CEFET/RJ

TRANSAÇÕES

Eduardo Ogasawara
eogasawara@ieee.org
https://eic.cefet-rj.br/~eogasawara

Conceito de transação

- 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 sistema
  - Execução simultânea de múltiplas transações

Propriedades ACID

- Uma transação é unidade da execução do programa que acessa e possivelmente atualiza vários itens de dados
- 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, $\boldsymbol{T_i}$ e $\boldsymbol{T_j}$, para $\boldsymbol{T_i}$ parece que $\boldsymbol{T_j}$ terminou a execução antes que $\boldsymbol{T_i}$ começasse ou $\boldsymbol{T_j}$ iniciou a execução depois que $\boldsymbol{T_i}$ terminasse
- 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

Exemplo de transferência de fundos

- Transação para transferir US$ 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

Exemplo 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, 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 US$ 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
- Confirmada
  - Após o término bem sucedido

Estado da transação (cont.)

Execuções simultâneas

- Vantagens quando as transações são executadas simultaneamente no sistema:
  - 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

Escalonamento (Schedule)

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

Schedule 1

- Suponha que T_1 transfira US$ 50 de A para B e T_2 transfira 10% do saldo de A para B
  - A seguir está um escalonamento serial em que T_1 é seguido de T_2

T_1	T_2
read(A) A ← A − 50 write(A) read(B) B ← B + 50 write(B)	 read(A) temp ← A * 0.1 temp ← A - temp write(A) read(B) B ← B + temp write(B)

Schedule 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 temp ← A - temp write(A)
read(B) B ← B + 50 write(B)	
	read(B) B ← B + temp write(B)

Schedule 3

- Sejam T_1 e T_2 as transações definidas anteriormente
  - 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 temp ← A - temp write(A) read(B)
write(A) read(B) B ← B + 50 write(B)	
	B ← B + temp write(B)

Seriação

- 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 seriável se for equivalente a um escalonamento serial.
- Diferentes formas de equivalência de escalonamento ensejam as noções de:
  - 1. Seriação de conflito
  - 2. Seriação de visão (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 $\boldsymbol{l_i}$ e $\boldsymbol{l_j}$ das transações $\boldsymbol{T_i}$ e $\boldsymbol{T_j}$, respectivamente, estão em conflito se e somente se algum item $\boldsymbol{Q}$ acessado por $\boldsymbol{l_i}$ e por $\boldsymbol{l_j}$ e pelo menos uma destas instruções escreveram $\boldsymbol{Q}$
  - $\boldsymbol{l_i} \leftarrow \boldsymbol{read}(\boldsymbol{Q})$, $\boldsymbol{l_j} \leftarrow \boldsymbol{read}(\boldsymbol{Q})$. $\boldsymbol{l_i}$ e $\boldsymbol{l_j}$ não estão em conflito
  - $\boldsymbol{l_i} \leftarrow \boldsymbol{read}(\boldsymbol{Q})$, $\boldsymbol{l_j} \leftarrow \boldsymbol{write}(\boldsymbol{Q})$. Estão em conflito
  - $\boldsymbol{l_i} \leftarrow \boldsymbol{write}(\boldsymbol{Q})$, $\boldsymbol{l_j} \leftarrow \boldsymbol{read}(\boldsymbol{Q})$. Estão em conflito
  - $\boldsymbol{l_i} \leftarrow \boldsymbol{write}(\boldsymbol{Q})$, $\boldsymbol{l_j} \leftarrow \boldsymbol{write}(\boldsymbol{Q})$. Estão em conflito
- Intuitivamente, um conflito entre $\boldsymbol{l_i}$ e $\boldsymbol{l_j}$ força uma ordem temporal (lógica) entre eles
- Se $\boldsymbol{l_i}$ e $\boldsymbol{l_j}$ 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 por conflito se ele for equivalente em conflito a um escalonamento serial S'

Seriação de conflito (cont.)

- O escalonamento A pode ser transformado no escalonamento B
  - Note que em B, T_2 segue T_1, por uma série de trocas de instruções não conflitantes
  - Portanto, o schedule A é serial de conflito para $< T_1, T_2 >$

Escalonamento A

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

Escalonamento B

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

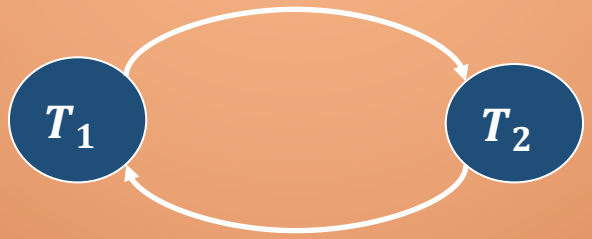
Seriação de conflito (cont.)

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

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

Testando a seriação

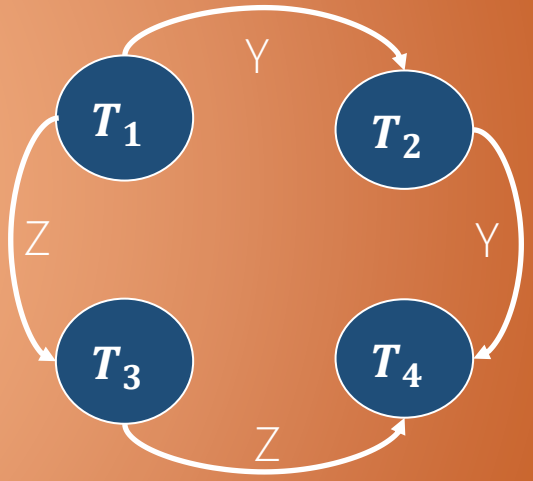
- Considere um escalonamento de um conjunto de transações T_1, T_2, ..., T_n
- Grafo de precedência
  - Um grafo direcionado onde os vértices são as transações (nomes)
  - Desenhamos um arco de T_i até T_j se a transação T_j tiver uma instrução conflitante com T_i, onde:
    - T_i manipula o dado antes de T_j
    - T_j tem T_i como transação mais próxima a ter conflito com ela
  - Podemos rotular o arco pelo item que foi acessado



Exemplo de escalonamento

Escalonamento C

T_1	T_2	T_3	T_4	T_5
	read(X)			
read(Y)				
read(Z)				
				read(V)
				read(W)
				read(W)
	read(Y)			
	write(Y)			
		write(Z)		
read(U)				
			read(Y)	
			write(Y)	
			read(Z)	
			write(Z)	
read(U)				
write(U)				



Teste para seriação de conflito

- Um escalonamento é serial de conflito se e somente se seu grafo de precedência for acíclico
- 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 $\boldsymbol{C}$ seria
    $< \boldsymbol{T_5}, \boldsymbol{T_1}, \boldsymbol{T_3}, \boldsymbol{T_2}, \boldsymbol{T_4} >$

Seriação de visão

- Sejam S e S' dois escalonamentos 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 $\boldsymbol{T_i}$ fizer o $read(Q)$ no começo do escalonamento S, então, $\boldsymbol{T_i}$ precisa também fazer o $read(Q)$ no começo do escalonamento S'
  - 2. Para cada item de dados Q, se a transação $\boldsymbol{T_i}$ fizer o $read(Q)$ no escalonamento S e se esse valor tiver sido produzido, $write(Q)$, pela transação $\boldsymbol{T_j}$, então a transação $\boldsymbol{T_i}$, no escalonamento S', $\boldsymbol{T_i}$ precisa executar o $read(Q)$ que tiver sido produzido pela transação $\boldsymbol{T_j}$
  - 3. Para cada item de dados Q, se a transação $\boldsymbol{T_i}$ fizer o $write(Q)$ final no escalonamento S, então, $\boldsymbol{T_i}$ precisa também fazer o $write(Q)$ no final do escalonamento S'
- Como podemos ver, a equivalência em visão também é baseada unicamente em $reads$ e $\boldsymbol{writes}$ isolados

Seriação de view (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 tem-se um escalonamento que é serial de visão mas não serial de conflito
  - Todo schedule serial de visão que não é serial de conflito possui escritas cegas

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

Teste para seriação de visão

- O problema de verificar se um schedule é serial de visão entra na classe dos problemas não procedurais completos (NP-hard)
- Embora a existência de um algoritmo eficiente seja improvável, os algoritmos práticos que simplesmente verificam algumas condições suficientes para a seriação de visão ainda podem ser usados

Outras noções de seriação

- O escalonamento abaixo produz o mesmo resultado do escalonamento serial $< T_1, T_5 >$, mas não é equivalente em conflito ou equivalente em visão a ele
- Determinar essa equivalência exige análise de operações diferentes de $read$ e $write$

T_1	T_5
read(A) A ← A - 50 write(A)	
	read(B) B ← B - 10 write(B)
read(B) B ← B + 50 write(B)	
	read(A) A ← A + 10 write(A)

Facilidade de recuperação

- É necessário tratar do efeito das falhas de transação nas transações sendo executadas simultaneamente
- Schedule recuperável
  - Se uma transação T_j lê um item de dados anteriormente escrito por uma transação T_i, então a operação $commit$ de T_i aparece antes da operação $commit$ de T_j
  - O escalonamento abaixo não é recuperável se T_9 for confirmado imediatamente após o $read$
  - Se T_8 abortasse, T_9 teria lido (e possivelmente mostrado ao usuário) um estado inconsistente. Portanto, o banco de dados precisa garantir que os escalonamentos sejam recuperáveis

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

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

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

Testes de controle e de seriação

- Testar a seriação de um schedule após ele ter sido executado é um pouco tarde de mais...
- Objetivo:
  - Desenvolver os protocolos de controle de concorrência que garantam a capacidade de seriação
- Eles normalmente não examinam o grafo de precedência enquanto está sendo criado; em vez disso, um protocolo deve impor regras que evitem schedules não serializáveis
- Os testes para a seriação ajudam a entender por que um protocolo de controle de concorrência está correto

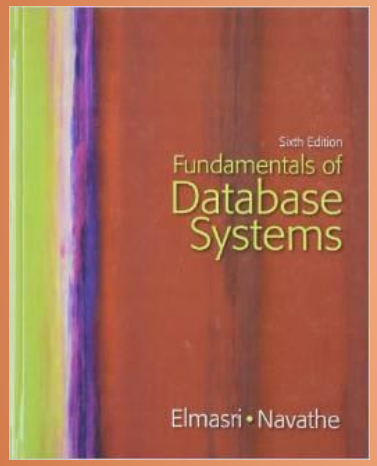
Definição de transação na SQL

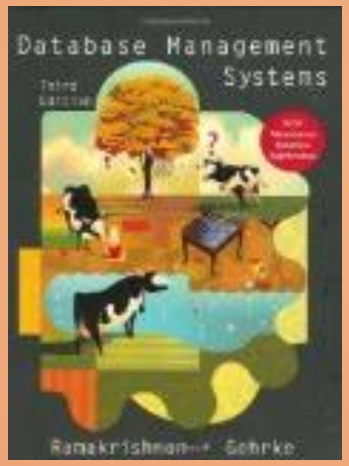
- A linguagem de manipulação de dados precisa incluir uma construção para especificar o conjunto de ações que compõem uma transação
- Na SQL, uma transação começa implicitamente
- Uma transação na SQL termina por:
  - Commit work confirma a transação atual e inicia uma nova transação
  - Rollback work faz com que a transação atual seja abortada
- Níveis de consistência especificados pela SQL-92:
  - Serializable — padrão
  - Repeatable read
  - Read committed
  - Read uncommitted

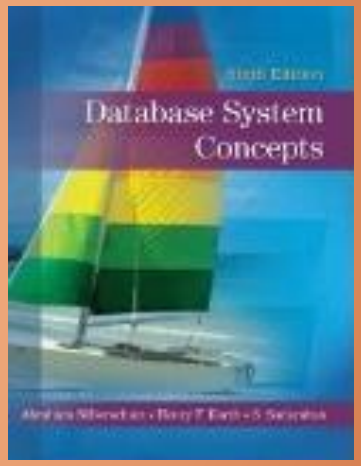
Níveis de consistência na SQL-92

- Serializable
  - Padrão
- Repeatable read
  - Apenas registros confirmados podem ser lidos
  - reads repetidos do mesmo registro precisam retornar o mesmo valor
  - Entretanto, uma transação pode não ser seriável - ela pode encontrar alguns registros inseridos por uma transação mas não encontrar outros
- Read committed
  - Apenas registros confirmados podem ser lidos,
  - reads sucessivos do registro podem retornar valores diferentes (mas confirmados)
- Read uncommitted
  - Mesmo registros não confirmados podem ser lidos
- Graus de consistência mais baixos são úteis para coletar informações aproximadas sobre o banco de dados; por exemplo, estatística para otimizador de consulta

Referências







A maioria dos slides apresenta exemplos do livro Database System Concepts