem temporal (lógica) entre eles. Se Ii e Ij são consecutivos em um escalonamento e não entram em conflito, seus resultados permanecem inalterados mesmo se tiverem sido trocados no escalonamento.

Seriação de conflito

- Se um escalonamento S puder ser transformado em um escalonamento S′ por uma série de trocas de instruções não conflitantes, dizemos que S e S′ são equivalentes em conflito.
- Dizemos que um escalonamento S é serial de conflito se ele for equivalente em conflito a um escalonamento serial

Seriação de conflito (cont.)

- O Escalonamento 3 abaixo pode ser transformado em Escalonamento 1 (slide a seguir), um escalonamento onde T_2 segue T_1 , por uma série de trocas de instruções não conflitantes. Portanto, o escalonamento 3 é serial de conflito.

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

Seriação de conflito (cont.)

- A instrução write (A) de T_2 não conflita com a instrução read(B) de T_1
- Continuando a inverter instruções não conflitantes:
  - Trocar a instrução read(B) de T_1 pela instrução read(A) de T_2
  - Trocar a instrução write(B) de T_1 pela instrução write(A) de T_2
  - Trocar a instrução write(B) de T_1 pela instrução read(A) de T_2

Escalonamento 1

T_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)

Seriação de conflito (cont.)

- Exemplo de um escalonamento que não é serial de conflito:
- Não podemos trocar instruções no escalonamento acima para obter o escalonamento serial < T_3, T_4 >, ou o escalonamento serial < T_4 , T_3>.

T_3	T_4
read(**Q**) write(**Q**)	write(**Q**)

Seriação de Visão

- Sejam S e S´ dois escalonamento com o mesmo conjunto de transações. S e S´ são equivalentes em visão se as três condições a seguir forem satisfeitas:

  1. **Para cada item de dados Q, se a transação Ti ler o valor inicial de Q no escalonamento S, então, Ti precisa, no escalonamento S´, também ler o valor inicial de Q.**
  2. **Para cada item de dados Q, se a transação Ti executar read(Q) no escalonamento S, e se esse valor pela transação Tj (se houver), então a transação Ti, no escalonamento S', também precisa ler o valor de Q que foi produzido pela transação Tj.**
  3. **Para cada item de dados Q, a transação (se houver) que realiza a operação write(Q) final no escalonamento S precisa realizar a operação write(Q) final no escalonamento S'.**

- As condições 1 e 2 garantem que cada transação lê os mesmos valores nos dois schedules e, portanto, realiza a mesma computação. A condição 3, junto com as condições 1 e 2, garante que os dois schedules resultam no mesmo estado final
- Como podemos ver, a equivalência em visão também é baseada unicamente em reads e writes isolados.

Seriação de Visão (cont.)

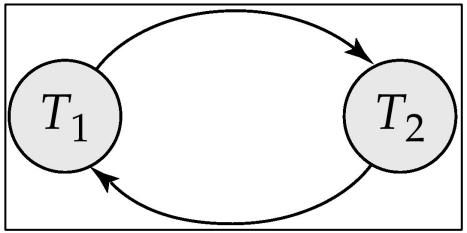
- Um escalonamento S é serial de visão se ele for equivalente em visão a um escalonamento serial
- Todo escalonamento serial de conflito também é serial de visão
- A seguir está um escalonamento que é serial de visão mas não serial de conflito

T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

- Todo escalonamento serial de visão que não é serial de conflito possui escritas cegas

Testando a seriação

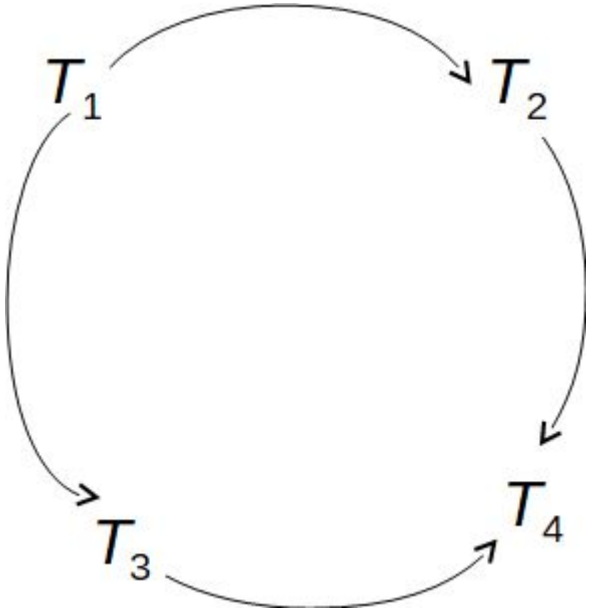
- Considere um escalonamento de um conjunto de transações T_1, T_2, ..., Tn
- Grafo de precedência — Um grafo direcionado onde os vértices são as transações (nomes)
- Desenhamos um arco (aresta) de Ti até Tj se a transação entra em conflito e Ti acessou o item de dados no qual surgiu o conflito anteriormente
- Podemos rotular o arco pelo item que foi acessado
- Exemplo 1



Exemplo de escalonamento (Escalonamento A)

T1	T2	T3	T4	T5
read(Y) read(Z) read(U) read(U) write(U)	read(X) read(Y) write(Y)	 write(Z)	 read(Y) write(Y) read(Z) write(Z)	 read(V) read(W) read(W)

Gráfo de precedência para o escalonamento A



Teste para seriação de conflito

- Um escalonamento é serial de conflito se e somente se seu grafo de precedência for acíclico
- Existem algoritmos de detecção de ciclo que assumem a ordem cronológica n2, onde n é o número de vértices no grafo. (Algoritmos melhores assumem a ordem n + e, onde e é o número de arestas.)
- Se o grafo de precedência for acíclico, a ordem de seriação pode ser obtida por meio da classificação topológica do grafo. Essa é a ordem linear com a ordem parcial do grafo. Por exemplo, uma ordem de seriação para o escalonamento A seria T_5 T_1 T_3 T_2 T_4 .

Teste para seriação de View

- O teste do grafo de precedência precisa ser modificado para se aplicar a um teste para seriação de view
- O problema de verificar se um escalonamento é serial de view entra na classe dos problemas não procedurais completos. Portanto, embora a existência de um algoritmo eficiente seja improvável, os algoritmos práticos que simplesmente verificam algumas condições suficientes para a seriação de view ainda podem ser usados.

Facilidade de recuperação

- É necessário tratar do efeito das falhas de transação nas transações sendo executadas simultaneamente.
- Escalonamento recuperável — Se uma transação Tj lê um item de dados anteriormente escrito por uma transação Ti , então a operação commit de Ti aparece antes da operação commit de Tj.
- O escalonamento (Escalonamento 11) não é recuperável se T_9 for confirmado imediatamente após o read

T_8	T_9
read(A)	
write(A)	
	read(A)
read(B)	

- Se T_8 abortasse, T_9 teria lido (e possivelmente mostrado ao usuário) um estado inconsistente. Portanto, o banco de dados precisa garantir que Escalonamentos sejam recuperáveis

Facilidade de recuperação (cont.)

- Rollback em cascata – Uma única falha de transação leva a uma série de rollbacks de transação. Considere o seguinte Escalonamento onde nenhuma das transações ainda foi confirmada (portanto, o Escalonamento é recuperável)

T_{10}	T_{11}	T_{12}
read(A)		
read(B)		
write(A)		
	read(A)	
	write(A)	
		read(A)

- Se T_{10} falhar, T_{11} e T_{12} também precisam ser revertidos
- Pode chegar a desfazer uma quantidade de trabalho significativa

Facilidade de recuperação (cont.)

- Escalonamentos não em cascata — Rollbacks em cascata não podem ocorrer; para cada par de transações Ti e Tj tal que Tj leia um item de dados escrito anteriormente por Ti, a operação commit de Ti apareça antes da operação read de Tj.
- Todo escalonamento não em cascata também é recuperável
- É desejável restringir os escalonamentos aos não em cascata

Aspectos da implementação

- Um banco de dados (SGBD) precisa fornecer um mecanismo que garanta que todos os escalonamentos possíveis sejam seriais de conflito ou de view, e sejam recuperáveis e, preferivelmente, não em cascata
- Uma política em que apenas uma transação pode ser executada de cada vez gera escalonamentos seriais, mas fornece um menor grau de concorrência
- Os esquemas de controle de concorrência conciliam entre a quantidade de concorrência que permitem e a quantidade de sobrecarga a que ficam sujeitos

Testes de controle de serialização

- Testar a seriação de um escalonamento após ele ter sido executado é um pouco tarde demais!
- Objetivo – Desenvolver os protocolos de controle de concorrência que garantirão a capacidade de seriação. Eles normalmente não examinam o gráfo de precedência enquanto está sendo criado, em vez disso, um protocolo imporá uma regra que evita escalonamentos não serializáveis.
- Os testes para a seriação ajudam a entender por que um protocolo de controle de concorrência está correto