Unidade VI
Transação e Concorrência

Curso Ciência da Computação
Bancos de Dados – PUCMinas
Prof. Manoel **Palhares** Moreira

Bibliografia

- NAVATHE, Shamkant B., ELSMARI, Ramez. Sistemas de Bancos de Dados. 6 ed., Addinson Wesley Publishing, 2011. Cpas. 21, 22 e 23.
- GARCIA-MOLINA, Hector, ULLMAN, Jeffrey, WINDOM, Jeninifer. Implementação de sistemas de Banco de Dados. Rio de Janeiro: Campus, 2001.
- KORTH, Henry F., SILBERSCHATZ, A., SUDARSHAM, S. Sistema de bancos de dados. 5ª ed. São Paulo: Editora Campus/Elsevier, 2006.

Objetivo

Conhecer o significado de transação e as operações realizadas em um sistema de gerenciamento de banco de dados para manutenção das propriedades básicas da transação. Conhecer os mecanismos com os quais um sgbd faz a gerência da concorrência entre transações de um ambiente.

Transação

Conceitos iniciais

- Um SGBD será monousuário se somente um usuário pode utilizá-lo de cada vez
- Um SGBD será multiusuário se diversos usuários podem usar o sistema concorrentemente, ou seja ao mesmo tempo

Conceitos iniciais

- Multiprogramação: permite que cada computador execute diversos programas ou processos ao mesmo tempo; ou seja diversos usuários podem ter acesso aos bancos de dados - e usar os sistemas de computador - simultaneamente
- Sistemas operacionais com multiprogramação executam alguns comandos de um processo, depois suspendem esse processo e executam comandos do próximo processo, e assim por diante

Transação

- Uma transação é um programa em execução que forma uma unidade lógica de processamento no banco de dados.
- Uma transação inclui uma ou mais operações de acesso ao banco de dados e engloba operações de inserção, exclusão, alteração ou recuperação (leitura)

Transação

- Transação "é uma unidade de execução de programa que acessa e, possivelmente, atualiza vários itens de dados".
- ... É o resultado da execução de um programa escrito em DML de alto nível ...
- ... É delimitada por declarações da forma *begin transaction* e *end transaction*. A transação consiste em todas as operações realizadas neste intervalo

Transação

- Para exemplificar todos os processos existentes em um SGBD, dizemos que existem dois tipos de transação apenas:
  - ler_item(X)
  - escrever_item(X)
- ou
  - read (x)
  - write (x)

Transação

- Uma transação compreende operações de leitura e escrita para acessos e atualizações do banco de dados.

T_1	T_2
read(X,Y)	read(x)
X=x-n	X=x+m
write(x)	write(x)
Y=y+n	
write(y)	

Tipos possíveis de falhas

- Tipos de falhas:
  - de transação
  - de sistema
  - de mídia
  - por motivo externo

Motivos de falhas

- *crash* ou queda do sistema
- erro de transação (*overflow* do buffer - diversas linhas atualizadas sem comando *commit*)
- erros locais ou condições de execução (a transação é cancelada por alguma falha, ou por não encontrar os dados, por valor de dado: saldo insuficiente, etc)
- imposição do controle de concorrência
- falha na mídia (blocos com defeitos)
- problemas físicos ou catástrofes

Transação

'Uma transação é uma unidade atômica de trabalho que ou estará completa ou não foi realizada'

Para isso o sistema mantém o controle do início e do término da transação:

- - begin_transaction (início da transação)
- - read ou write
- - end_transaction
- - commit_transaction;
- - roolback

Propriedades ACID da transação

- Atomicidade: ou todas as operações da transação são refletidas corretamente no banco de dados ou nenhuma o será
- Consistência: A execução de uma transação isolada (ou seja, sem a execução concorrente de outra transação) preserva a consistência do banco de dados).

Propriedades ACID da transação

- Isolamento: O sistema garante que cada transação não toma conhecimento de outras transações concorrentes no sistema
- Durabilidade: Depois da transação completar-se com sucesso, as mudanças que ela fez no banco de dados persistem, até mesmo se houver falhas no sistema.

Exemplo

Ti transfere $50 da conta A para a conta B

Ti
read(A)
A:=A-50
write(A)
read(B)
B:=B+50
write(B)

Exemplo

Análise da transação:

Read(X) transfere o item de dados *X* do banco de dados para um buffer local alocado à transação que executou a operação de *read*

WRITE(X) transfere o item de dados *X* do buffer local da transação que executou a operação de *write* de volta ao banco de dados

Exemplo

Análise da transação

- Consistência: soma de A+B permanece inalterada. Responsabilidade do programador da aplicação. *Constraints* como facilitador.
- Atomicidade: Supor Va=$1000, Vb=$2000. Valores possíveis após a transação serão $950 e $2050. (conceito de *log;* componente do sgbd responsável pelo processo é o de gerenciamento de transações)

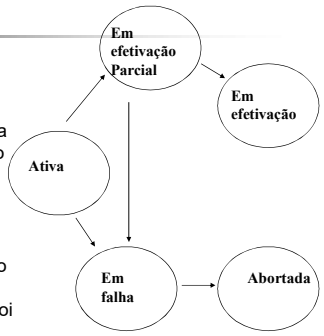
Exemplo

Análise da transação

- Durabilidade: uma vez completada a transação o resultado final permanece mesmo com falhas no sistema. (responsabilidade do gerenciador de recuperação)
- Isolamento: Tratadas isoladamente, duas transações concorrentes podem chegar a valores distintos. Componente responsável gerenciador de concorrência.

Estados da Transação

- Ativa: estado inicial. Permanece nele enquanto estiver executando
- Em efetivação parcial: após a execução da ultima declaração
- Em falha: após descoberto que a execução normal não pode ser executada
- Abortada: a transação foi desfeita e o BD volta ao estado normal
- Em efetivação: a transação foi concluída com sucesso



Estados da Transação

Uma transação é dita concluída se estiver *em efetivação* ou *abortada*

Estados da Transação

Uma transação entra no ***estado de falha*** quando o sistema determina que ela já não pode prosseguir sua execução normal. Então ela precisa ser desfeita e entra no estado abortada. Neste instante o SGBD opta por:

- ***reiniciar a transação***, mas somente se ela foi abortada por problemas de hardware ou software não criado pela lógica interna da transação
- ***matar a transação***, quando houver erro de lógica, aplicação, entrada não adequada, etc..

Implementação de Atomicidade e Durabilidade

- responsabilidade do Gerenciador de Recuperação (subsistema de restauração): suporte à atomicidade e durabilidade
- cópias ***shadow*** (exemplos em editores) + ponteiros db_pointer

Implementação de Isolamento e Consistência

- A consistência geralmente fica a cargo do programador. Implementação de restrições no sgbd
- O isolamento é responsabilidade do Gerenciador de Concorrência

Execuções Concorrentes

- Os sistemas de gerenciamento de banco de dados permitem que diversas transações ocorram concorrentemente
- O mais fácil seria se elas ocorressem de forma seqüencial. Teríamos então problemas:
  - com a otimização da utilização da de operações de I/O e CPU (*throughput* do sistema - número de transações em determinado tempo)
  - transações curtas e longas estão misturadas. O tempo de resposta ficaria comprometido

Execuções Concorrentes

Exemplo: Sejam duas transações T_1 e T_2

- T_1 uma transação que transfere \$50 da Conta *A* para a Conta *B*
- T_2 uma transação que transfere 10% do saldo da conta A para a conta B

Escala 1 - sequencialmente T_1 e T_2

T1	T2
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)

Escala 2 - seqüencialmente T_2 e T_1

T1	T2
	read(A)
	temp:=A*0,1
	A:=A - temp
	write(A)
	read(B)
	B:=B+temp
	write(B)
read(A)	
A:=A - 50	
write(A)	
read(B)	
B:=B+50	
write(B)	

Execuções Concorrentes

- Estas duas escalas são seqüenciais
- Quando várias transações são executadas simultaneamente, a escala correspondente pode já não ser seqüencial
- Várias seqüências de transação são possíveis e não se pode prever qual delas fará a CPU. O número de escalas possíveis para um conjunto de transações *n* é muito maior que *n!*

Escala 3 - uma escala de transação concorrente equivalente a escala 1

T1	T2
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)

Escala 4 - uma escala concorrente sem a preservação da soma *A + B*

T1	T2
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)	

Por que controle de concorrência?

- Por que é necessário o controle de concorrência?

a) Problema da atualização perdida

ocorre quando duas transações que acessam os mesmo itens de dados tiverem suas operações intercaladas de forma que tornem o valor de alguns dos itens do banco de dados incorretos

Por que controle de concorrência?

T_1	T_2
read(x)	
x=x-n	
	read(x)
	x=x+m
	write(x)
write(x)	

Por que controle de concorrência?

b) Problema de atualização temporária (*Dirty Read* ou Leitura Suja)

Ocorre quando uma transação atualiza um item de um banco de dados e a seguir falha por alguma razão e o item atualizado é acessado antes que retorne a seu valor original.

Por que controle de concorrência?

T_1	T_2
read(x)	
x=x-n	
write(x)	
	read(x)
	x=x+m
	write(x)
read(y)	

- Se a transação T_1 falha, X deve retornar a seu valor inicial. Enquanto isso, T_2 lê seu valor 'temporário' e incorreto

Por que controle de concorrência?

c) Problema do sumário incorreto

Se uma transação aplicar uma função agregada para sumário de um número de registros enquanto outra transação está atualizando esses registros, a função agregada deverá calcular alguns valores antes de sua atualização.

Por que controle de concorrência?

T_1	T_2
	sum=0
	read(a)
	sum=sum+a (sum{a[)
	read(b)
	sum=sum+b (sum{b])
read(x)	
x=x-n	
write(x)	
	read(x)
	sum=sum+x (sum[x])
	read(y)
	sum=sum+y (sum[y])
read(y)	
y=y+m	
write(y)	

Por que controle de concorrência?

d) O problema da leitura sem repetição.

Uma transação T_1 lê um item de dados duas vezes e o item foi mudado por uma transação T_2 entre essas leituras. T_1 recebe valores diferentes para essas leituras.

Exemplo típico é assento em vôos; saldo bancário; etc.

Por que controle de concorrência?

Por que a restauração (recuperação) é necessária!

O SGBD garante que:

- todas as operações na transação são completadas com sucesso e seu resultado é gravado no bancos de dados ou
- a transação não terá nenhum efeito
- com outras palavras: um SGBD sempre sai de um estado consistente para outro estado consistente

Serialização

- O sistema de gerenciamento de banco de dados deve controlar a execução concorrente de transações para assegurar que o estado do banco de dados permaneça consistente
- precisamos entender quais escalas de execução podem garantir esta consistência

Serialização

Sejam as transações Ti e Tj, com instruções Ii e Ij em um mesmo item de dado:

- se Ii e Ij são instruções de *read*, a seqüência de execução não importa
- se Ii é uma instrução de *read* e Ij uma instrução de write, a ordem como acontecem as instruções influencia o resultado de Ti e Tj
- o mesmo se Ii for uma instrução de *write* e Ij uma instrução de *read*
- se Ii e Ij são instruções de *write*, a ordem como acontecem não afeta as transações Ti e Tj, mas o resultado final é sempre o da ultima transação executada

Conflito de transações

Dizemos que duas instruções Ii e Ij entram em **conflito** caso sejam operações pertencentes a transações diferentes, agindo sobre um mesmo dados, e pelo menos uma delas é uma operação de WRITE

Exemplo de conflito - Escala 3

T1	T2
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)

A instrução write(A) de T_1 entra em conflito com a instrução read(A) de T_2.

A instrução write(A) de T_2 não está em conflito com a instrução read(B) de T_1 pois trabalham com dados diferentes.

Conflito de transações

- Sejam I_i e I_j instruções consecutivas de uma escala de execução S. Se elas não entram em conflito entre si, então podemos trocar a ordem de I_i e I_j para produzir nova escala de execução S'.
- Espera-se que S' seja equivalente a S, já que todas as instruções aparecem na mesma ordem, com exceção de I_i e I_j cuja ordem não importa.

Escala 5: após a troca de instruções na escala 3

T1	T2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

T1	T2
read(A)	
write(A)	
	read(A)
read(B)	
	write(A)
write(B)	
	read(B)
	write(B)

Escala 6 resultado de trocas de instruções sucessivas na escala 5

T1	T2
read(A)	
write(A)	
	read(A)
read(B)	
	write(A)
write(B)	
	read(B)
	write(B)

T1	T2
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

Se uma escala S puder ser transformada em outra, S', por uma série de instruções não conflitantes, então S e S' são consideradas **equivalentes no conflito**.

Conflito de transações

Uma escala S é conflito serializável se ela é equivalente no conflito a uma escala de execução seqüencial.

Como exemplo a própria escala 3 que é equivalente a escala seqüencial 1.

Exemplo: conflito serializável

Escala 3	
T1	T2
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

Esacala 1	
T1	T2
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

Escala 7: contra exemplo de escala conflito serializável

T3	T4
read(Q)	
	Write(Q)
write(Q)	

A escala 7 ao lado não é conflito serializável, já que não é equivalente à escala seqüencial <T_3, T_4> ou à escala seqüencial <T_4, T_3>.

Escala 8: escalas não equivalentes no conflito

T1	T5
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)

Pode-se ter duas escalas que produzam o mesmo resultado mas que não sejam equivalentes no conflito.

Verifique que a escala 8 ao lado não é equivalente no conflito à escala seqüencial <T_1,T_5>, porém os valores finais são os mesmos para ambas as escalas. Porém ...

Escala 8: escalas não equivalentes no conflito

... para o sistema determinar se a escala 8 produz o mesmo resultado que a escala seqüencial <T_1,T_5>, ele tem de analisar toda a computação executada por T_1 e T_5, em vez de analisar apenas operações de *read* e *write*.

Isto é difícil e oneroso!

Visão serializada

Considere duas escalas S e S', com o mesmo conjunto de transações participando das mesmas. Estas escalas são equivalentes na visão se:

1. Para cada item de dados *Q*, se a transação T_i fizer uma leitura no valor inicial de *Q* na escala *S*, então, a transação T_i deve, também, na escala *S'*, ler o valor inicial de *Q*;

Visão serializada

2. Para cada item de dados *Q*, se a transação T_i executar um read(*Q*) na escala *S*, e aquele valor foi produzido por meio de uma transação T_j (se houver), então, a transação T_i deve, também, na escala *S'*, ler o valor de *Q que foi produzido por meio da transação* T_j;
3. Para cada item de dados *Q*, a transação (se houver) que executa a operação final write(*Q*) na escala S tem de executar a operação de write(*Q*) final na escala *S'*.

Escala 9: exemplo visão serializada

T3	T4	T6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

Escala 9

Esta escala é visão serializada pois é equivalente em visão à escala seqüencial <T3, T4, T6> já que uma instrução read(Q) lê o valor inicial de Q em ambas as escalas, e T6 executa a escrita final de Q em ambas as escalas.

Visão serializada

- A escala 1 não é equivalente na visão à escala 2 já que em 1 o valor da conta A lido por T_2 foi produzido por T_1, e isso não ocorre na escala 2;
- A escala 1 é equivalente em visão à escala 3 porque os valores das contas A e B lidos por T_2 foram produzidos por T_1 em ambas as escalas.

Dizemos que uma escala de execução S tem visão serializada se for equivalente em visão a uma escala de execução seqüencial

Recuperação

T8	T9
read(A)	
write(A)	
	read(A)
read(B)	

Escala 10

- Existem situações em que é impossível recuperar. Na escala 10 como T_9 leu um item de dados A escrito por T_8, se T_8 falhar antes da efetivação e depois de T_9 ser efetivada, é impossível recuperar T_9.
- A maioria dos SGBDs exige que as escalas sejam recuperáveis.

Recuperação

- Uma escala recuperável é aquela na qual, para cada par de transações T_i e T_j, tal que T_j leia dados escritos previamente por T_i, a operação de efetivação de T_i aparece antes da operação de efetivação de T_j.

Recuperação em cascata

- Mesmo em uma escala recuperável, para o sistema recuperar-se corretamente da falha de uma transação T_j pode ser necessário desfazer diversas transações. Tais situações ocorrem se as transações leram dados escritos por T_j. Este fenômeno é conhecido por retorno em cascata ou *cascading rollback*.

Isolamento

- Bloqueio (lock)
- esquema de controle de concorrência

Transação em SQL

- commit
- rollback

Transação em SQL

Nível de consistência especificado pela SQL-92

- serializável (padrão)
- read repetitivo (só permite a leitura de registros que sofreram efetivação, exigindo que nenhuma outra transação atualize um registro entre duas leituras feitas por uma transação)
- read com efetivação (só permite a leitura de registros que sofreram efetivação, mas não exige read repetitivo)
- read sem efetivação (permite a leitura de registros que não sofreram efetivação. É o menor nível de consistência permitido)

Testes de serialização

- Ao projetarmos esquemas de controle de concorrência deveremos mostrar que as escalas geradas por eles são serializáveis (equivalentes a escala seqüencial)

- Teste para serialização de conflito feito através de grafo de precedência, onde um write antes de um read ou um read antes de um write ou um write antes de um write geram uma aresta de i para j. Se houver ciclos a escala não é conflito serializável.

Testes de serialização de conflito

- Grafo de precedência para a escala 1

- Grafo de precedência para a escala 4

O tempo gasto por este teste é da ordem de n^2, onde n é o número de transações.

Testes de serialização de visão

O teste para serialização de visão é mais complicado. Prevê a construção de grafos de precedência rotulados, verificando a ausência de ciclos em todos os rotulados distintos possíveis. Este teste é a ordem de 2^n, onde n é o número de transações.

Controle de Concorrência

Controle de Concorrência

- Quando diversas transações são executadas de modo concorrente, a propriedade de isolamento pode não ser preservada
- É necessário que o sistema controle a interação entre elas e este controle é chamado Controle de Concorrência

Gerenciador de recuperação

- Responsável pelo suporte à atomicidade e durabilidade
- cópias ***shadow*** (exemplos em editores)
- ponteiros db_pointer

Controle de Concorrência

- Base na propriedade da serialização
- existem mecanismos para o controle de concorrência que admite a ordenação do processamento de forma não serializada

Serialização

Sejam as transações Ti e Tj, com instruções Ii e Ij em um mesmo item de dado:

- se Ii e Ij são instruções de read, a seqüência de execução não importa
- se Ii é uma instrução de read e Ij uma instrução de Write, a ordem como acontecem as instruções influencia o resultado de Ti e Tj
- o mesmo se Ii for uma instrução de write e Ij uma instrução de read
- se Ii e Ij são instruções de write, a ordem como acontecem não afeta as transações Ti e Tj, mas é necessário verificar o que se deseja para obtenção da ordem correta de execução, pois prevalece o ultimo write.

Execuções concorrentes

- Seja T1 uma transação que transfere $50 da Conta *A* para a Conta *B*
- Seja T2 uma transação que transfere 10% do saldo da conta A para a conta B

Conflito

Ti	Tj
read(a)	
write(a)	
	read(a)
	write(a)
read(b)	
write(b)	
	read(b)
	write(b)

Ti	Tj
read(a)	
write(a)	
read(b)	
write(b)	
	read(a)
	write(a)
	read(b)
	write(b)

Se uma escala S puder ser transformada em uma escala S', por uma série de instruções não conflitantes, dizemos que S e S' são equivalentes em conflito

Conflito Serializável

O conceito de equivalência no conflito leva ao conceito de serialização de conflito. Dizemos que uma escala de execução S é conflito serializável, se ela é equivalente no conflito a uma escala de execução sequencial.

Lock

Protocolos com base em Bloqueios (lock)

Uma forma de se garantir a serialização é obrigar que o acesso aos itens de dados seja feito de maneira mutuamente exclusiva, isto é, enquanto uma transação acessa um item de dados, nenhuma outra transação poderá modificá-lo.

Bloqueios

Formas de bloqueio:

Compartilhado (s): Se uma transação T obteve um bloqueio compartilhado sobre um item de dados, então T pode ler mas não escrever sobre o item de dados

Exclusivo (x): Se uma transação T obteve um bloqueio exclusivo sobre um item de dados, então T pode tanto ler como escrever sobre o item de dados

Bloqueios

- Toda transação solicita ao gerenciador de concorrência permissão para suas operações
- E só inicia essas operações após obter esta permissão.

Bloqueios

Quando uma transação *A* consegue um bloqueio sobre um item de dado *Q* imediatamente, a despeito da presença de um bloqueio do modo *B*, então dizemos que o modo *A* é compatível com o modo *B*. *Com(A,B)*

Bloqueios

Transações bloqueio compatível

	Compatilhado	Exclusivo
Compartilhado	V	F
Exclusivo	F	F

Bloqueios

- Um bloqueio exclusivo é solicitado por meio de um *lock*-x*(Q)* e o desbloqueio por *unlock(Q).*
- Para acessar um item de dados, uma transação solicita primeiramente o bloqueio.
- Se já estiver incompativelmente bloqueado o gerenciador de controle de concorrência não concederá o bloqueio até que todos os bloqueios incompatíveis sejam desfeitos

Bloqueios

- Nem sempre o desbloqueio termina após o acesso final pois isto pode comprometer a serialização
- Exemplo disto uma transação Ti que tira $50 de uma conta B e soma esta quantia a uma conta A, executada simultaneamente com uma transação Tj que soma os saldos de A e B.
- Suponha inicialmente A = 100 e B=200. Veja o instante dos desbloqueios de Ti e o da soma Tj

Bloqueios

Ti	Tj
lock-x (b)	lock-s (a)
read(b)	read(a)
b=b-50	unlock (a)
write b	lock-s (b)
unlock(b)	read (b)
lock-x (a)	unlock (b)
a = a + 50	display (a+b)
write a	
unlock (a)	

Conflito

Ti	Tj
Lock-x(b)	
Read(b)	
B = b- 50	
Write (b)	
unlock (b)	lock-s(a)
	read(a)
	unlock (a)
	lock-s(b)
	read(b)
	unlock(b)
	display (a+b)
Lock-x(a)	
a = a+50	
Write (a)	
Unlock (a)	

- Se as transações Ti e Tj não forem executadas serialmente, na forma TiTj ou TjTi, pode haver erro no final
- Verifiquem

Bloqueios

- Muitas vezes bloqueios podem causar situações indesejáveis

  Por exemplo, uma transação A ficar esperando que outra B libere o bloqueio sobre determinado dado Q, enquanto a transação B fica esperando que a transação A libere outro dado Q_1 para terminar.

- Esta situação é chamada de *deadlock* (impasse)

Bloqueios

- Deadlocks são preferíveis a estados inconsistentes, já que podem ser tratados por *rollback*
- Cada transação do sistema segue determinado conjunto de regras chamado protocolo de bloqueio.
- Um protocolo de bloqueio reduz o número de escalas de execuções possíveis, mas todas elas serializadas.

Concessão de bloqueios

- Situação de transação estagnada (*starved*) Como evitar?

Quando uma transação T solicita o bloqueio de um item de dado de modo exclusivo, o bloqueio será concedido se:

- não houver nenhuma transação com bloqueio sobre o item de dado de modo conflitante com a transação
- se não houver outra transação que esteja esperando um bloqueio sobre o item de dado e que tenha feito esta solicitação antes de T

Concessão de bloqueios

Protocolo de Bloqueio em duas fases

- fase de expansão: uma transação pode obter bloqueios, mas não pode liberar nenhum
- fase de encolhimento: uma transação pode liberar bloqueios mas não consegue obter nenhum bloqueio novo
- este protocolo garante a serialização de conflitos

Concessão de bloqueios

Protocolo de bloqueio em duas fases severo:

- exige em adição ao protocolo de bloqueio em duas fases tal que bloqueios exclusivos sejam mantidos até o final da transação. Isto evita os rollbacks em cascata.

Concessão de bloqueios

Protocolo de bloqueio em duas fases rigoroso:

- exige em adição ao protocolo de bloqueio em duas fases que todos os bloqueios (independentes do tipo) sejam mantidos até o final da transação.
- A maioria dos sgbds comerciais exige que o bloqueio seja rigoroso ou severo.

Concessão de bloqueios

Protocolo com base em Grafos (árvores)

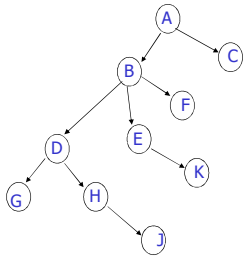
- gera-se uma árvore sobre determinada ordem de acesso aos dados
- o primeiro bloqueio de T pode ser em qualquer dado

Concessão de bloqueios

Protocolo com base em Grafos (árvores)

- subseqüentemente, outro dado pode ser bloqueado, mas somente se seu pai já estiver bloqueado por T
- um item de dado pode ser bloqueado a qualquer instante
- um item de dado que foi bloqueado e desbloqueado por T, não pode ser bloqueado novamente

Concessão de bloqueios



Concessão de bloqueios

- O protocolo de arvores não garante facilidades de recuperação e inexistência de cascata
- Bloqueios exclusivos até o final da transação diminuem a concorrência

Concessão de bloqueios

- Uma alternativa para melhorar a concorrência sem garantir a recuperação:
  - para cada transação com escrita de dados sem confirmação (*commit*) registra-se que ela realizou a ultima escrita de dados e que existe uma dependência de confirmação
  - essa transação não tem permissão para confirmar antes de outras que já tenham dependência de commit
  - Se alguma anterior for abortada, ela também será abortada

Protocolos com base em Time Stamp

- Toda transação possui seu TS(T), criado pelo gerenciador de banco de dados antes que T inicie sua execução. Se uma transação T_i recebeu seu TS(T_i) e uma nova transação T_j entra no sistema.
  - W-TS: denota o maior TS de qualquer transação que execute um WRITE no item de dado
  - R-TS denota o maior TS de qualquer transação que execute um READ no item de dado

Protocolos de ordenação por Time Stamp

- Assegura que quaisquer operações de leitura e escrita sejam executadas por ordem de TS.

  Suponha que T_i emite um *read* em um item de dado
  - se TS(T_i) < W-TS, então T_i precisa ler o valor do item de dado que foi sobreposto, a operação *read* é rejeitada e Ti é desfeita
  - se TS(T_i) >= W-TS, então a operação *read* é executada e R-TS recebe o maior valor do item de dado entre R-TS e TS(T)

Protocolos de ordenação por Time Stamp

Suponha que T_i emite um *write* em um item de dado

- se TS(T_i) < R-TS, então o valor do item de dado que T_i está produzindo foi necessário antes e o sistema assumiu que aquele valor não seria modificado. A operação *write* é rejeitada e T_i é desfeita
- se TS(T_i) < W-TS, então T_i está tentando escrever um valor obsoleto. A operação *write* é rejeitada
- de outro modo a operação *write* é executada e W-TS é registrado em TS(T)

Protocolos com base de Validação

- Fase de leitura
- Fase de validação
- Fase de escrita

- cada qual com seu TS

Granularidade Múltipla: *lock escalation*

- Quando T solicita diversos itens de dados, talvez seja melhor um *lock* em todo o banco de dados.
- Quando em apenas parte dos dados, o l*ock* pode ser feito em partes (tabelas, páginas ou registros)

Deadlock

Um sistema está em *deadlock* se há um conjunto de transações tal que toda a transação deste conjunto está esperando outra transação também nele contida.

Esquema de prevenção de Deadlock

- esquema esperar-morrer (*wait-die*) (uma transação só pode esperar se possuir um TS menor que a transação em curso, ou seja se for mais antiga)
- esquema ferir-esperar (*wound-die*) (é uma contrapartida a anterior. A transação só pode esperar se possuir um TS maior, ou seja, se for menor que a transação em curso. Caso contrário a transação em curso será desfeita)

Esquema de prevenção de Deadlock: Timeout

- Tempo esgotado para o bloqueio (timeout)
- Facilita transações curtas com freqüentes esperas longas

Fim da unidade

Exercícios