RESUMO

CAPÍTULO 16

“OVERVIEW OF TRANSACTION MANAGEMENT”

Prof.: Geovane Magalhães

Alunos:

Gabriela G. Martins

Edmar R. S. de Rezende

ÍNDICE ANALÍTICO

16 VISÃO GERAL SOBRE GERENCIAMENTO DE TRANSAÇÕES

Neste capítulo é discutido o conceito sobre uma transação, que é o ponto inicial para execução concorrente e recuperação de falhas de sistemas em um SGBD. Uma transação é definida como qualquer execução de um programa usuário em um SGBD e difere da execução de um programa ao lado de fora do SGBD em importantes aspectos.

Por razões de performance, o SGBD deve paralelizar as ações de várias transações. Entretanto, para dar aos usuários uma maneira simples de entender a execução de seus programas, a paralelização é feita cuidadosamente para garantir que o resultado da execução concorrente de transações é equivalente a uma execução serial do mesmo conjunto de transações. O modo como o SGBD mantém execuções concorrentes é um aspecto importante do gerenciamento de transações e ponto principal do controle de concorrência. Um ponto atual é como o SGBD mantém transações parciais. O SGBD garante que as alterações feitas por transações parciais não são vistas por outras transações. Esse é o ponto principal para recuperação de falha.

Na seção 16.1 são discutidas quatro propriedades fundamentais sobre transações e como o SGBD as garante. Na seção 16.2 é apresentada uma maneira abstrata para descrever a execução concorrente de várias transações, chamada de escalonamento. Na seção 16.3 vários problemas que podem surgir por causa da execução concorrente são discutidos. O protocolo de controle de concorrência lock-based é introduzido na seção 16.4. Falhas de performance associadas ao protocolo lock-based são discutidas na seção 16.5. Bloqueio e propriedades das transações no contexto SQL são considerados na seção 16.6. Finalmente, na seção 16.7 é apresentada uma visão geral de como o sistema de banco de dados recupera-se de falhas e quais passos são tomados durante uma execução normal para suportar recuperação de falhas.

16.1 AS PROPRIEDADES ACID

Uma transação é a execução de um programa, vista pelo SGBD como uma série de leituras e escritas.

Um SGBD deve garantir quatro propriedades de transação para manter os dados protegidos de acesso concorrente e de falhas de sistema:

- Atomicidade: A execução de toda transação deve ser considerada atômica; ou todas as ações são executadas ou nenhuma delas é.;
- Consistência: Cada transação deve preservar a consistência do banco de dados;
- Isolamento: Toda transação é isolada ou protegida das ações de outras transações concorrentes;

- Durabilidade: Uma vez informada a conclusão de uma transação de maneira satisfatória, o SGBD deve persistir os resultados da transação mesmo que o sistema sofra uma queda antes que esses resultados sejam persistidos no disco.

O acrônimo ACID é utilizado para referências das quatro propriedades citadas acima: atomicidade, consistência, isolamento e durabilidade.

16.1.1 CONSISTÊNCIA E ISOLAMENTO

Os usuários são responsáveis pela consistência de uma transação. O SGBD não pode ser responsabilizado por erros de lógica no programa do usuário, os quais podem resultar em transações que poderão deixar o banco de dados em um estado inconsistente.

O isolamento deve garantir que mesmo havendo ações de transações concorrentes correndo em paralelo com as ações de uma determinada transação, o último resultado deve ser o mesmo de quando as transações são executadas de maneira serial.

16.1.2 ATOMICIDADE E DURABILIDADE

A causa de erros durante a execução de transações pode ser de três tipos:

1. uma transação pode ser interrompida, ou finalizada insatisfatoriamente, pelo SGBD porque alguma anormalidade ocorreu durante sua execução. Quando essa situação ocorre, a transação é automaticamente reiniciada e re-executada.

2. o sistema pode sofrer uma queda enquanto uma ou mais transações estão em progresso;

3. uma transação pode encontrar uma situação inesperada e resolver interromper a si própria.

O fato de uma transação poder ser interrompida ao meio pode deixar o banco de dados em um estado inconsistente. Por causa da propriedade de atomicidade, o SGBD deve garantir a retirada dos efeitos de transações parciais do banco de dados. Para realizar isso, o SGBD mantém um registro, chamado *log*, de todos os dados que foram escritos no bando de dados. O log é utilizado para garantir a propriedade de durabilidade – se o sistema sofrer uma queda antes que as alterações geradas por uma transação que foi executada satisfatoriamente forem persistidas em disco, o *log* é utilizado para restaurar essas informações quando o sistema for normalizado.

16.2 TRANSAÇÕES E AGENDAMENTOS

Uma transação é vista pelo SGBD como um conjunto de ações. Dentre as ações que podem ser executadas por uma transação temos a leitura e a escrita. Para mantermos uma notação simples, assumiremos que o objeto O é sempre lido de uma variável de programa de mesmo nome. Denotaremos, então, a ação de uma transação T lendo o objeto O como $R_T(O)$. Similarmente,

denotaremos a escrita como $W_T(O)$. Na situação da transação T ser excluída do contexto, omitiremos o subscrito.

Além disso, toda transação deve especificar *commit* ou *abort* como ação final. Tais ações serão representadas por C_T ou A_T, respectivamente.

Nesse ponto, faremos as seguintes considerações:

- Transações interagem entre si apenas via banco de dados; não é permitida a troca de mensagens;
- Um banco de dados é uma coleção fixa de objetos independentes.

Um escalonamento é uma lista de ações vindas de um conjunto de transações. A ordem dos elementos da lista de ações deve seguir a ordem desses elementos em suas transações. Podemos considerar que um escalonamento é uma seqüência de execução.

16.3 EXECUÇÃO CONCORRENTE DE TRANSAÇÕES

O SGBD paraleliza as ações de transações diferentes para melhoramento da performance, mas nem todo paralelismo deve ser permitido.

16.3.1 MOTIVAÇÃO DE EXECUÇÃO CONCORRENTE

Garantir o isolamento da transação enquanto cada execução concorrente é permitida é difícil mas necessário por razões de performance. Primeiramente, enquanto uma transação espera para ler uma página do disco, a CPU pode processar outra transação. Isso é possível porque a atividade de I/O pode ser feita em paralelo com a atividade da CPU. Além disso, paralelizar a execução de uma transação pequena com a execução de uma transação mais complexa normalmente permite que a primeira seja finalizada rapidamente. Numa execução serial, uma transação pequena poderia ser temporariamente bloqueada por uma transação complexa, levando a esperas não previstas no tempo de resposta ou aumentando o tempo gasto para completar uma transação.

16.3.2 SERIALIZABILIDADE

Um escalonamento serializável aplicado sobre um conjunto S de transações finalizadas satisfatoriamente é um escalonamento cujo efeito sobre qualquer instância de banco de dados consistente será idêntico a um escalonamento serial aplicado a S. Isto é, a instância do banco de dados resultante do escalonamento citado é idêntica à instância do banco de dados resultante da execução de transações em alguma ordem serial.

A execução de transações de forma serial em ordens diferentes pode produzir resultados diferentes, porém, assume-se que todos serão aceitáveis; o SGBD não dá garantias de qual delas será conseqüência de uma transação paralela.

Um SGBD deve executar as transações, em algumas vezes, por caminhos não equivalentes a qualquer execução serial; ou seja, utilizando um escalonamento que não é serializável. Isso pode acontecer por duas razões:

1. o SGBD deve utilizar um método para controle de concorrência que garanta o escalonamento utilizado;
2. SQL dá aos programadores de aplicação a opção de instruir o SGBD a escolher escalonamentos não-serializáveis.

16.3.3 IREGULARIDADES DEVIDO À EXECUÇÃO PARALELA

Existem três principais caminhos em que um escalonamento de transações finalizadas satisfatoriamente poderiam ser aplicadas a uma base de dados consistente e deixá-la em um estado inconsistente. Duas ações aplicadas a um mesmo objeto são conflitantes se, pelo menos uma delas, é a ação de escrita. As três situações irregulares podem ser descritas em termos de quando as ações de duas transações T1 e T2 são conflitantes: em uma escrita-leitura, T2 lê um objeto antes do objeto ser escrito por T1; os conflitos de leitura-escrita e escrita-escrita são definidos de maneira semelhante.

LENDO DADOS NÃO PERSISTIDOS (CONFLITOS DE ESCRITA-LEITURA)

A primeira fonte de irregularidades é que uma transação T2 poderia ler um objeto de banco de dados A que vêm sendo modificado por uma outra transação T1 e que ainda não foi persistido. Tal leitura é chamada leitura “suja”.

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
	Commit
R(B)	
W(B)	
Commit	

Tabela 1 - Leitura de Dados não Persistidos

LEITURAS NÃO-REPETÍVEIS (CONFLITOS DE LEITURA-ESCRITA)

O segundo caminho em que comportamentos irregulares poderiam resultar é que uma transação T2 poderia alterar o valor de um objeto A que está sendo lido por uma transação T1, enquanto T1 ainda está em progresso.
Se T1 tentar ler o valor de A novamente, terá um resultado diferente. Isso é chamado de leitura não-repetível.

EXCESSO DE ESCRITAS EM DADOS NÃO PERSISTIDOS (CONFLITOS DE ESCRITA-ESCRITA)

A terceira fonte de comportamento irregular é que a transação T2 poderia reescrever o valor de um objeto A, o qual já havia sido alterado por uma transação T1, a qual ainda está em progresso. Esse tipo de leitura é chamado de leitura cega.

16.3.4 AGENDAS QUE ENVOLVEM TRANSAÇÕES FALHAS

Intuitivamente, todas as ações de transações falhas devem ser desfeitas. A definição de escalonamentos serializáveis será extendida como se segue: um escalonamento serializável aplicada a um conjunto S de transações é uma agenda cujo efeito em qualquer instância consistente de banco de dados é idêntico àquele de uma agenda serial aplicada a um conjunto de transações finalizadas em S.
Essa definição também é válida para ações de transações falhas, no caso de serem desfeitas completamente, o que pode ser impossível em certas situações.
Supondo que uma transação T2 reescreva o valor do objeto A que está sendo modificado por uma transação T1, a qual está ainda em progresso, e T1 falha. Todas as mudanças de T1 são desfeitas, ou seja, todo objeto modificado por T1 será restaurado com o valor que possuía antes de sofrer a ação de T1. Sendo assim, as alterações de T2 são perdidas, mesmo que T2 decida realizar o *commit*.

16.4 CONTROLE DE CONCORRÊNCIA LOCK-BASED

Um SGBD deve garantir que apenas escalonamentos serializáveis e recuperáveis sejam permitidos e que nenhuma ação de transações finalizadas sejam perdidas enquanto transações que falharam são desfeitas. O SGBD utiliza um protocolo de travamento para realizar isso. Um *lock* é um pequeno objeto de histórico associado a um objeto de banco de dados. Um protocolo de travamento é um conjunto de regras a serem seguidas por cada transação para garantir que, mesmo que as ações de várias transações forem paralelizadas, o resultado final é idêntico ao resultado de executar todas as transações de maneira serial. Cada protocolo de travamento utiliza *locks* diferentes, como *locks* compartilhados ou *locks* exclusivos.

16.4.1 STRICT TWO-PHASE LOCKING

O protocolo de travamento mais utilizado, também denominado de STRICT 2PL, possui duas regras:

1. Se uma transação T deseja ler um objeto, ela primeiramente efetua a requisição de um *lock* exclusivo para o objeto.

Uma transação que possui um *lock* exclusivo pode também ler um objeto. Ao realizar a requisição a transação é suspensa até que o SGBD esteja apto a conceder o *lock* requerido. Além disso, uma pilha do *lock é* mantida e existe a garantia de que se uma transação consegue um *lock* exclusivo ou compartilhado para um determinado objeto, nenhuma outra transação irá conseguir um *lock* exclusivo ou compartilhado para o mesmo objeto.

2. Todos os *locks* conseguidos por uma transação são liberados quando a transação é completada.

Os pedidos para adquirir ou libertar *locks* podem ser automaticamente inseridos nas transações pelo SGBD.

O protocolo de travamento fornece apenas segurança para transações paralelizadas. Se duas transações acessam partes do banco de dados completamente independentes, elas obtém os *locks* de forma concorrente e prosseguem. Por outro lado, se duas transações acessam o mesmo objeto e uma delas deseja modificá-lo, suas ações são ordenadas serialmente – todas as ações de uma dessas transações são completadas antes que outra transação possa prosseguir.

16.4.2 DEADLOCKS

Assumindo T1 e T2 como transações, considere o exemplo a seguir

- transação T1 adquire um *lock* exclusivo para o objeto A;
- T2 adquire um *lock* exclusivo para o objeto B;
- T1 requisita um *lock* exclusivo para o objeto B e o pedido é armazenado numa fila;
- T2 requisita um *lock* exclusivo para o objeto A e o pedido é armazenado numa fila.

Desta forma, T1 está aguardando que T2 liberte o *lock* do objeto B e T2 está aguardando que T1 liberte o *lock* do objeto A. Tal ciclo de transações aguardando a liberação de *locks* é chamado de *deadlock*. Fica claro que essas transações não terão progresso. Ou pior, podem estar solicitando *locks* que estão sendo requeridos por outras transações. O SGBD deve prever ou detectar (e resolver) qualquer situação de *deadlock*.

Uma maneira simples de detectar *deadlocks* é utilizar um mecanismo de *timeout*. Se uma transação tem aguardado muito tempo por um pedido de *lock*, pode-se assumir que um ciclo de *deadlock* foi detectado e escolhe-se por abortá-lo.

16.5 PERFORMACE DO MECANISMO DE LOCK

Esquemas baseados em *lock* são projetados para resolver conflitos entre transações e usam dois mecanismos básicos: bloqueio e aborto. Ambos os mecanismos acarretam em uma penalidade na performance: transações bloqueadas podem obter *locks* que forcem outras transações a esperar, causando o aborto e o reinício da transação, resultando obviamente na perda de todo o trabalho realizado pela transação.

Um *deadlock* representa uma instância extrema de um bloqueio no qual um conjunto de transações fica bloqueado eternamente, a não ser que uma das transações que faz parte do *deadlock* seja abortada pelo SGBD.

Na prática, pouco menos de 1% das transações ficam envolvidas em um deadlock, resultando em relativamente poucos abortos. Contudo, o overhead dos locks é devido principalmente aos atrasos causados pelos bloqueios.

Analisando o impacto do mecanismo de bloqueio sobre as transações, podemos perceber que em um primeiro momento existem poucas transações com poucos conflitos. À medida que o número de transações ativas cresce o throughput também cresce na mesma proporção. Como mais e mais transações são executadas concorrentemente sobre o mesmo número de objetos, os impactos do mecanismo de lock começam a aparecer. O throughput começa a crescer de forma mais lenta, até o ponto em que começa a ser reduzido, devido ao crescimento do número de bloqueios causado pelo aumento do número de transações ativas.

Throughput pode ser aumentado de três formas:

- Realizando o lock nas menores partes possíveis dos objetos (minimizando a possibilidade de duas transações necessitarem do mesmo lock).
- Reduzindo o tempo que uma transação permanece com o lock (assim outras transações serão bloqueadas por períodos de tempo mais curtos).
- Reduzindo os *hot spots*. Um *hot spot* é um objeto da base de dados que é freqüentemente acessado e modificado, causando um grande atraso em relação aos bloqueios. *Hot spots* podem afetar significativamente a performance.

16.6 SUPORTE A TRANSAÇÕES EM SQL

Até o momento, as transações e o gerenciamento de transações têm sido estudados usando um modelo abstrato de uma transação como uma seqüência de leituras, escritas, e ações de *abort* / *commit.* Nesta seção será considerado o suporte que o SQL provê para os usuários especificarem o comportamento no nível de transação.

16.6.1 CRIANDO E TERMINANDO TRANSAÇÕES

Uma transação é automaticamente iniciada quando um usuário executa uma instrução que acessa a base de dados ou os catálogos, tais como uma consulta SELECT, um comando UPDATE, ou uma instrução CREATE TABLE.

Uma vez que uma transação foi iniciada, outras instruções podem ser executadas como parte desta transação até a transação terminar com um comando de COMMIT ou um comando de ROLLBACK (a palavra chave no SQL que faz o papel do *abort*).

O SQL:1999 inclui duas novas funcionalidades para suportar aplicações que envolvem transações de longa duração. A primeira é chamada de *savepoint*, que permite identificar um ponto seguro em uma transação para onde realizar operações de *rollback* em caso de falha. Isto é feito utilizando os comandos:

SAVEPOINT <nome do *savepoint*>

ROLLBACK TO SAVEPOINT <nome do *savepoint*>

A segunda é o mecanismo de *transações encadeadas*, que permite realizar o *commit* ou o *rollback* em uma transação e imediatamente iniciar outra transação. Isto é feito utilizando a expressão AND CHAIN em um COMMIT ou um ROLLBACK.

16.6.2 O QUE DEVEMOS BLOQUEAR?

Uma questão importante a considerar no contexto de SQL é o que o SGBD deve tratar como um objeto durante a atribuição de locks para uma determinada expressão SQL?

O SGBD pode conceder locks a objetos de diferentes granularidades: podemos ter o lock para uma tabela inteira ou apenas para uma linha da tabela. Esta última abordagem é adotada nos sistemas atuais pelo fato de oferecer uma melhor performance. Na prática, apesar do lock no nível de linhas da tabela ser geralmente melhor, a escolha da granularidade do lock é uma decisão complicada. Uma transação que examina várias linhas e modifica aquelas que satisfazem uma determinada condição pode tirar proveito de um esquema onde locks compartilhados são atribuídos à toda a tabela e locks exclusivos são atribuídos às linhas da tabela.

Além disso, dependendo de como os locks são atribuídos, pode ocorrer o problema conhecido como “o problema do fantasma”. Esse problema pode fazer com que uma transação acesse um conjunto de tuplas duas vezes seguidas e encontre valores diferentes em cada uma delas, mesmo que não tenha modificado tais tuplas. Para evitar o problema do fantasma, o SGBD deve conceitualmente realizar o lock em todas as possíveis linhas contendo o valor procurado. Uma forma de fazer isso é realizando o lock de toda a tabela, diminuindo assim a concorrência. Também é possível tirar proveito dos índices para alcançar um melhor resultado, mas em geral a prevenção do problema do fantasma impacta significativamente na concorrência.

16.6.3 CARACTERÍSTICAS DAS TRANSAÇÕES EM SQL

Para permitir que o programador tenha controle sobre o overhead causado pelo mecanismo de lock sobre suas transações, o SQL permite que sejam especificadas três características de uma transação: modo de acesso, tamanho do diagnóstico e nível de isolamento.

O tamanho do diagnóstico determina o número de condições de erro que podem ser gravadas.

Já o modo de acesso pode ser READ ONLY ou READ WRITE. No modo de acesso READ ONLY não é permitido à transação modificar a base de dados. Assim, os comandos INSERT, DELETE, UPDATE e CREATE não podem ser executados. Caso esses comandos precisem ser executados, o modo de acesso READ WRITE deve ser utilizado. Com o modo de acesso READ ONLY, apenas locks compartilhados precisam ser obtidos, aumentando assim a concorrência.

O nível de isolamento controla o quanto uma determinada transação está exposta à ação de outras transações executando concorrentemente. Optando por um dos quatro níveis de isolamento possíveis, o usuário poderá obter um alto grau de concorrência, aumentando contudo o grau de exposição a modificações ainda não efetivadas por transações.

As opções de níveis de isolamento são: READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ e SERAILIZABLE. O mais alto grau de isolamento é proporcionado pelo nível SERIALIZABLE, onde a transação obtém os locks antes da leitura ou escrita de objetos e os libera apenas ao fim de sua execução, de acordo com o protocolo Strict 2PL. Além disso, este nível de isolamento é geralmente o mais seguro e recomendado para a maioria das transações.

Quando uma transação é iniciada, por padrão ela é definida como SERIALIZABLE e READ WRITE.

16.7 INTRODUÇÃO À RECUPERAÇÃO DE FALHAS

O gerenciador de falhas do SGBD é responsável por garantir a atomicidade e a durabilidade das transações. Ele garante a atomicidade desfazendo as ações de transações que não realizaram o commit, e a durabilidade garantindo que todas as ações de transações que realizaram o commit sobreviverão a falhas do sistema.

Quando um SGBD reinicia após uma falha, o gerenciador de recuperação assume o controle e deve retornar a base de dados a um estado consistente. Comumente assumimos que as escritas em disco são operações atômicas, ou seja, não é possível ocorrer falhas durante a operação de escrita. No entanto, tal hipótese não é nada realista. Na prática, as escritas em discos não possuem essa propriedade, e a alguns passos devem ser realizados durante a reinicialização após uma falha, para verificar quais escritas mais recentes em uma determinada página foram bem sucedidas e para poder lidar com aquelas que não foram.

16.7.1 ROUBANDO FRAMES E FORÇANDO PÁGINAS

Com relação à escrita de objetos, duas questões adicionais podem surgir:

- As modificações feitas em um objeto O no buffer pool por uma transação T pode ser escrita no disco antes de T realizar o commit? Tais escritas são executadas quando outra transação precisa carregar uma página e o gerenciador de buffer opta por substituir o frame contendo O. Obviamente, esta página deve ser salva por T. Se tal escrita é permitida, dizemos que uma abordagem de “roubar” está sendo usada.
- Quando uma transação realiza o commit, deve ser garantido que todas as transações feitas sobre objetos no buffer pool sejam imediatamente escritas em disco? Se a resposta é sim, então dizemos que uma abordagem de forçar está sendo utilizada.

Estas políticas possuem conseqüências importantes. Se uma abordagem de “não roubar” assume que todas as páginas modificadas pelas transações em andamento podem ser acomodadas no buffer pool, na presença de grandes transações essa hipótese não é muito realista. Já a abordagem de “forçar” resulta em um número excessivo de operações de entrada/saída. Por estas razões, muitos sistemas utilizam uma abordagem baseada em “roubar” e “não forçar”.

16.7.2 PASSOS RELACIONADOS À RECUPERAÇÃO DURANTE UMA EXECUÇÃO NORMAL

O gerenciador de recuperação de um SGBD mantém algumas informações durante uma execução normal das transações para permitir que ele consiga realizar sua tarefa durante uma falha. Em particular, um log de todas as modificações realizadas na base de dados é salvo em um meio de armazenamento estável.

É importante garantir que todas as entradas no log descrevendo as modificações na base de dados sejam escritas em um meio de armazenamento estável *antes* das respectivas modificações serem feitas. Caso contrário, o sistema pode falhar imediatamente após uma modificação na base de dados sem existir um registro correspondente no log. Este método de escrita no log é conhecido como *Write-Ahead Log*, ou simplesmente WAL.

O log permite ao gerenciador de recuperação desfazer as ações de transações abortadas ou não completadas e refazer as ações de transações que realizaram o commit.

16.7.3 VISÃO GERAL DO ARIES

O ARIES é um algoritmo de recuperação que é projetado para trabalhar com uma abordagem de “roubar” e “não forçar”. Quando o gerenciador de recuperação é invocado após uma falha, o reinício se procede em três fases:

- Fase de Análise: identifica páginas sujas no buffer pool e transações ativas no momento da falha.
- Fase de Refazer: repete todas as ações, começando do ponto apropriado no log e restaura o estado da base de dados idêntico ao momento da falha.

- Fase de Desfazer: desfaz as ações das transações que não realizaram o commit, de forma que a base de dados reflita apenas as ações das transações que realizaram o commit.

16.7.4 ATOMICIDADE: IMPLEMENTANDO ROLLBACK

É importante destacar que o subsistema de recuperação é também responsável pela execução do comando ROLLBACK, que aborta uma transação. De fato, a lógica envolvida em desfazer uma transação é idêntica à lógica usada durante a fase de desfazer do mecanismo de recuperação de uma falha do sistema. Todos os registros de uma determinada transação são organizados em uma lista ligada e podem ser eficientemente acessados na ordem reversa para facilitar o rollback da transação.