Universidade da Beira Interior
Departamento de Informática

Gestão de Transações

Elaborado por:

Luís Almeida Nº13006

Duarte Gonçalves Nº12458

João Antunes Nº12989

Marco Matos Nº13028

Orientador:

Professor Doutor João Muranho

6 de dezembro de 2022

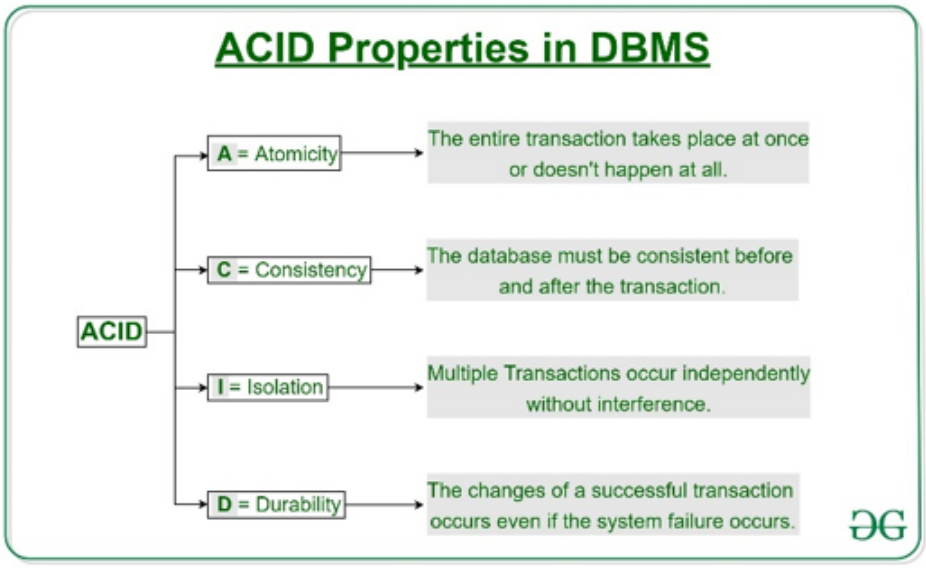
Conteúdo

Capítulo

1

Visão Geral da Gestão de Transações

1.1 As propriedades ACID



1.1.1 Consistência e Isolamento

Consistência: Isto significa que as restrições de integridade devem ser mantidas para que a base de dados seja consistente antes e depois da transação. Refere-se à correção de uma base de dados. Referindo-se ao exemplo acima, O montante total antes e depois da transação deve ser mantido.

Isolamento: Esta propriedade assegura que múltiplas transações podem ocorrer simultaneamente sem levar à inconsistência do estado da base de dados. As transações ocorrem de forma independente, sem interferência. As alterações que ocorrem numa determinada transação não serão visíveis para qualquer outra transação até que essa alteração específica seja escrita na memória ou tenha sido cometida. Esta propriedade assegura que a execução de transações em simultâneo resultará num estado equivalente a um estado alcançado, que foi executado em série em alguma ordem.

1.1.2 Atomicidade e Durabilidade

Atomicidade: Com isto, queremos dizer que ou toda a transação se realiza de uma só vez ou não acontece de todo. Não há meio termo, ou seja, as transações não ocorrem parcialmente. Cada transação é considerada como uma unidade que decorre até à sua conclusão ou não é executada de todo.A atomicidade é também conhecida como a "regra do tudo ou nada". Envolve as duas operações seguintes.

-Abortar: Se uma transação abortar, as alterações feitas à base de dados não são visíveis.

-Comprometer: Se uma transação se compromete, as alterações feitas são visíveis.

Durabilidade: Esta propriedade assegura que, uma vez concluída a transação, as atualizações e modificações da base de dados são armazenadas e gravadas no disco e persistem mesmo que ocorra uma falha do sistema. Estas atualizações tornam-se agora permanentes e são armazenadas em memória não volátil. Os efeitos da transação, portanto, nunca são perdidos.

1.2 Transações e Horários

1.2.1 Transações

Uma transação é uma sequência de múltiplas operações realizadas numa base de dados, onde todas essas operações atuam como uma única unidade lógica, ocorrendo totalmente ou não ocorrendo de todo. Por outras palavras, nunca há um caso em que apenas metade das operações seja executada e os resultados guardados. Quando uma transação de base de dados está em voo, o estado da base de dados pode ser temporariamente inconsistente, mas quando a transação é comprometida ou termina, as alterações são aplicadas.

1.2.2 Horários

Uma série de operações de uma transação para outra é conhecida como horário. É utilizada para preservar a ordem da operação em cada uma das transações individuais.

1.2.3 Horário em série

O horário de série é um tipo de horário em que uma transação é executada completamente antes de se iniciar outra transação. No cronograma de série, quando a primeira transação completa o seu ciclo, então a transação seguinte é executada.

Por exemplo: Suponha que existem duas transacções T1 e T2 que têm algumas operações. Se não tiver intercalação de operações, então há os dois resultados possíveis seguintes:

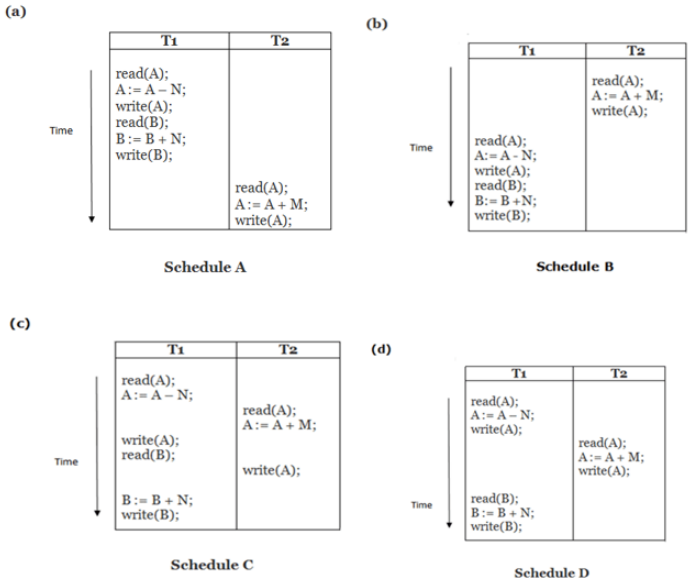
Executar todas as operações de T1 que foram seguidas por todas as operações de T2. Executar todas as operações de T1 que foram seguidas por todas as operações de T2. Na figura (a) dada, o esquema A mostra o esquema em série onde T1 foi seguido por T2. Na figura indicada (b), o esquema B mostra o esquema de série em que T2 é seguido de T1.

1.2.4 Horário não em série

Se for permitida a intercalação de operações, então haverá um horário não seriado. Contém muitas ordens possíveis em que o sistema pode executar as operações individuais das transações. Nas figuras (c) e (d) dadas, o horário C e o horário D são os horários não seriais. Tem intercalação de operações.

1.2.5 Horários serializáveis

A serialização dos horários é utilizada para encontrar horários não seriais que permitam a execução de uma transação sem interferirem com outra. Identifica quais os horários correctos quando as execuções da transacção têm intercalação das suas operações. Uma programação não serial será serializável se o seu resultado for igual ao resultado das suas transacções executadas em série.



1.3 Execução simultânea de transações

Uma transacção é uma unidade de processamento de base de dados que contém um conjunto de operações. Por exemplo, depósito de dinheiro, consulta de saldo, reserva de bilhetes, etc. Cada transacção começa com delimitadores que iniciam a transacção e termina com delimitadores que terminam a transacção. O conjunto de operações dentro destes dois delimitadores constitui uma única transacção.

Há três formas possíveis de executar uma transação. Estas são as seguintes

- Execução em série.
- Execução em paralelo.
- Execução simultânea.

A transacção ou execução simultânea inclui múltiplas transacções que são executadas concomitantemente ou simultaneamente no sistema.

Vantagens As vantagens das transacções simultâneas são as seguintes: aumento da produção, que não é mais do que o número de transações concluídas por unidade de tempo. Reduz o tempo de espera.

Desvantagem : A desvantagem é que a execução de transações simultâneas pode resultar em incoerência.

1.4 Protocolo de Controlo de Concorrência Baseado em Bloqueios

Um bloqueio é uma variável associada a um item de dados que descreve um estado do item de dados em relação a uma possível operação que lhe pode ser aplicada. Sincronizam o acesso por transações simultâneas aos itens da base de dados. É exigido neste protocolo que todos os itens de dados devem ser acedidos de forma mutuamente exclusiva. Neste tipo de protocolo, qualquer transação não pode ler ou escrever dados até que adquira um bloqueio apropriado sobre os mesmos. Bloqueio restrito em duas fases (Strict-2PL)

- Na primeira fase, após a aquisição de todas as fechaduras, a transação continua a ser executada normalmente.
- O protocolo Strict-2PL não liberta uma fechadura depois de a utilizar.
- A Strict-2PL espera até que toda a transação se comprometa, e depois desbloqueia todas as fechaduras de cada vez.
- O protocolo Strict-2PL não tem uma fase de redução de libertação da fechadura.

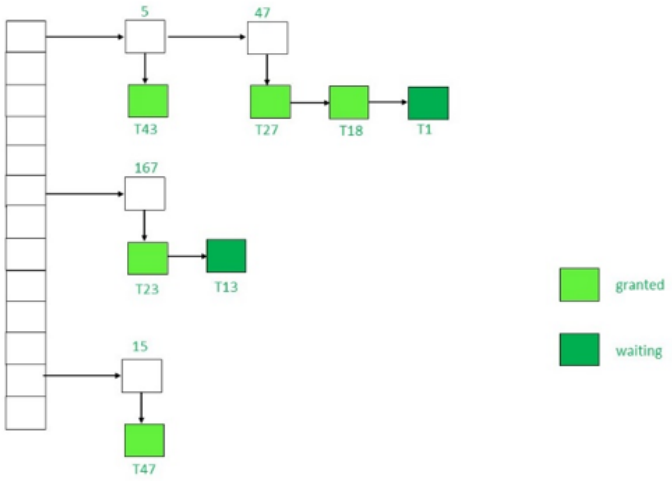
1.5 Desempenho do bloqueio

Os protocolos de bloqueio são utilizados nos sistemas de gestão de bases de dados como meio de controlo da concorrência. Transações múltiplas podem solicitar um bloqueio de um item de dados em simultâneo. Por conseguinte, exigimos um mecanismo para gerir os pedidos de bloqueio feitos por transações. Tal mecanismo é chamado de Administrador de Bloqueio. Baseia-se no processo de passagem de mensagens em que as transações e o gestor de bloqueio trocam mensagens para tratar do bloqueio e desbloqueio dos itens de dados. Estrutura de dados utilizada no Administrador de Bloqueio - A estrutura de dados necessária para a implementação do bloqueio é chamada de Tabela de Bloqueio.

1. É uma tabela de hash onde os nomes dos itens de dados são utilizados como índice de hashing.
2. Cada item de dados bloqueado tem uma lista ligada associada a ele.
3. Cada nó da lista ligada representa a transação que foi solicitada para bloqueio, modo de bloqueio solicitado (mútuo/exclusivo) e estado atual do pedido (concedido/em espera).

4. Cada novo pedido de bloqueio do item de dados será adicionado no final da lista ligada como um novo nó.

5. As colisões na tabela de hash são tratadas através da técnica de encadeamento separado.



1.5.1 Funcionamento do Lock Manager

1. Inicialmente a tabela de bloqueio está vazia, uma vez que nenhum item de dados está bloqueado.

2. Sempre que o gestor de bloqueio recebe um pedido de bloqueio de uma transacção Ti sobre um determinado item de dados Qi, podem surgir casos seguintes:

3. Se o Qi ainda não estiver bloqueado, será criada uma lista ligada e o bloqueio será concedido ao Ti da transacção solicitante.

4. Se o item de dados já estiver bloqueado, será acrescentado um novo nó no final da sua lista ligada contendo a informação sobre o pedido feito por Ti.

5. Se o modo de bloqueio solicitado por Ti for compatível com o modo de bloqueio da transacção que tem actualmente o bloqueio, Ti adquirirá também o bloqueio e o estatuto será alterado para "concedido". Caso contrário, o estado da fechadura da Ti será 'à espera'.

6. Se uma transacção Ti quiser desbloquear o item de dados que tem actualmente, enviará um pedido de desbloqueio ao gestor do bloqueio. O gestor do bloqueio eliminará o nó de Ti desta lista ligada. O bloqueio será concedido à próxima transacção da lista.

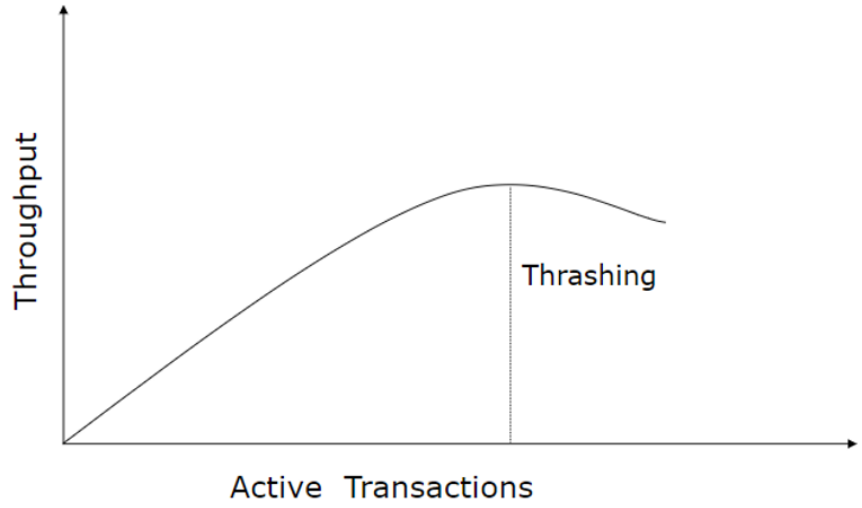
7. Por vezes, a transacção Ti poderá ter de ser abortada. Nesse caso, todo o pedido de espera feito por Ti será eliminado das listas ligadas presentes na tabela de bloqueios. Uma vez concluído o aborto, os cadeados mantidos pela Ti também serão libertados.

1.6 Performace of locking

São projetados para resolver conflitos entre transações e usam dois mecanismos básicos: bloqueio e cancelamento.

Ambos os mecanismos envolvem uma penalidade de desempenho: transações bloqueadas podem conter bloqueios que forçam outras transações a esperar, e abortar e reiniciar uma transação obviamente desperdiça o trabalho feito até agora por aquela transação. bloqueado para sempre, a menos que uma das transações em impasse seja abortada pelo DBMS.

Na prática, menos de 1% das transações estão envolvidas em um impasse e há relativamente poucos abortos. Portanto, a sobrecarga de bloqueio vem principalmente de atrasos devido ao bloqueio. Considere como os atrasos de bloqueio afetam o Taxa de transferência. É improvável que as primeiras transações entrem em conflito, e o Taxa de transferência aumenta proporcionalmente ao número de transações ativas. À medida que mais e mais transações são executadas simultaneamente no mesmo número de objetos de banco de dados, a probabilidade de bloquearem umas às outras aumenta. Assim, os atrasos devido ao bloqueio aumentam com o número de transações ativas e o Taxa de transferência aumenta mais lentamente do que o número de transações ativas. Na verdade, chega um ponto em que adicionar outra transação ativa realmente reduz a taxa de venda; a nova transação é bloqueada e efetivamente compete com (e bloqueia) as transações existentes.



Se um sistema de banco de dados começar a travar, o administrador do banco de dados deve reduzir o número de transações permitidas para serem executadas simultaneamente. Empiricamente, o thrashing ocorre quando 30% das transações ativas são bloqueadas, e um DBA deve monitorar a fração de transações bloqueadas para ver se o sistema está em risco de thrashing. Através do aumento; de três maneiras (além de comprar um sistema mais rápido):

- Bloqueando os objetos de menor tamanho possíveis (reduzindo a probabilidade de que duas transações precisem do mesmo bloqueio)
- Ao reduzir o tempo que a transação mantém os bloqueios (para que outras transações sejam bloqueadas por um tempo menor).
- Reduzindo pontos de acesso. Um ponto de acesso é um objeto database.ge que é acessado e modificado com frequência e causa muitos atrasos de bloqueio. Os pontos quentes podem afetar significativamente o desempenho.

A granularidade do bloqueio é amplamente determinada pela implementação de bloqueio do sistema de banco de dados, e os programadores de aplicativos e o DBA têm pouco controle sobre isso.

1.7 Suporte de transações em SQL

Até agora, estudamos transações e gerenciamento de transações usando um modelo abstrato de uma transação como uma sequência de leitura, gravação e cancelamento/commitações. Agora consideramos o suporte que o SQL fornece para os usuários especificarem o comportamento no nível da transação.

1.7.1 Criação e terminio de transações

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

Depois que uma transação é iniciada, outras instruções podem ser executadas como parte dessa transação até que a transação seja encerrada por um comando COMMIT ou um comando ROLLBACK (a palavra-chave SQL para abortar).

No SQL: 1999, são fornecidos dois novos recursos para dar suporte a aplicativos que envolvem transações de execução longa ou que devem abranger várias transações uma após a outra. Para entender essas extensões, lembre-se de que todas as ações de uma determinada transação são executadas em ordem, independentemente de como as ações de diferentes transações são intercaladas.

O primeiro recurso, chamado de ponto de salvamento, permite identificar um ponto em uma transação e retroceder seletivamente as operações realizadas após esse ponto. Isso é especialmente útil se a transação realizar tipos de operações hipotéticas e desejar fazer as alterações ou manter as alterações com base nos resultados. Isso pode ser feito definindo pontos de salvamento.

SQL:1999 Nested Transactions: O conceito de uma transação como uma sequência atômica de ações foi estendido no SQL:1999 através da introdução do recurso savepoint. Isso permite que partes de uma transação sejam revertidas seletivamente. A introdução de pontos de salvamento representa o primeiro suporte SQL para o conceito de transações aninhadas, que foram extensivamente estudadas na comunidade de pesquisa. A idéia é que uma transação pode ter várias subtransações aninhadas, cada uma das quais pode ser revertida seletivamente. Os pontos de salvamento suportam uma forma simples de inesting de um nível.

Em uma transação de longa duração, podemos definir uma série de pontos de salvamento. O comando de ponto de salvamento nos permite dar um nome a cada ponto de salvamento:

-savepdint(savepoint name)

Um comando subsequente de retrocesso pode especificar o ponto de salvamento para o qual reverter:

-Rollback to savepoint(savepoint name)

Se definirmos três pontos de salvamento A, B e Cin nessa ordem e, em seguida, revertermos para A, todas as operações desde A serão desfeitas, incluindo a criação dos pontos de salvamento Banda C. De fato, o próprio ponto de salvamento A é feito quando revertemos para ele, e devemos estabelecê-lo (através de outro ponto de salvamento e) se quisermos ser capazes de reverter

para ele novamente.

É instrutivo comparar o uso de pontos de salvamento com a alternativa de executar uma série de transações (ou seja, tratar todas as operações entre dois pontos de salvamento consecutivos como uma nova transação). O mecanismo de ponto de salvamento oferece duas vantagens. Primeiro, podemos reverter vários pontos de salvamento. Na abordagem alternativa, podemos reverter apenas a transação mais recente, que é equivalente a reverter para o ponto de salvamento mais recente. Em segundo lugar, a sobrecarga de iniciar várias transações é evitada.

Mesmo com o uso de pontos de salvamento, certos aplicativos podem exigir que realizemos várias transações uma após a outra. Para minimizar a sobrecarga em tais situações, o SQL:1999 apresenta outro recurso, chamado transações encadeadas, podemos executar ou reverter uma transação e iniciar outra transação imediatamente. Isso é feito usando as palavras-chave opcionais AND CHAIN nas instruções COMMIT e ROLLBACK.

1.8 O que devemos dar Lock?

Até agora, discutimos transações e controle de simultaneidade em um modelo abstrato no qual o banco de dados contém uma coleção fixa de objetos e cada transação é uma série de operações de leitura e gravação em objetos individuais. Uma questão importante a considerar no contexto do SQL é o que o DBMS deve tratar como um objeto ao definir bloqueios para determinada instrução SQL (que faz parte de uma transação).

Considere a seguinte Query:

SELECT S.rating, MIN (S.age)
FROM Sailors S
WHERE S.rating = 8

Suponha que essa query seja executada como parte da transação T1 e uma instrução SQL que modifique a idade de um determinado marinheiro, digamos Joe, com classificação=8 executada como parte da transação T2. Quais 'objetos' o DBMS deve bloquear ao executar essas transações? Intuitivamente, devemos detetar um conflito entre essas transações.

O DBMS poderia definir um bloqueio compartilhado em toda a tabela Sailor para T1 e definir um bloqueio exclusivo em Sailors para T2, o que garantiria que as duas transações fossem executadas de maneira separada. No entanto, essa abordagem gera uma simultaneidade lenta e podemos melhorar bloqueando objetos menores, refletindo o que cada transação realmente acessa. Assim, o DBMS poderia definir um bloqueio compartilhado em cada linha com rating = 8 para a transação T1 e definir um bloqueio exclusivo ape-

nas na linha para a tupla modificada para a transação T2. Agora, outras transações somente leitura que não envolvem linhas = 8 podem continuar sem esperar por T1 ou T2.

Como este exemplo ilustra, o DBMS pode bloquear objetos em diferentes granularidades: podemos bloquear tabelas inteiras ou definir bloqueios de nível de linha. A última abordagem é adotada nos sistemas atuais porque oferece um desempenho muito melhor. Na prática, embora o bloqueio no nível da linha seja geralmente melhor, a escolha da granularidade do bloqueio é complicada. Nós discutimos esta questão mais adiante na Seção 17.5.3.

Um segundo ponto a observar é que as instruções SQL acessão conceitualmente uma coleção de linhas descritas por um predicado de seleção. No exemplo anterior, a transação T1 acessa todas as linhas com rating=8. sugerimos que isso poderia ser resolvido definindo bloqueios compartilhados em todas as linhas em Marinheiros com rating = 8.

Infelizmente, isso é um pouco simplista demais. Para saber por que, considere uma instrução SQL que insere um novo marinheiro com rating=8 e é executada como transação T3. Suponha que o DBJiIS defina bloqueios compartilhados em cada linha Sailor existente com rating=8 para Tl. Isso não impede que a transação T3 crie uma linha com rating=8 e defina um bloqueio exclusivo nesta linha. Se essa nova linha tiver um valor de idade menor do que as linhas existentes, Tl retornará uma resposta que depende de quando foi executada em relação a T2. No entanto, nosso esquema de bloqueio não impõe nenhuma ordem relativa nessas duas transações.

Esse fenômeno é chamado de problema phantom: uma transação recupera uma coleção de objetos (em termos SQL, uma coleção de tuplas) duas vezes e vê resultados diferentes, embora não modifique nenhuma das configurações em si. Para evitar fantasmas, o DBMS deve bloquear conceitualmente todas as linhas possíveis com rating=8 em nome de T1. Uma maneira de fazer isso é bloquear toda a tabela, ao custo da simultaneidade de fluxo. Para evitar phantoms, o DBMS deve bloquear conceitualmente todas as linhas possíveis com rating=8 em nome de T1. Uma maneira de fazer isso é bloquear toda a tabela, ao custo da simultaneidade de fluxo.

Pode ser que o aplicativo que invoca T1 possa aceitar o potencial de precisão devido a fantasmas. Nesse caso, a abordagem de configuração de bloqueios compartilhados em listas imutáveis existentes para T é adequada e oferece melhor desempenho.

1.9 Características das transações em SQL

Para dar aos programadores controle sobre a sobrecarga de bloqueio incorrida por suas transações, o SQL permite que eles especifiquem três características de uma ação de 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 registadas; não discutiremos mais esse recurso.

Se o modo de acesso for READ ONLY, a transação não tem permissão para modificar as bases de dados. Assim, INSERT, DELETE, UPDATE e CREATE comandos não podem ser executados. Se tivermos de executar um desses comandos, o modo de acesso deve ser para READWRITE. Para as transações com o modo de acesso READ ONLY apenas bloqueios compartilhados precisam ser obtidos, aumentando assim a simultaneidade. O nível de isolamento controla a extensão em que determinada transação é exposta às ações de outras transações executadas simultaneamente. Ao escolher uma das nossas possíveis configurações de nível de isolamento, um usuário pode obter maior simultaneidade ao custo de aumentar a exposição da transação a alterações não confirmadas de outras transações.

As opções de nível de isolamento são READUNCOMMITTED, READCOMMITTED, REPEATABLE READ e SERIALIZABLE. O efeito desses níveis é resumido na Figura 16.10. Nesse contexto, leitura suja e leitura irrepetível são definidas como de costume.

Level	Dirty Read	Unrepeatable Read	
READ UNCOMMITTED	Maybe	Maybe	Maybe
READ COMMITTED	No	Maybe	Maybe
REPEATABLE READ	No	No	Maybe
SERIALIZABLE	No	No	No

O mais alto grau de isolamento dos efeitos de outras transações é obtido definindo o nível de isolamento para uma transação T como SERIALIZABLE. Este nível de isolamento garante que T lê apenas as alterações feitas por transações confirmadas, que nenhum valor lido ou escrito por T seja alterado por qualquer outra transação até que seja concluído e que, se T lê for um conjunto de valores com base em alguma condição de pesquisa, esse conjunto não será alterado por outras transações até que seja concluído (ou seja, T evita o fenômeno phantom).

Dentro de uma implementação baseada em bloqueio, uma transação SERIALIZABLE obtém bloqueios antes de ler ou gravar objetos, incluindo bloqueios em conjuntos de objetos que precisam ser e os mantém até o fim.

REPEATABLE READ garante que T leia apenas as alterações feitas por tran-

sações confirmadas e nenhum valor lido ou escrito por T seja alterado por qualquer outra transação até que esteja completo. No entanto, T poderia experimentar o fenômeno phantom; por exemplo, enquanto T examina todos os registos de marinheiros com rating = 1, outra transação pode adicionar um novo registo de marinheiro, que é perdido por T.

Uma transação REPEATABLE READ define o mesmo bloqueio como uma transação SERIALIZABLE, exceto que ela não faz bloqueio de índice; ou seja, ela bloqueia apenas objetos individuais, não conjuntos de objetos. READ COMMITTED garante que T leia apenas as alterações feitas por transações confirmadas e que nenhum valor gravado por T seja alterado por qualquer outra transação até que T esteja completo. No entanto, um valor lido por T pode muito bem ser modificado por outra transação enquanto T ainda está em andamento e T está exposto ao problema phantom.

Uma transação READ COMMITTED obtém bloqueios exclusivos antes de escrever objetos e mantém esses bloqueios até o fim. Também obtém bloqueios compartilhados antes de ler os objetos, mas esses bloqueios são liberados imediatamente; seu único efeito é garantir que a última transação que modificou o objeto seja concluída.

Uma transação READ UNCOMMITTED pode ler alterações feitas em um objeto por uma transação em andamento; obviamente, o objeto pode ser alterado ainda mais enquanto está em andamento e também está vulnerável ao problema phantom. O nível de isolamento SERIALIZABLE é geralmente o seguro e é recomendado para a maioria das transações. Algumas transações, no entanto, podem ser executadas com um nível de isolamento mais baixo, e o menor número de bloqueios solicitados pode contribuir para melhorar o desempenho do sistema.

O nível de isolamento e o modo de acesso podem ser definidos usando o comando SET TRANSACTION. Por exemplo, o comando a seguir declara que a transação atual deve ser SERIALIZABLE e ONLY READ:

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE READ ONLY

Quando uma transação é iniciada, o padrão é SERIALIZABLE e READ WRITE.

1.10 Introdução ao Crash Recovery

O recovery manager de um DBMS é responsável por garantir a atomicidade e durabilidade da transação. Ele garante a atomicidade ao desfazer as ações das transações que não são confirmadas e a durabilidade ao garantir que todas as ações das transações confirmadas sobrevivam a falhas do sistema (por exemplo, um despejo causado pelo erro do user) e media failures (por exemplo, um disco corrompido).

Em seguida, um DBMS, é reiniciado após travamentos. Depois disso, o recovery manager deve trazer o banco de dados. 'para um estado consistente. O recovery manager também é responsável por desfazer as ações de uma transação abortada.

O manager de transações de um DBMS controla a execução das transações. Antes de ler e gravar objetos durante a execução normal, os bloqueios devem ser adquiridos (e liberados posteriormente) de acordo com um protocolo de bloqueio escolhido.

1.10.1 Stealing Frames and Forcing Pages

Duas abordagens para escrever objetos:

• Steal Approach: Mudanças feitas em um objeto no Buffers pool por uma transação são gravadas em disco antes da transação ser confirmadas. Essas gravações são executadas quando outra transação deseja trazer uma página. E dizemos que a segunda transação rouba um frame da primeira transação.

• Force Approach: quando uma transação é confirmada, todas as alterações feitas em objetos no buffer pool são imediatamente forçadas para o disco.

Do ponto de vista da implementação de um gerenciador de recuperação, é mais simples usar um gerenciador de buffer com uma abordagem forçada e sem roubo. Se for usada uma abordagem no-steal, não precisamos desfazer as alterações de uma transação abortada. Se for usada uma abordagem force, não precisamos refazer as alterações de transações confirmadas. No entanto, essas políticas têm desvantagens importantes:

• A abordagem no-steal assume que todas as páginas modificadas por transações em andamento podem ser acomodadas no buffer pool e, na presença de grandes transações, essa suposição é irrealista.

• A abordagem force resulta em custos excessivos de Número de leituras ou gravações de páginas no disco. Se uma página muito usada for atualizada sucessivamente por 20 transações, ela será gravada no disco 20 vezes.

1.10.2 Passos de recuperação normais durante a execução

O manager de recuperação de um DBMS mantém algumas informações durante execução de transações para permitir que ele execute sua tarefa no caso de uma falha. Em particular, um log de todas as modificações na base de dados é salvo em armazenamento estável, que é garantido para sobreviver a falhas de sistema e falhas de visualização.

O armazenamento estável é implementado mantendo várias cópias de informações em dispositivos de armazenamento não voláteis, como discos ou

fitas. Conforme discutido anteriormente, é importante garantir que o log que descrevem uma alteração na base de dados são gravadas no armazenamento estável antes a mudança é feita; caso contrário, o sistema pode travar logo após a alteração, deixando-nos sem um registo da mudança. (Lembre-se que este é o Write-Ahead Log, ou WAL, propriedade.) O log permite que o gerenciador de recuperação desfaça as ações de abortado e transações incompletas e refazer as ações de transações confirmadas. Por por exemplo, uma transação que foi confirmada antes da falha pode ter feito atualizações para uma cópia no buffer pool, e essa mudança pode não ter sido gravada no disco antes da falha, devido a uma abordagem no-force. Assim as alterações devem ser identificadas usando o log e gravadas no disco. Além disso, mudanças de transações que não foram confirmadas antes da falha podem ter sido gravadas para o disco devido a uma abordagem de steal. Tais mudanças devem ser identificadas usando o log e depois desfeito.

A quantidade de trabalho envolvido durante a recuperação é proporcional às mudanças feitas por transações confirmadas que não foram gravadas no disco no momento do acidente. Para reduzir o tempo de recuperação de uma falha, o DBMS atualiza periodicamente as páginas do buffer para o disco durante a execução normal usando um plano de fundo de processo (certificando-se de que todas as entradas de log que descrevam as alterações das páginas são gravadas primeiro no disco, ou seja, seguindo o protocolo WAL).

1.10.3 ARIES

ARIES Recovery algorithm projetado para trabalhar com uma abordagem steal e no-force. Existem 3 fases no protocolo de ARIES Recovery:

- Análise: Examine o log forward (do ponto de verificação mais recente) para identificar todos os Xacts que estavam ativos e todas as páginas sujas no buffer pool no momento da falha.
- Redo: Refaz todas as atualizações em páginas sujas no buffer pool, conforme necessário, para garantir que todas as atualizações registadas sejam de fato executadas e gravadas no disco.
- Desfazer: As gravações de todos os Xacts que estavam ativos na falha são desfeitas (restaurando o valor anterior da atualização, conforme encontrado no log), funcionando de trás para frente no log.

Por fim, o banco de dados reflete apenas as ações das transações confirmadas.

1.10.4 Atomicity: Implementing Rollback

É importante reconhecer que o subsistema de recuperação também é responsável por executará o comando ROLLBACK, que aborta uma única transação. De facto, a lógica (e o código) envolvida em desfazer uma única transação é idêntica àquela usado durante a fase desfazer na recuperação de uma falha do sistema. Todos os registos de log para uma determinada transação são organizados em uma lista encadeada e podem ser eficientemente acessadas na ordem inversa para facilitar a reversão da transação.

Capítulo

2

Controlo de concorrência

2.1 Bloqueio em Duas Fases

Diz-se que uma transação segue o protocolo de bloqueio de duas fases se o bloqueio e o desbloqueio puderem ser feitos em duas fases.

Fase de crescimento: Podem ser adquiridos novos bloqueios nos itens de dados, mas nenhum pode ser libertado. Fase de Encolhimento: Os bloqueios existentes podem ser libertados, mas não podem ser adquiridos novos bloqueios. Vejamos uma transação implementando 2-PL. Onde existem dois tipos de bloqueios: partilhados S(a) e exclusivos X(a).

Nota - Se for permitida a conversão do bloqueio, então a atualização do bloqueio(de S(a) para X(a)) é permitida na fase de crescimento, e a descida do bloqueio (de X(a) para S(a)) deve ser feita na fase de Encolhimento.

	T_1	T_2
1	lock-S(A)	
2		lock-S(A)
3	lock-X(B)	
4	.....	.....
5	Unlock(A)	
6		Lock-X(C)
7	Unlock(B)	
8		Unlock(A)
9		Unlock(C)
10	.....	.....

Esta é apenas uma transacção de esqueleto que mostra como o desbloqueio e o bloqueio funcionam com 2-PL. Nota para:

Transacção T1:

- A fase de crescimento é dos passos 1-3.
- A Fase de contracção é dos passos 5-7.
- Ponto de Bloqueio a 3

Transacção T2:

- A fase de crescimento é das etapas 2-6.
- A fase de contracção é dos passos 8-9.
- Ponto de Fechadura a 6

Lock Point - O Ponto em que termina a fase de crescimento, ou seja, quando uma transação leva o fecho final de que necessita para continuar o seu trabalho. Agora olhe para o horário, certamente compreenderá.

Como dissemos anteriormente o 2-PL assegura a seriação, mas ainda há alguns inconvenientes do 2-PL que são: Rollbacks em cascata em 2-PL e Deadlocks e Starvation.

2.2 Deadlock em 2-PL

Considerando este exemplo, será fácil de compreender. Digamos que temos duas transações T1 e T2.

Schedule: Lock-X1(A) Lock-X2(B) Lock-X1(B) Lock-X2(A)

Ao desenhar o gráfico de precedência, pode detetar o laço. Portanto, o bloqueio também é possível em 2-PL.

O bloqueio em duas fases pode também limitar a quantidade de simultaneidade que ocorre numa programação, porque uma transação pode não ser capaz de libertar um item depois de o ter utilizado. Isto pode ser devido aos protocolos e outras restrições que podemos colocar no cronograma para assegurar a serialidade, liberdade de bloqueio, e outros fatores. Este é o preço que temos de pagar para assegurar a serialidade e outros fatores, pelo que pode ser considerado como uma pechincha entre a simultaneidade e a manutenção das propriedades ACID.

2.3 Lock Management

Os sistemas de gestão de bases de dados empregam protocolos de bloqueio para gerir a simultaneidade. O bloqueio de um item de dados pode ser solicitado por várias transacções ao mesmo tempo. Como resultado, precisamos de um sistema para tratar os pedidos de bloqueio de transacções. Lock Manager é o nome deste mecanismo. O bloqueio e desbloqueio de itens de dados é tratado através do processo de passagem de mensagens, no qual o gestor de bloqueio e as transacções trocam mensagens.

Estrutura de dados utilizada no Lock Manager - A tabela Lock é o nome da estrutura de dados necessária para implementar o bloqueio.

1. O nome dos itens de dados é utilizado como o índice de hash nesta tabela de hash.

2. Uma lista ligada é ligada a cada item de dados bloqueado.

3. Cada um dos nós da lista ligada representa uma transacção que solicitou um bloqueio, o tipo de bloqueio (mútuo ou exclusivo), e o estado actual do pedido (concedido/em espera).

4. Cada novo pedido de bloqueio de um item de dados será adicionado como um novo nó ao fim da lista ligada.

5. As colisões da tabela de hastes são tratadas utilizando a técnica de encadeamento separada.

Como funciona o Lock Manager

1. Uma vez que nenhum item de dados está bloqueado no início, a tabela de bloqueio está vazia.

2. As seguintes situações podem ocorrer sempre que o gestor de bloqueios recebe um pedido de bloqueio de uma transação Ti sobre um item de dados específico Qi:

   • Será criada uma lista ligada e será dado um bloqueio ao Ti da transação requerente, se o Qi ainda não estiver bloqueado.

   • Se o item de dados já estiver bloqueado, um novo nó contendo detalhes sobre o pedido feito por Ti será acrescentado no final da sua lista ligada.

3. Se o modo de bloqueio solicitado por Ti for aceitável para o modo de bloqueio da transação que já se encontra na posse do bloqueio, Ti também obterá o bloqueio, e o estatuto mudará para "concedido". Caso contrário, o cadeado de Ti será marcado como "em espera".

4. Uma transação Ti notificará o gestor do cadeado com um pedido de desbloqueio se este pretender libertar o item de dados que está atualmente na posse. O nó de Ti será removido desta lista ligada pelo gestor do bloqueio. A transação seguinte na lista receberá um bloqueio.

5. A Transaction Ti pode ocasionalmente precisar de ser abandonada. Neste cenário, todos os pedidos pendentes de Ti serão removidos das listas interligadas da tabela de fechaduras. As fechaduras que Ti tem na sua posse serão libertadas assim que o aborto estiver terminado.

2.4 Lock Conversion

Em palavras simples, os lock Conversion resultam da conversão de um tipo de fechadura para outro. Podemos ter três tipos de fechaduras de conversão e são listadas abaixo com uma breve explicação;

- Partilhado com a Intent Update (SIU) : Uma transacção que detém um bloqueio Partilhado também tem algumas páginas ou linhas bloqueadas com um bloqueio de Actualização.

- Partilhado com a Intent Exclusive (SIX) : Uma transacção que detém um cadeado Partilhado também tem algumas páginas ou linhas bloqueadas com um cadeado Exclusivo.

- Actualização com Intent Exclusive (UIX) : Uma transacção que detém um bloqueio de Actualização também tem algumas páginas ou linhas bloqueadas com um Bloqueio Exclusivo.

2.5 Gestão de locks

Os sistemas de gestão de bases de dados empregam protocolos de bloqueio para gerir a simultaneidade. O bloqueio de um item de dados pode ser solicitado por várias transações ao mesmo tempo.

Como resultado, precisamos de um sistema para tratar os pedidos de bloqueio de transações. Lock Manager é o nome deste mecanismo. De um modo geral, existem três formas de lidar com os bloqueios:

- Prevenção de impasses;
- Deteção e recuperação de impasses- Abortar um processo ou antecipar alguns recursos quando são detetados bloqueios.
- Ignorar o problema todos juntos - Se os bloqueios só ocorrem uma vez por ano ou mais, pode ser melhor simplesmente deixá-los acontecer e reinicializá-los conforme necessário do que incorrer nas constantes despesas gerais e penalidades de desempenho do sistema associadas à prevenção ou deteção de bloqueios.

2.6 Conversão de locks

Uma transação pode precisar de adquirir uma fechadura exclusiva sobre um objeto para o qual já detém uma fechadura partilhada. Por exemplo, uma instrução de update SQL poderia resultar na colocação de bloqueios partilhados em cada linha de uma tabela.

Se uma linha satisfaz a condição (na cláusula WHERE) para ser atualizada, deve ser obtido um bloqueio exclusivo para essa linha.

Infelizmente, não impede bloqueios causados por dois pedidos de atualização conflituosos. Para exemplo, se duas transações que possuem um lock partilhado num objeto pedirem ambas uma atualização para um cadeado exclusivo, isto conduz a um impasse (dead lock).

Uma melhor abordagem é evitar completamente a necessidade de atualizações de fechaduras, obtendo fechaduras exclusivas inicialmente, e a descida para uma fechadura partilhada uma vez que seja claro que isto é suficiente. A abordagem de rebaixamento reduz a simultaneidade, obtendo blo-

queios de escrita em alguns casos em que não são necessários. Mas reduz o número de deadlocks.

2.7 Impasse (DeadLock)

O impasse é uma situação em que um conjunto de processos é bloqueado porque cada processo está a reter um recurso e à espera de outro recurso adquirido por algum outro processo.

Considere-se um exemplo quando dois comboios se aproximam um do outro na mesma via e só existe uma via, nenhum dos comboios pode mover-se uma vez que estejam um em frente do outro. Uma situação semelhante ocorre em sistemas operacionais quando existem dois ou mais processos que detêm alguns recursos e aguardam recursos detidos por outros(s).

Existem quatro condições que são necessárias para se chegar a um impasse:

- Mutual Exclusion -Pelo menos um recurso deve ser mantido num modo não partilhável. Infelizmente, alguns recursos, tais como impressoras, requerem acesso exclusivo através de um único processo.

- Hold and Wait - Um processo deve ser simultaneamente a detenção de pelo menos um recurso e a espera de pelo menos um recurso que está atualmente a ser detido por algum outro processo. Para evitar esta condição os processos devem ser impedidos de possuir um ou mais recursos enquanto se espera simultaneamente por um ou mais outros.

- No Preemption - Uma vez que um processo esteja em posse de um recurso então esse recurso não pode ser retirado desse processo até que o processo o liberte voluntariamente.A preempção da atribuição de recursos do processo pode evitar esta condição de impasse, quando tal é possível.

- Circular Wait - Deve existir um conjunto de processos P0, P1, P2, . . ., PN de tal forma que cada P[i] esteja à espera de P[(i + 1) % (N + 1)]. Uma forma de evitar a espera circular é numerar todos os recursos, e exigir que os processos solicitem recursos apenas em ordem estritamente crescente (ou decrescente).

2.8 Handling DeadLocks

A fim de se livrar de impasses, o OS verificam periodicamente o sistema quanto a qualquer bloqueio. No caso de encontrar algum dos bloqueios, o sistema operativo recuperará o sistema utilizando algumas técnicas de recuperação como:

- Regresso a um estado seguro(Rollback) - No momento em que entrarmos em impasse, vamos fazer retroceder todas as atribuições para entrarmos no estado seguro anterior. Para este fim, o SO precisa de implementar checkpoints.

- Anular um processo - Normalmente o sistema operativo mata um processo que até agora tem feito o mínimo de trabalho.

Esta não é uma abordagem sugestionável, mas pode ser implementada se o problema se tornar muito grave.

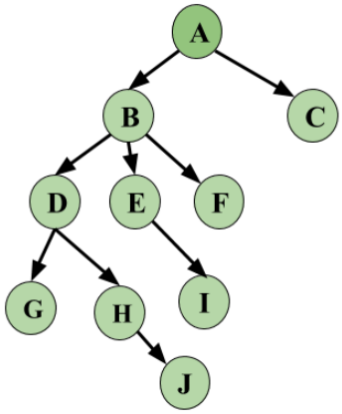
2.9 Controlo de Concorrência

O Controlo de Concorrência é o procedimento de gestão que é necessário para controlar a execução simultânea das operações que têm lugar numa base de dados.

A questão é que a execução simultânea que é realizada deve ser feita de forma intercalada, e nenhuma operação deve afetar as outras operações de execução, mantendo assim a consistência da base de dados. Assim, ao fazer a execução simultânea das operações de transação, ocorrem vários problemas desafiantes que precisam de ser resolvidos.

2.9.1 Protocolo Baseado em árvores

Seguir o Protocolo baseado em árvores assegura a schedule livre de impasses. Não precisamos de esperar por desbloquear um item de Dados como fizemos no protocolo 2-PL, aumentando assim a concorrência. Se di -> dj então qualquer transacção que aceda a di e dj deve aceder a di antes de aceder a dj.

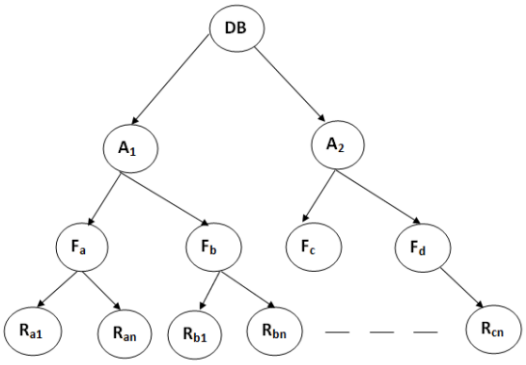


2.9.2 Protocolo de granularidade múltipla

O protocolo de Granularidade Múltipla aumenta a simultaneidade e reduz a sobrecarga de bloqueio.

Mantém o rasto do que bloquear e como bloquear.

Torna fácil a decisão de bloquear um item de dados ou desbloquear um item de dados. Este tipo de hierarquia pode ser representado graficamente como uma árvore.



2.9.3 Controlo de Concorrência sem bloqueios

Controlo optimista de concorrência - assume que múltiplas transacções podem frequentemente ser concluídas sem interferirem umas com as outras (sem adquirir bloqueios).

Controlo de concorrência com base no tempo- É utilizado em algumas bases de dados para manusear transacções com segurança, utilizando o tempo.

Controlo de concorrência de multivariedade - é uma técnica de optimização de bases de dados que cria cópias duplicadas de registos para que os dados possam ser lidos e actualizados em segurança ao mesmo tempo.

Capítulo

3

Crash Recovery

O que é que acontece se um sistema de base de dados der Crash?

Quando se fala de o Crash de uma base dados geralmente refere-se a qualquer tipo de bugs ou mau funcionamento de hardware no sistema operacional ou no software da base de dados. Este pode interromper o processamento da transação e até causar a perda de conteúdo que reside em armazenamento volátil, como memória principal, memória de cache, RAM entre outras.

Devido a este problema faz se a Introdução ao Recovery Manager of a DBMS

O recovery manager de um DBMS é responsável por garantir 2 propriedades importantes de transações: Atomicidade e Durabilidade. Este garante Atomicidade desfazendo as ações das transações que ainda não estão committed e a Durabilidade fazendo com que as ações de transações committed sobrevivem o Crash do sistema. Existem diversos algoritmos de Recovery, no entanto iremos falar do algoritmo ARIES.

3.1 Log

O Log/Registro é a história de ações executadas pela DBMS (Database Management System). Fisicamente o Log é um ficheiro de registos guardado em armazenamento estável, em que este é esperado que sobreviva a Crashes. Esta durabilidade é alcançada mantendo duas ou mais cópias do Log em discos diferentes (possivelmente até mesmo em localizações diferentes), para que a chance de todas as copias do Log serem simultaneamente perdidas é insignificante. A parte mais recente do Log, é chamada de log tail ou "rabo"do Log, este é mantido na memória principal e periodicamente é forçado para o ar-

mazenamento estável. Desta maneira, registos Log e dados são escritos para o disco com a mesma granularidade (páginas ou conjunto de páginas).

Um Log é escrito para cada uma das seguintes ações:

- Updating a Page: Depois de modificar uma página, um registo do tipo "update"é adicionado na Log tail;
- Commit: Quando uma transação decide dar commit, dá um force-write a um registo do tipo "commit"contendo o id da transação;
- Abort: Quando uma transação é abortada, um registo do tipo "abort"é adicionado ao Log contendo o id da transação, e uma ação Undo é iniciada para esta transação.
- End: Como mencionado anteriormente, quando uma transação é aborted ou committed, são necessárias ações adicionais para além de adicionar um registo Log de "abort"ou "commit". Depois destes passos adicionais estarem completos, um registo do tipo "end"contendo o id da transação é adicionado ao Log.
- Undoing an update: Quando uma transação é revertida (porque a transação é abortada, ou durante a recuperação de um Crash), os seus updates são desfeitos. Adicionalmente quando uma ação descrita por um Log do tipo "update"é desfeita, um "compensation log record", ou CLR, é escrito.

Para além do Log, existem duas tabelas que contém informação importante relacionadas com a recuperação do sistema DPT e TT, estas vão ser exploradas posteriormente.

3.2 Estruturas Relacionadas à Recuperação

Para reunir as informações necessárias para os Logs, duas estruturas de dados devem de ser mantidas: a Dirty Page Table (DPT) e a Transaction Table (TT).

• Transaction Table: Esta tabela contém uma entrada para cada transação ativa. Esta entrada contém (entre outras coisas) o id da transação, o estado da transação, e lastLSN. Isto é, a tabela contém todas as transações que estão atualmente em execução e o SN (Sequence Number) da última entrada de Log que elas criaram. O Estado da transação pode ser, in progress, committed ou aborted. (Nos últimos dois casos, a transação irá ser removida da tabela após certos passos de 'clean up' sejam completados).

• Dirty Page Table: Esta tabela mantém o registo de todas as páginas que foram modificadas e ainda não gravadas no disco, e o primeiro SN que fez com que a página ficasse "dirty".

3.3 WAL

Write-Ahead Log ou WAL é um protocolo que desempenha um papel muito importante no que toca a garantir, que os Logs de todas as alterações feitas da base de dados, estão disponíveis enquanto é feita uma tentativa de recuperações depois de um Crash. WAL é um protocolo que descreve determinada ação realizada pelo sistema, esta ação é que, antes de uma página ser escrita para o disco, todos os Logs do tipo update que descrevem uma alteração nesta página tem de ser forçados para armazenamento estável.

3.4 CHECKPOINTING

Um checkpoint é basicamente um "snapshot"do estado da DBMS, e fazendo checkpoints periodicamente, reduz a quantidade de trabalho que a DMBS irá ter de fazer durante o processo de restart subsequente a um Crash.

Checkpointing em ARIES (Termo mencionado posteriormente) ocorre em 3 passos:

- 1º Passo: Begin-checkpoint é registado de modo a indicar quando é que o checkpoint começa.
- 2º Passo: End-checkpoint este é construído incluindo os conteúdos da Transaction Table (TT) e da Dirty Page Table (DPT), e são anexados ao Log.
- 3º Passo: O último passo é executado após o registo do end-checkpoint ser escrito para armazenamento estável. Enquanto o end-checkpoint está a ser construído, o DMBS continua a executar transações e a escrever outros Logs. A única garantia é que a Transaction Table (TT) e Dirty Page Table (DPT) são precisas a partir do momento do registo do begin-checkpoint.

3.5 ARIES

O algoritmo ARIES ou Algorithms for Recovery and Isolation Exploiting Semantics baseia-se no logging/registo de todas as operações da base de dados

com número de sequência crescentes. Normalmente o arquivo de logs/registos é armazenado no chamado "armazenamento estável", que é um meio de armazenamento que é esperado conseguir sobreviver a Crashes e falhas de hardware.

Os 3 principais pontos por de trás de ARIES:

- Write-ahead Logging (WAL): Como descrito anteriormente, qualquer alteração num objeto é primeiro guardada no Log, e o Log precisa de ser escrito para o armazenamento estável antes que as alterações no objeto sejam escritas para o disco.
- Repeating history during Redo: Quando o sistema é reiniciado depois de um Crash, ARIES refaz as ações da base de dados antes do Crash e traz o sistema para o estado exato em que este estava antes da falha.
- Logging changes during Undo: Alterações feitas na base de dados ao desfazer transações, são registadas no Log para garantir que tal ação não é repetida na eventualidade de o sistema reinicializar repetidamente.

Quando o Recovery Manager é invocado depois de um Crash, o processo de restart é executado em 3 fases:

1. Analise: Identifica dirty pages no buffer pool (i.e., alterações que ainda não foram escritas para o disco) e transações ativas no momento do Crash.
2. Redo: Repete todas as ações, começando por um sítio em específico no Log, e restaura o estado da base de dados para como estava antes do Crash.
3. Undo: Desfaz todas as ações das transações que não estão committed, para que a base de dados reflita apenas as ações de transações que estão committed.

Capítulo

4

Refêrencias

- https://bizfluent.com/about-6525978-role-information-systems-organization.html
- https://emeritus.org/in/learn/the-6-types-of-information-systems-and-their-applications/
- https://www.youtube.com/watch?v=S2jiDuN4k70
- www.javatpoint.com
- www.geeksforgeeks.org
- http://redbook.cs.berkeley.edu/redbook3/
- Ramakrishnan - Database Management Systems 3rd Edition