Introdução a Banco de Dados - Trabalho 2 - Conceitos de Transação e Concorrência

Tiago Gonçalves da Silva[1]

[1]Departamento Acadêmico de Informática - UTFPR
Curitiba - PR - Brasil

tiagosilva.2019@alunos.utfpr.edu.br

1. Controle de Transações

1.1. O que são transações

Uma transação trata-se de uma sequência de *queries* que acessam e atualizam dados. Uma transação é delimitada por chamadas de função do tipo ”iniciar transação” e ”finalizar transação”, desta forma, uma transação trata-se de todas as operações executadas entre estas chamadas.

O conceito mais importante da transação é sua indivisibilidade, todos os seus passos deve parecer para o usuário como uma unidade indivisível. Assim, caso ocorra qualquer problema durante sua execução, todas as mudanças que ocorreram ao banco de dados devem ser desfeitas.

Um exemplo mais simples de transação em banco de dados é uma transferência bancária entre duas contas. Neste caso o dinheiro da conta da pessoa ”A” está sendo transferido para a pessoa ”B”, o que não pode ocorrer, por exemplo, é que o dinheiro seja retirado da conta da pessoa ”A”, o sistema dê problema no meio da transação, e a pessoa ”B” fique sem receber o dinheiro, para isso o banco de dados precisa garantir que um estado pré transação seja restaurado.

1.2. Conceitos de ACID

ACID é um acrônimo criado para descrever quatro propriedades importantes para o conceito de transações, sendo que a inicial de cada uma destas propriedades corresponde a uma letra.

- **Atomicidade**: é a propriedade descrita anteriormente, que diz que cada transação é indivisível, assim, ao fim de uma transação todas as suas operações individuais foram completadas ou nenhuma foi. Caso ocorra qualquer problema durante, seja com a transação, com o banco de dados ou com o computador, um estado pré-transação deve ser restaurado;
- **Consistência**: A execução de uma transação é feita isoladamente, para preservar a consistência do banco de dados. Se uma transação iniciar atomicamente partindo de um banco de dados consistente, ela deve garantir que os dados se mantenham consistentes após sua execução;
- **Isolamento**: O sistema do banco de dados deve garantir que todas as operações das transações operem isoladas e sem interferência de outros comandos que estão ocorrendo ao mesmo tempo no banco de dados;
- **Durabilidade**: As mudanças feitas por uma transação devem persistir mesmo após o término de sua execução, então, caso o sistema falhe, ele não pode apagar as modificações feitas pela transação.

1.3. Modelo de transações e como implementa ACID

Toda transação está em um dos seguintes estados:

- **Ativa**: trata-se do estado inicial, a transação se mantém assim enquanto está executando;
- ***Failed***: trata-se do estado após ser descoberto que transação não pode mais ser executada normalmente;
- **Abortada**: é o estado após a transação e o banco de dados terem sido restaurados para um estado anterior ao início da transação;
- **Parcialmente *commited***: estado após o último comando da transação foi executado;
- ***Commited***: estado após a transação ser executada com sucesso e suas atualizações feitas no banco de dados.

1.4. Níveis de isolamento de transações

Controle de concorrência é importante para que haja o devido isolamento entre transações, entretanto isolamento demais pode fazer com que a execução de transações longas seja adiada, fazendo com que a execução da aplicação seja longa demais, assim, alguns protocolos podem precisar de menos níveis de isolamento. Os níveis de isolamento específicos do padrão SQL são:

- **Serializável**: pode garantir uma execução serializável, entretanto alguns sistemas de banco de dados podem, em alguns casos, permitir execuções não serializáveis;
- **Leitura repetível**: permite que apenas dados *commited* sejam lidos, e precisa que entre duas leituras de um dado durante um transação, não seja possível que outra transação autalize este dado;
- **Leitura *commited***: permite que apenas dados *commited* sejam lidos, entretanto, não necessita lerituras repetíveis;
- **Leitura não *commited***: permite que até dados não *commited* sejam lidos, é o menor nível de isolamento permitido pelo SQL.

1.5. Implementações destes controles

Existem diversas políticas de controle de concorrência para garantir que, mesmo com múltiplas transações ocorrendo de forma concorrente, o banco de dados se mantenha em um estado consistente. Os principais mecanismos de controle de concorrência são:

- ***Locking***: toda vez que uma transação for acessar e atualizar um conjunto de dados, ela recebe um *lock*, fazendo com que outras transações que desejam acessar estes dados tenham que esperar o *lock* ser liberado. Neste caso a transação que se apodera do *lock* deve segurá-lo somente o tempo necessário para sua execução, de forma a não prejudicar o desempenho das outras transações;
- ***Timestamps***: para cada dado, o sistema mantém dois *timestamps*, um para a transação que mais recentemente leu o item, e outro para a transação que escreveu o valor atual do dado. Servem para garantir que cada transação acesse os dados na ordem dos *timestamps* caso haja conflito;
- **Múltiplas versões**: neste caso cada transação ao iniciar recebe uma versão do banco de dados, esta versão é de acesso exclusiva dela, assim, cada transação vai fazer leitura e escrita de sua respectiva versão do banco de dados. As atualizações no banco de dados real só são feitas após feito o *commit* da transação.

2. Controle de Concorrência

2.1. Soluções para controlar a concorrência em bancos de dados

Como dito anteriormente, quando muitas transações executam de maneira concorrente, é necessário violar as propriedades de isolamento para garantir um bom desempenho do sistema, entretanto, é necessário que ainda se mantenha a consistência entre as transações.Para isso, são implementados mecanismos para o controle de concorrência, a fim de garantir a integridade do banco de dados enquanto possui desempenho aprimorado pelo uso de transações concorrentes.

2.2. Protocolos de *lock*

O método mais simples de implementar acesso exclusivo a dados é com protocolos de *lock*, isto é, somente a transação que estiver com o *lock* de um dado pode acessá-lo. Neste caso o processo de acesso a dados é definido da seguinte forma: uma transação faz o requerimento de um lock sobre o dados que deseja acessar, então, o gerenciador de controle de concorrência verifica se consegue dar acesso imediato a esse dado, caso não consiga a transação deve esperar até que o *lock* seja liberado por quem estiver fazendo uso dele.

Locks podem ter dois tipos: *lock* compartilhado, que permite que a transação leia os conteúdos do dado em questão; e *lock* exclusivo, que permite que a transação leia e escreva sobre o item. Múltiplas transações podem acessar um *lock* compartilhado, porém apenas uma pode possuir um *lock* exclusivo, este é o conceito de compatibilidade de *locks*.

2.3. Tratamento de *deadlocks*

Dependendo da maneira com que os *locks* são fornecidos às transações, o sistema pode entrar em um estado de *deadlock*, que é quando um conjunto de transações está em espera circular, onde uma espera que seguinte libere um *lock*, nesta situação nenhuma das transações consegue progredir em sua execução.

Existem duas formas principais de tratar o problema de *deadlock*, uma delas é a utilização de mecanismos de prevenção de *deadlock*, arquitetando o sistema de forma que ele nunca entre em um estado de *deadlock*. A outra forma são os métodos de detecção e recuperação de *deadlocks*, que resultam em ações drásticas para resolver um *deadlock* que esteja ocorrendo.

A prevenção de *deadlock* pode ser feita de duas formas. A primeira trata de garantir que esperas circulares não ocorram, ou fazer com que todos os *locks* precisem ser recebidos ao mesmo tempo pela transação. Na segunda é feito um *rollback* da transação sempre que a espera de um *lock* possa causar um *deadlock*.

Para detecção e recuperação de *deadlocks* o sistema precisa executar um procolo periódico de três passos: manter um registro sobre a alocação atual de *locks* para as transações e quais delas estão esperando por *locks*; possuir um algoritmo que permita usas estas informações para determinar se um sistema está em *deadlock*; possuir um algoritmo que, em caso de *deadlock*, faça os *rollbacks* necessários para sair deste estado.

É importante ressaltar que as duas abordagens resultam em *rollbacks* de transações, a prevenção é indicada quando a chance do sistema entrar em *deadlock* é considerada alta, do contrário, a detecção e recuperação são melhores.

2.4. Protocolos de transações

Uma forma de determinar a ordem de seriabilidade é selecionar a ordem das transações previamente, o método mais comum para isso é o *timestamp-ordering scheme*, neste protocolo cada transação possui seu devido *timestamp*, associado a ela antes que comece a sua execução. Quando outra transação entrar no sistema, esta obrigatoriamente terá um *timestamp* maior que as anteriores. Assim o sistema deve garantir que as transações executem suas operações de leitura e escrita conflitantes na ordem dos *timestamps*.

Em casos que a maioria das transações executa somente operações de escrita, a quantidade de conflitos por concorrência fica bem pequena, então, muitas destas transações se executadas sem controle de transação deixariam o sistema sem problemas de consistência. Isto aliado ao fato que o sistema de controle de transações pode atrasar o sistema, faz com que possa ser necessário um protocolo de transação mais flexível, neste caso é o protocolo de validação.

Este protocolo requer que cada transação passe por três fases sequenciais:

- **Fase de leitura**: permite que a transação leia os valores e guarde em suas variáveis locais, sem atualizar o banco de dados real;
- **Fase de validação**: a transação, ao precisar escrever no banco de dados, passa em um teste para validar se ela pode proceder para a fase seguinte sem violar a seriabilidade, caso não passe, o sistema aborta a transação;
- **Fase de escrita**: se a transação passou pela validação, suas variáveis locais que estão com valores para a escrita são copiadas para o banco de dados.

3. Referências

Silberschatz, A., Korth, H. F., and Sudarshan, S. (2020). *Database System Concepts, Seventh Edition*. McGraw-Hill Book Company.