Gerencia de Transações

A questão do gerenciamento de transações é um dos grandes avanços da tecnologia de banco de dados e ainda assim é um assunto que merece atenção especial. No passado os sistemas de informação atualizavam as informações em arquivos sequenciais indexados ou arquivos de acesso direto que eram mecanismos de armazenamento e consulta de informações bem limitados e sem os recursos de gerenciamento existentes nos sistemas de bancos de dados.

Nessa época o gerenciamento de transações era feito pelo sistema de informação pois o sistema de arquivos não tinha recursos para o gerenciamento de transações.

Com o surgimento dos primeiros sistemas multiusuário, onde haviam diferentes usuários acessando e atualizando simultaneamente as informações, surgiu o aspecto da concorrência pelos dados e era necessário administrar todos estes acessos, consultas e atualizações que ocorriam simultaneamente. Os projetistas perceberam rapidamente que todos estes acessos concorrentes poderiam gerar diversos problemas nos dados. Vou citar aqui dois problemas típicos.

Primeiro, os projetistas descobriram que era necessário garantir o isolamento de cada transação para que uma transação não interferisse na outra pois haviam situações em que os dados eram atualizados incorretamente, versões incorretas dos dados eram consultadas e o resultado era inconsistência nos dados, gerando informações erradas.

Além disso era preciso garantir que cada transação fosse atômica de forma a ser concluída integralmente ou não ser efetivada. Atualizações parciais não deviam ser permitidas. Se uma transação fosse composta por várias atualizações então elas deveriam ocorrer todas juntas ou então nenhuma delas poderia ocorrer, sob pena de gerar inconsistência nos dados.

O tratamento para esta situação era fazer o gerenciamento de transações pelo aplicação, ou seja, o próprio sistema de informação era responsável por garantir o gerenciamento de cada transação realizada. Isso era uma dor de cabeça e embora os analistas e projetistas se esforçassem para implementar controles para garantir a integridade das transações sempre haviam questões fora do controle da aplicações que eram relacionadas ao ambiente (rede) ou ao hardware. Ou seja, o gerenciamento de transações implementado pela aplicação era deficiente e os problemas eram apenas minimizados e não resolvidos.

Conceito de Transação, Item de Banco de Dados e Operações de Leitura e Gravação

Uma transação é uma unidade lógica de operações em um banco de dados. Uma transação pode ser composta de uma ou mais operações de banco de dados, que podem ser de leitura ou de atualização, estas podem ser operações de inserção (INSERT) , atualização (UPDATE) ou exclusão (DELETE). As operações de banco de dados que formam uma transação podem ser embutidas em um programa de aplicação ou podem ser executadas de forma interativa com o uso de SQL em um utilitário de banco de dados do tipo Oracle SQL Plus. Ou seja, transações podem ser enviadas a partir de programas de aplicação ou por utilitários de banco de dados. Em ambos os casos, devem ser processadas apropriadamente pelo SGBD.

De que forma o software de banco de dados reconhece uma transação a fim de poder tratá-la adequadamente? De que forma uma transação é especificada? Uma forma de especificar uma transação é determinar o início e fim da transação, os limites da transação com o uso das instruções explícitas begin transaction e end transaction. Neste caso, todas as operações de banco de dados compreendidas entre estas instruções begin transaction/end transaction são consideradas como sendo parte da transação.

Toda transação depende de quatro propriedades que são conhecidas pela sigla ACID:

Atomicidade - Toda transação deve ser atômica, ou seja, não pode ser dividida, ou fragmentada em partes. A idéia é que todas as operações de banco de dados que a compõem devem ser executadas como se fosse uma única operação. Se alguma operação de banco de dados falhar toda a transação deve ser desfeita. Quando todas as operações de banco de dados que compõem a transação forem executadas com sucesso, a transação pode ser efetivada.

Consistência - Toda transação, após ser executada, deve deixar o banco de dados em um estado consistente.

Isto significa que a transação deve satisfazer todas as regras e restrições definidas no banco de dados, o que inclui regras de integridade referencial, regras de domínio com valores permitidos para colunas, definição de chave primária, índices únicos e colunas de preenchimento obrigatório.

Isolamento - Toda transação deve ocorrer de forma isolada em relação às demais transações que estão ocorrendo no banco de dados. Os resultados parciais de cada transação não devem estar disponíveis para as demais transações. A idéia é que nenhuma transação possa interferir no funcionamento de outra transação sendo executada no mesmo banco de dados.

Durabilidade - Toda transação tem seus resultado permanente no banco de dados, somente podendo ser desfeito por uma transação subsequente.

Os tipos de falhas que podem causar os problemas gerados pela falta de gerenciamento de transações

Uma falha do computador (falha de sistema)

Um erro de transação ou do sistema

Uma condição de exceção detectada pelo sistema

Imposição de controle de concorrência

Falha de disco (falha de hardware)

Problemas físicos e catástrofes

Alguns problemas que podem ser gerados pela ocorrência de transações simultâneas em um ambiente multiusuário:

O problema da atualização perdida

O problema da atualização temporária/leitura suja (a consulta retorna dados gerados por outra transação mas que ainda nao foram confirmados pela outra transação)

O problema do resumo incorreto (a consulta retorna dados para calcular uma agregação mas o resumo resultante está desatualizado)

O problema da leitura não repetitiva (a consulta não repete o mesmo resultado pois os dados foram alterados por outra transação)

Para resolver os problemas acima foram criados níveis de isolamento de transação que são propostos pelo padrão SQL. O padrão SQL define quatro níveis de isolamento de transação em termos de três fenômenos que devem ser evitados entre transações simultâneas. Os fenômenos não desejados são:

dirty read (leitura suja) - A transação lê dados escritos por uma transação simultânea não efetivada (uncommitted).

nonrepeatable read (leitura que não pode ser repetida) - A transação lê novamente dados lidos anteriormente, e descobre que os dados foram alterados por outra transação (que os efetivou após ter sido feita a leitura anterior).

phantom read (leitura fantasma) - A transação executa uma segunda vez uma consulta que retorna um conjunto de linhas que satisfazem uma determinada condição de procura, e descobre que o conjunto de linhas que satisfazem a condição é diferente por causa de uma outra transação efetivada recentemente.

A tabela abaixo apresenta os quatro níveis de isolamento de transação, e seus comportamentos correspondentes:

Nível de isolamento	**Dirty Read**	**Nonrepeatable Read**	**Phantom Read**
Read uncommitted	Possível	Possível	Possível

Nível de isolamento	Dirty Read	Nonrepeatable Read	Phantom Read
Read committed	Impossível	Possível	Possível
Repeatable read	Impossível	Impossível	Possível
Serializable	Impossível	Impossível	Impossível

Podemos alterar o nível de isolamento da transação com o comando SET TRANSACTION, porém este procedimento deve ser feito com cuidado.

Nível de Isolamento Read Committed

O Read Committed é o nível de isolamento padrão do PostgreSQL e na maioria dos SGBDs existente no mercado. Quando uma transação é processada sob este nível de isolamento, o comando SELECT enxerga apenas os dados efetivados antes da consulta começar; nunca enxerga dados não efetivados, ou as alterações efetivadas pelas transações simultâneas durante a execução da consulta. Contudo, comando SELECT enxerga os efeitos das atualizações executadas dentro da sua própria transação, mesmo que ainda não tenham sido efetivadas. Por exemplo, caso uma transação tenha a sequencia de comandos abaixo:

BEGIN WORK;

SELECT * FROM EMPREGADOR WHERE cod_depto = 100;

INSERT INTO EMPREGADOR(..) VALUES (..);

SELECT * FROM EMPREGADOR;

COMMIT WORK;

O segundo comando SELECT enxerga a nova linha incluída pelo comando INSERT pois o mesmo está dentro do escopo da transação ainda que não tenha sido efetivado, confirmado pelo comando COMMIT.

Na verdade, o comando SELECT enxerga um instantâneo do banco de dados, como este era no instante em que a consulta começou a executar. Deve ser observado que dois comandos SELECT sucessivos podem enxergar dados diferentes, mesmo estando dentro da mesma transação, se outras transações efetivarem alterações durante a execução do primeiro comando SELECT.

Veja outro exemplo:

Transação A	Transação B
BEGIN WORK;	
SELECT count(*) FROM CURSOS;	BEGIN WORK;
UPDATE CURSOS WHERE... ;	INSERT INTO CURSOS(...) VALUES (...);
	COMMIT WORK;
SELECT count(*) FROM CURSOS;	
COMMIT WORK;	

No exemplo acima o primeiro comando SELECT da transação A obtém o total de cursos cadastrados. Apos isso uma outra transação B é iniciada e inclui um novo curso e confirma a inclusão com o comando COMMIT WORK. Desta forma, na transação A, o comando UPDATE CURSOS WHERE... ainda não enxergará o novo curso cadastrado, porém o segundo SELECT na transação A conseguirá enxergar o novo curso cadastrado, pois a transação B confirmou a inclusão com o comando COMMIT WORK.

Os comandos UPDATE, DELETE e SELECT FOR UPDATE se comportam do mesmo modo que o SELECT para encontrar as linhas de destino: somente encontram linhas de destino efetivadas até o momento do início do comando. Entretanto, no momento em que foi encontrada alguma linha de destino pode ter sido atualizada (ou excluída ou marcada para atualização) por outra transação simultânea. Neste caso, a transação que pretende atualizar fica aguardando a outra transação de atualização que começou primeiro efetivar ou desfazer (se ainda estiver executando) a alteração na linha.

Se a transação de atualização que começou primeiro desfizer as atualizações, então a segunda transação de atualização pode prosseguir com a atualização da linha original encontrada. Se a transação de atualização que começou primeiro efetivar as atualizações, a segunda transação de atualização ignora a linha caso tenha sido excluída pela primeira transação de atualização, senão tenta aplicar sua operação na versão atualizada da linha.

A condição de procura do comando (a cláusula WHERE) é avaliada novamente para verificar se a versão atualizada da linha ainda corresponde à condição de procura. Se corresponder, a segunda transação de atualização prossegue sua operação começando a partir da versão atualizada da linha.

Como no modo Read Committed cada novo comando começa com um novo instantâneo incluindo todas as transações efetivadas até este instante, de qualquer modo os próximos comandos na mesma transação vão enxergar os efeitos das transações simultâneas efetivadas. O ponto em questão é se, dentro de um único comando, é enxergada uma visão totalmente consistente do banco de dados.

Devido à regra acima, é possível um comando de atualização enxergar um instantâneo inconsistente: pode enxergar os efeitos dos comandos simultâneos de atualização que afetam as mesmas linhas que está tentando atualizar, mas não enxerga os efeitos destes comandos de atualização nas outras linhas do banco de dados. Este comportamento torna o Read Committed inadequado para os comandos envolvendo condições de procura complexas. Entretanto, é apropriado para casos mais simples.

Nível de Isolamento Serializável

O nível Serializable fornece o isolamento de transação mais rigoroso. Este nível emula a execução serial das transações, como se todas as transações fossem executadas uma após a outra, em série, em vez de simultaneamente. Entretanto, os aplicativos que utilizam este nível de isolamento devem estar preparados para tentar executar novamente as transações, devido a falhas de serialização.

Quando uma transação está no nível serializável, o comando SELECT enxerga apenas os dados efetivados antes da transação começar, ou seja qualquer atualização feita após o início da transação é ignorada. Portanto uma transação neste nível de isolamento nunca enxerga dados de alterações não efetivadas ou alterações efetivadas após o início da transação. É diferente do Read Committed, porque o comando SELECT enxerga um instantâneo do momento de início da transação, e não do momento de início do comando corrente dentro da transação. Portanto, comandos SELECT sucessivos dentro de uma mesma transação sempre enxergam os mesmos dados.

Os comandos UPDATE, DELETE e SELECT FOR UPDATE se comportam do mesmo modo que o comando SELECT para encontrar as linhas de destino: somente encontram linhas de destino efetivadas até o momento do início da transação. Entretanto, alguma linha de destino pode ter sido atualizada (ou excluída ou marcada para atualização) por outra transação simultânea no momento em que foi encontrada. Neste caso, a transação serializável aguarda a transação de atualização que começou primeiro efetivar ou desfazer as alterações (se ainda estiver executando). Se a transação que começou primeiro desfizer as alterações, então seus efeitos são negados e a transação serializável pode prosseguir com a atualização da linha original encontrada.

Porém, se a transação que começou primeiro efetivar (e realmente atualizar ou excluir a linha, e não apenas selecionar para atualização), então a transação serializável é desfeita e é exibida uma mensagem informando que não foi possível serializar o acesso devido a atualização simultânea feita por outra

transação. Isto ocorre porque uma transação serializável não pode alterar linhas alteradas por outra transação após a transação serializável ter começado.

Quando o aplicativo receber esta mensagem de erro deverá interromper a transação corrente, e tentar executar novamente toda a transação a partir do início. Da segunda vez em diante, a transação passa a enxergar a alteração efetivada anteriormente como parte da sua visão inicial do banco de dados e, portanto, não existirá conflito lógico em usar a nova versão da linha como ponto de partida para atualização na nova transação.

Isto pode afetar o desempenho do banco de dados pois Uma instrução COMMIT WORK geralmente é executada muito rápido, uma vez que – geralmente – tem muito pouco trabalho a ser executado. Retornar transações, no entanto, normalmente envolve muito mais trabalho para o banco de dados executar, pois, além dos trabalhos já executados até então dentro da transação, deve desfazer os mesmos. Ou seja, se você inicia uma transação que demora 3 minutos para ser executada e, ao final você decide executar um ROLLBACK WORK para retornar à situação inicial, então, não espere que seja executado instantaneamente. Isto poderá demorar facilmente mais do que 3 minutos para restaurar todas as alterações já executadas pela transação. Todo este processamento pode afetar o desempenho do banco de dados.

Deve ser observado que somente as transações que fazem atualizações podem precisar de novas tentativas; as transações somente para leitura nunca estão sujeitas a conflito de serialização.

O modo serializável fornece uma garantia rigorosa que cada transação enxerga apenas visões totalmente consistentes do banco de dados. Entretanto, o aplicativo deve estar preparado para executar novamente a transação quando atualizações simultâneas tornarem impossível sustentar a ilusão de uma execução serial. Como o custo de refazer transações complexas pode ser significativo, este modo é recomendado somente quando as transações efetuando atualizações contêm lógica suficientemente complexa a ponto de produzir respostas erradas no modo Read Committed. Habitualmente, o modo serializável é necessário quando a transação executa vários comandos sucessivos que necessitam enxergar visões idênticas do banco de dados.

É boa prática manter as transações no menor tamanho possível (e viável). Qualquer banco de dados relacional tem que executar muito trabalho para assegurar que transações de diferentes usuários sejam executadas separadamente de forma isolada. Uma conseqüência disto é que partes do banco de dados envolvidas numa transação, freqüentemente precisam se tornar parcialmente inacessível (trancadas), para garantir que as transações sejam mantidas separadas. Ou seja para garantir a consistência dos dados no gerenciamento de transações o acesso aos dados manipulados pelas transações é comprometido e o desempenho geral do banco afetado de forma significativa.

Embora o SGBD gerencie transações no banco de dados automaticamente, uma transação de longa duração geralmente impossibilita que outros usuários acessem os dados envolvidos nesta transação, até que a mesma seja completada ou cancelada.

Imagine que uma aplicação iniciou uma transação assim que a pessoa chegou cedo ao escritório, sentou para trabalhar, e deixou uma transação rodando o dia inteiro enquanto executava diversas alterações no banco de dados. Supondo que esta pessoa efetivou a transação com o comando COMMIT somente no final do expediente, a performance do banco de dados, e a capacidade de outros usuários acessarem os dados desta enorme transação foram severamente impactados. Desta forma, mantenha o escopo das transações pequeno para que tenham curta duração e sejam executadas rapidamente para não afetar o acesso aos dados por outros usuários e sistemas.