IB - O que o faz diferente

Interbase: O que o faz diferente ?
por Bill Todd
Tradução e Adaptação : Carlos H. Cantu (Interbase-BR)

Quando voce entra no mundo Client/Server a primeira coisa que voce tem que fazer é selecionar um servidor SQL. As arquiteturas de servidores de banco de dados variam amplamente e como resultado o comportamento deles em uma determinada situação também varia. Isso significa que para selecionar o servidor correto para sua aplicação você tem que entender duas coisas :

Como os dados serão acessados e modificados em sua aplicação.
Como o servidor se comportará em cada acesso aos dados ou situação de atualização.

A diferença mais significante entre o Interbase e outros servidores de banco de dados é sua arquitetura multi-geracional. A arquitetura Multi-geracional também é chamada arquitetura de versões e para entender como ela é diferente, você precisa explorar os diferentes métodos de controlar o acesso simultâneo aos dados em um ambiente multi-usuário.

Esquemas de travamento

O mais antigo e mais comum método de controlar o acesso simultâneo aos dados por vários usuários é o travamento. Quando um usuário trava um objeto em um banco de dados , ele restringe a habilidade de outros usuários ter acesso àquele objeto. Quando uma trava afeta a concorrência em uma tabela inteira, ela restringe o acesso de outros usuários em todos os registros da tabela. Portanto, uma trava de tabela tem uma granularidade muito baixa. Uma trava à nível de página na tabela limita o acesso a todos os registros pertencentes àquela página. Uma trava à nível de página é mais granular do que uma trava à nível de tabela. Em contraste, uma trava atribuída à um simples registro é muito granular e provê a restrição mínima ao acesso de dados simultâneo.

Alguns servidores de banco de dados suportam tanto travamento de registros por páginas como à nível de registro. O problema com o travamento de página é fácil de se verificar se você considerar um exemplo : Suponha que o tamanho de uma página é 2k (2048 bytes) e o tamanho de um registro é 100 bytes. Então cada página pode armazenar 20 registros e cada vez que uma página é travada é travado o acesso à 20 registros. No travamento à nível de registro, somente um único registro seria travado e outros usuários estariam livres para ter acesso aos outros registros na página, portanto uma travamento de registro provê uma concorrência melhor.

Se voce está acostumado com a utilização de banco de dados desktop então voce provavelmente está familiarizado com o esquema pessimista de travamento. O travamento pessimista é chamado assim porque assume que a probabilidade de outro usuário tentar modificar o mesmo objeto no banco de dados que você está alterando é alta. Em um ambiente de travamento pessimista, o objeto que você deseja alterar é bloqueado antes que você começe a alterá-lo e permanece travado até que você

confirme (commit) sua alteração. A vantagem do travamento pessimista é que você voce garante que poderá postar o registro alterado.

Suponha que você precise alterar o endereço de um cliente. Usando o travamento pessimista voce inicialmente bloquearia o registro à nível de página ou de registro. Você pode então ler o registro do cliente, altera-lo e terá certeza que poderá gravar suas mudanças ao banco de dados. Depois das alterações terem sido gravadas, sua trava é liberada e outros estarão livres para alterar esse registro. Os bloqueios podem persistir por muito tempo quando o travamento pessimista é utilizado. Fica óbvio que o uso de travas com maior granulidade é mais interessante no travamento pessimista em um ambiente multi-usuário. Se você tem que travar uma página inteira de registros de clientes enquanto você está alterando um único registro, nenhum outro usuário poderá alterar qualquer outro registro naquela página. Travamento à nível de registro são melhores quando o travamento pessimista for usado, pois ela impõe menos restrições de acesso por outros usuários. O travamento à nível de página é menos satisfatória pois ela restringe o acesso à muitos registros enquanto a trava existir.

O esquema de travamento mais comum encontrado em servidores de banco de dados é o travamento otimista. O mecanismo de trava é otimista de maneira que ele assume que é muito improvável que outro usuário tentará mudar o mesmo registro que você está alterando. Uma trava otimista não é colocada até que você tente gravar(commit) suas alterações. para entender o travamento otimista, considere dois usuários, John e Jane, tentando alterar o registro de um cliente. Primeiro John lê o registro e começa a fazer alterações. Depois Jane lê o registro e começa a fazer alterações. Isto é possível porque no esquema otimista não é colocada nenhuma trava quando um usuário começa a alterar um registro. Depois que John completa as alterações e tenta grava-las, o banco de dados trava o registro, grava as mudanças e libera a trava. Quando Jane tenta gravar suas mudanças, o software descobrirá que o registro já foi alterado após a leitura inicial e suas alterações serão rejeitadas. A Jane tem que re-ler o registro e começar novamente.

O método otimista tem uma vantagem clara onde os bloqueios só são mantidos durante o curto período de tempo em que os dados estão sendo atualizados. Isso significa que com um esquema otimista você pode alcançar uma concorrência adequada com menos granularidade de trava. Sendo assim, bancos de dados que usam o sistema otimista podem utilizar bloqueios à nível de página e não de registro.

Do ponto de vista do fabricante de banco de dados, o método de bloqueio de página é vantajoso pois menos travas devem ser colocadas, particularmente durante operações em batch que afetam muitos registros. Isto significa que as exigências de recursos do módulo gerenciador de travas no sistema de administração de banco de dados são mais baixas e isto pode ajudar a melhorar o desempenho do servidor de banco de dados. Porém, os usuários invariavelmente são a parte mais lenta de qualquer aplicação de banco de dados, assim você normalmente melhorará o desempenho global em um ambiente onde um usuário não pode bloquear outro.

Entender como seu banco de dados administra travas é extremamente importante. Considere uma tabela de orders (notas fiscais). Continuamente estão sendo adicionados novos registros quando novas orders são recebidas. Devido ao fato que os dados da order não incluem um campo ou campos que formariam uma chave primária natural, você decide usar um número de order artificialmente gerado como uma chave substituta. Serão gerados números de orders sequenciais conforme as orders forem

recebidas. Considerando que sua aplicação freqüentemente precisará selecionar grupos de orders, você cria um índice agrupado na coluna do número de order. Um índice agrupado provê desempenho superior ao trazer registros adjacentes porque os registros são armazenados fisicamente na ordem da chave dentro das páginas de banco de dados.

Infelizmente é provável que este design produza um desempenho pobre se o banco de dados usar o método de travamento de páginas. Como chaves sequenciais adjacentes estão sendo recuperadas e um índice agrupado(clustered) está sendo usado, cada registro novo que é adicionado provavelmente será armazenado na mesma página que o registro anterior. Como nas travas à nível de página dois usuários não podem acrescentar orders novas ao mesmo tempo na mesma página, cada order nova tem que esperar até a trava de página colocada pela order anterior seja libertada. Neste caso você obteria um desempenho maior gerando números aleatórios para as chaves para reduzir a chance dos registros serem acrescentados na mesma página.

Transações

Outra exigência de servidores de banco de dados é a habilidade para se agrupar alterações ao banco de dados em transações. Transações de atualização consistem em uma ou mais alterações a uma ou mais tabelas no banco de dados, que devem ser tratadas como um todo, de forma que ou todas as alterações são aplicadas ou nenhuma delas terão efeito.

O processamento de transações acontece em 3 fases. Primeiro você diz ao software de banco de dados que você deseja iniciar uma transação. Isto informa o sistema de banco de dados que todas as alterações serão tratadas como uma única unidade até notificação adicional. Depois, as alterações são aplicadas de fato às tabelas no banco de dados. Finalmente, você notifica o sistema de banco de dados que você deseja confirmar/commit ou desconsiderar/rollback a transação. Se você confirma a transação, então todas as alterações dentro dessa transação se tornarão permanentes. Se você desconsiderar/rollback a transação todas as alterações serão desfeitas.

O processamento de transações é vital para assegurar a integridade lógica do banco de dados. Considere o caso onde o John transfere $100 da poupança dele para a sua conta corrente. Esta transação procederia como segue :

Comece uma transação nova.
Atualize o saldo na poupança mostrando uma retirada de $100.
Atualize o saldo na conta corrente mostrando um aumento de $100.
Confirme ou desconsidere a transação.

Suponha que o sistema falhe após o segundo passo mas antes do passo 3. Sem o controle transacional, John teria perdido $100. Com o controle transacional, quando o sistema é reiniciado, o software de banco de dados vai automaticamente dar um rollback em qualquer transação que não tinha sido confirmada na hora da falha. Isto garante que o banco de dados mantenha um estado consistente dos dados.

Você também precisa de controle transacional para transações de leitura onde mais de um registro serão lidos e uma visão consistente dos dados deve ser mantida. Esta exigência é

descrita em mais detalhe na próxima seção.

O Isolamento transacional diz como as transações que estão sendo executadas simultaneamente interagem entre si. Muitos dos servidores de banco de dados de hoje foram projetados para processar transações de atualização com tempo muito curto intermixado com leituras únicas de registro. O exemplo perfeito é uma ATM de um banco (o BANCO 24 HORAS). Um BANCO 24 HORAS lê o saldo de uma única conta ou atualiza o saldo em uma ou mais contas. Neste ambiente, transações são muito curtas e as leituras envolvem um único registro de cada vez, portanto o isolamento transacional aqui não é uma preocupação muito séria. Porém, muitas das aplicações de banco de dados de hoje não se ajustam nesse modelo.

Transações de atualização curtas ainda são a norma, mas o advento de sistemas de informação executivos introduziram longas transações de leitura que percorrem tabelas inteiras e as vezes bancos de dados inteiros. Para entender o problema com transações de leitura longas, considere o seguinte cenário :

Um executivo pede o valor total do inventário da companhia. Enquanto a Query está varrendo a tabela de inventário, um usuário move uma qtde de barras de platina do armazém A para o armazém B e confirma a transação. É possível que a Query conte a platina em ambos os armazéns produzindo um relatório de inventário errado. A pergunta se torna então : Que atualizações uma transação de leitura deveria ver e quando deveria ver ? Isto é controlado pelo nível de isolamento transacional. Há quatro níveis de isolamento definidos no SQL 92 padrão, como segue :

Leitura não confirmada (Read Uncommitted) - este nível de isolamento, algumas vezes chamada de leitura-suja (dirty read), permite que qualquer registro do banco de dados possa ser lido, independente de já ter sido commitado ou não.

Leitura Confirmada (Read Committed) - este nível permite que transações de leitura vejam somente os registros que já foram commitados.

Leitura repetitiva (Repeatable Read) - Assegura que se a transação re-lê os registros que já foram lidos anteriormente, ela encontrará os mesmos valores em todos os campos nessa segunda leitura. Isso não quer dizer que o número de registros que retornará na segunda leitura será o mesmo que o da primeira. Por exemplo, se uma outra transação insere um registro que satisfaça o critério da cláusula WHERE no seu select você verá este registro novo na segunda vez sua transação executa o SELECT.

Serial (Serializable) - Assegura que se voce executar um mesmo select diversas vezes durante a duração de uma transação, voce verá exatamente o mesmo número de registros com os mesmos valores nos campos. Também assegura que há uma ordem serial de transações concorrentes que produzirão o mesmo resultado como o produzido por uma execução concorrente. O isolamento Snapshot do Interbase provê o mesmo nível de isolamento das alterações feitas por outras transações seriais.

No exemplo anterior, voce precisa do isolamento Serial para assegurar que seu relatório de inventário seja consistente. O problema é o preço que você tem que pagar para ter esse tipo de isolamento em um banco de dados que usa arquitetura de travas. Com o modelo de trava, o único modo de assegurar que aqueles dados não mudarão durante uma transação de leitura longa é impedir que qualquer atualização aconteça até que a transação de leitura termine, travando cada registro conforme eles são lidos. Isso em inúmeras vezes é inaceitável.

Versioning

Há outro modelo para controle de concorrência chamado versioning que supera os problemas que o modelo de travas têm quando o ambiente consiste em uma mistura de transações de leitura longas e atualizações. Este modelo é chamado de versioning e é o modelo usado pelo Interbase. Para entender esse modelo, considere o exemplo anterior. Em um banco de dados usando versioning, a transação de leitura para produzir o relatório de inventário se inicia. Quando a transação de atualização para mover a platina do armazém A para o B é confirmada uma nova versão do registro é criado no BD.

Em um banco de dados de versioning, cada transação recebe um número de transação sequencial. Além disso, o gerenciador de banco de dados mantém um inventário de todas as transações ativas. Esse inventário de transações mostra quando uma transação está ativa, confirmada (commited) ou anulada (rollback).

Quando uma transação de atualização commitar, o software de banco de dados confere para ver se há transações com o número de transação mais baixo ainda ativas. Nesse caso uma versão nova do registro é criada que contém os valores atualizados. Cada versão também contém o número de transação da transação que o criou. Veja ainda que o Interbase não cria uma cópia completa do registro. Ao invés disso, ele cria um registro de diferença que só contém os campos que foram alterados.

Quando uma transação de leitura inicia, ela recebe o próximo número de transação e uma cópia da página de inventário de transações que mostra o status de todas as transações que ainda não foram commitadas. Quando a transação reclama cada registro de uma tabela, o BD checa se o número de transação da última versão do registro é maior que o número de transação da transação que está reclamando o registro e se a transação estava commitada quando a transação de leitura se iniciou. Se o número de transação da última versão do registro é maior que o número de transação da transação requerente ou se a transação que criou a última versão do registro estava ativa quando a transação de leitura se iniciou, o Interbase olha para trás através das versões prévias do registro até encontrar um onde o número da transação seja menor que o número de transação da transação que está tentando ler o registro e cujo status estava commitado quando a transação de leitura se iniciou. Quando o gerenciador do banco de dados acha a versão mais recente que satisfaça estes critérios ele retorna aquela versão. O resultado é um isolamento de transação serial sem proibir atualizações durante a vida da transação de leitura.

Considere o exemplo seguinte de um registro onde 4 versões existem:

Tran=100 (status=committed)

Tran=80 (status=ativo quando a leitura começou)

Tran=60 (status=rolled back)

Tran=40 (estado = commitado quando leitura começou)

Assuma que uma transação de leitura com o número de transação 90 tenta ler esse registro. A transação de leitura não verá a versão do registro criada pela transação 100 porque a atualização que criou esta versão aconteceu depois que transação 90 começou. A transação 90 também não poderá ler a versão criada pela transação 80 embora ela tenha um número de transação mais baixo, porque a transação 80 ainda não commitou. Embora a versão para transação 60 ainda exista, ela foi desconsiderada e transações desconsideradas sempre são ignoradas. Então, a versão que a transação 90 lerá é a versão criada pela transação 40.

Note neste exemplo que a transação 80 não terá permissão para commitar. Quando a transação 80 tentar commitar, o sistema de BD verá que a transação 100 já commitou e a transação 80 será desconsiderada (rollback).

Como eles se comparam

Para ter um entendimento maior de como os modelos de travamento e versioning se comparam voce precisa ver os tipos de conflitos de concorrência que podem ocorrer em um ambiente multi-usuário e qual o comportamento de cada modelo em cada caso. Os exemplos seguintes assumem que o modelo de travamento usa uma trava de leitura compartilhada e uma trava com exclusividade para escrita para implementar um sistema otimista. Múltiplos usuários podem colocar taravas de leitura mas ninguém pode colocar uma trava de escrita se outro usuário já realizou uma trava de escrita ou de leitura. Se um usuário tem uma trava de escrita, então outros usuários não poderão ler ou gravar no registro. Isso é tipico de BD que utilizam arquitetura de travamento. Nos exemplos seguintes, 3 cenários são explorados. O primeiro é o que acontece quando não há controle de concorrência, o segundo é o que acontece quando se usa travas e o terceiro utiliza versioning.

Considere o caso onde um marido e a esposa vão à 2 Bancos 24 horas ao mesmo tempo sacar dinheiro de sua conta corrente. Sem um controle de concorrência a sucessão seguinte de eventos poderia acontecer :

John lê o saldo da conta que é de $1,000..

Jane lê o saldo da conta que ainda é $1,000.

John posta uma retirada de $700 .

Jane posta uma retirada $500 .

Neste momento o saldo da conta é - $200 e o banco não ficará muito feliz com isso. Isto aconteceu porque sem mecanismo de controle de concorrência, Jane não vê a mudança de saldo feita pelo saque do John em sua conta. Com o modelo de travas :

John lê o saldo da conta que causa uma trava de leitura.
Jane lê o saldo da conta que causa uma trava de leitura.
John posta o saque dele que falha devido à trava de leitura de Jane.
Jane posta o saque dela que falha devido à trava de leitura de John.

Um deadlock acaba de ocorrer. Com sorte, o software de banco de dados descobrirá o deadlock e ferá um rollback de uma das transações.

Usando o modelo de versões (versioning):

John lê o saldo da conta.
Jane lê o saldo da conta
John posta a retirada dele que faz com que uma nova versão com o novo saldo seja escrita.
Jane posta a retirada dela, mas um rollback é executado quando a versão mais nova é detectada.

Um problema diferente acontece se um usuário cancela a transação. Sem controle de concorrência :

John retira dinheiro da conta que atualiza o saldo.
Jane lê o saldo.
John cancela a operação antes que ela seja commitada.
Jane está vendo o saldo errado agora.

Neste caso, uma dependência existe entre as duas transações. A transação de Jane só produz os resultados corretos se a transação de John commitar. Isto mostra o perigo de se usar o isolamento uncommitted. Usando o modelo de travamento :

John lê o saldo que coloca uma trava de leitura.
John faz a retirada dele, o que coloca uma trava de gravação durante a atualização. Jane lê o saldo, o que tenta colocar uma trava de leitura mas tem que esperar devido a trava de John.

John cancela a transação antes de commitar. Isso gera um rollback que libera a trava de escrita.
Jane pode ler agora e adquirir o saldo correto.

Usando o modelo de versões (versioning) :

John retira dinheiro que atualiza o saldo e cria uma nova versão não commitada.
Jane lê o saldo. O saldo não mostra a retirada de John pois ainda não foi commitada.
John cancela a transação, o que gera um rollback na nova versão.

Isto mostra uma vantagem de desempenho do modelo de versões, pois Jane não teve que ficar esperando para ler o saldo.

O próximo exemplo é diferente mas usa o mesmo cenário que o exemplo anterior onde as barras de platina foram movidas de um armazém para outro. Sem o controle de concorrência :

John pede o total de todas as contas.
Jane transfere dinheiro da poupança para a conta corrente enquanto a transação de John está em andamento.
John vê o total errado.

Aqui a análise dos dados está incompatível porque o estado dos dados não foi preservado ao longo da vida da transação de leitura. Com o modelo de travas :

John pede o total de todas as contas que gera uma trava de leitura.
Jane transfere dinheiro mas não pode colocar uma trava de escrita devido à trava de leitura colocada por John. Jane tem que esperar até o fim da transação de leitura.
John vê o total certo, liberta a trava de leitura e a transação de Jane procede.

No modelo de versões (versioning) :

John pede o total de todas as contas.
Jane transfere dinheiro da poupança para a conta corrente resultando em novas versões
John vê o total certo e a atualização de Jane não tem que esperar.

Uma outra variação do problema de serialização acontece se você precisar re-ler os dados durante uma transação. Por exemplo:

Retorne todos os registros que satisfaçam uma condição.
Outro usuário insere um novo registro que satisfaz o critério anterior.
Se voce repetir a query, voce terá um resultado contendo o novo registro. O aparecimento deste registro fantasma não é consistente dentro da transação.

Com um banco de dados que usa o modelo de travas, o único modo para prevenir esta inconsistência é colocar uma trava de leitura para a tabela inteira durante a duração da transação. Assim a sucessão de eventos é:

Coloque uma trava de leitura na tabela.
Retorne todos os registros que satisfaçam a condição.
Outro usuário tenta inserir um registro mas é bloqueado pela trava da tabela.
Repita a query e você vai adquirir os mesmos resultados porque outros usuários não puderam cometer alterações.

Usando versioning não há nenhum problema porque o registro recentemente inserido tem um número de transação mais alto que a transação de leitura e assim é ignorado nas futuras repetições da query que são parte da mesma transação.

De longe voce pode ver que o modelo de versioning gerencia melhor a concorrência do que o modelo de travas. No entanto, isso não é verdade em todos os casos. Considere o seguinte caso onde o John e Jane tentam fazer seus salários serem iguais :

John lê o salário de John.
Jane lê o salário de Jane.
John fixa o salário de Jane igual para John.
Jane fixa o salário de John igual para Jane.

O resultado é que o salário de John e de Jane são trocados. Usando o modelo de travas voce pode prevenir isto travando ambos os registros.

John lê seu salário e coloca uma trava de leitura.
Jane lê seu salário e coloca uma trava de leitura.
John fixa o salário de Jane igual ao dele mas não pode commitar devido à trava de leitura de Jane.
Jane fixa o salário de John igual ao dela mas não pode commitar devido à trava de leitura de John.

Uma vez mais você tem uma paralisação completa que o sistema de banco de dados deveria solucionar rolando uma transação para trás. Outra solução, no modelo de travas, é colocar uma trava de escrita para toda a tabela, como no seguinte cenário:

John trava a tabela para escrita.
John lê o salário dele.
Jane tenta ler o salário dela mas é bloqueada pela trava de John.
John fixa o salário de Jane igual ao dele e libera a trava de escrita.
A transação de Jane agora pode proceder.

Usando o versioning:

John lê o salário dele.
Jane lê o salário dela.

John fixa o salário de Jane igual ao dele e commita.
Jane fixa o salário de John igual ao dela e commita.

Considerando que o versioning permite que ambas as transações processem concorrentemente, uma vez mais os salários são trocados. O único modo para resolver este problema com o modelo de versioning é o seguinte :

John lê o salário dele.
Jane lê o salário dela.
John fixa o salário de Jane igual ao dele.
John fixa o salário dele mesmo criando uma versão mais nova.
Jane fixa o salário de John igual ao dela mas gera um rollback pois uma versão mais nova já existe.

Aqui o problema é resolvido fixando o salário de John para ele mesmo. Isto força a criação de uma versão de registro nova para o salário de John. A arquitetura de versioning não permitirá commitar uma mudança quando uma versão do registro sendo atualizado existe e foi criada depois do início da transação atual. A transação de Jane será cancelada (rollback).

Recuperação

Um assunto muito importante em aplicações de banco de dados é tempo de recuperação se o servidor quebra. Não importa quanto robusto for o seu hardware e o software e qual o poder do seu no-break, sempre há bancos de dados que se recuperarão automaticamente quando o servidor for reiniciado, mas há uma diferença significante do tempo que a recuperação levará se muitas transações estavam ativas na hora em que o servidor quebrou.

Bancos de dados que utilizam o modelo de travas escrevem cada transação em um arquivo de log. Para se recuperar depois da quebra, o BD tem que ler o arquivo de log e dar um rollback em todas as transações que estavam ativas naquela hora, copiando as informações do arquivo de log para o BD.

Um banco de dados de versioning não tem um arquivo de log. As versões de registro no banco de dados provêem toda a informação necessária para a recuperação. Nenhum dado precisa ser copiado de um lugar a outro. Ao invés disso, quando o gerenciador de banco de dados volta a ficar on-line, ele simplesmente percorre o inventário de transações e altera o estado de todas as transações ativas para canceladas (rolled back). Isto levará no máximo alguns segundos mesmo com um número grande de transações ativas. Essa é outra área onde o modelo de versioning é superior.

O que acontece se um dos HDs onde está o BD parar de funcionar ? Nos modelos de BD com travas, voce pode colocar o arquivo de LOG em outro HD e logar todas as transações feitas desde o último backup. Se um desastre físico ocorrer no HD, voce pode recuperar o último backup e usar o arquivo de LOG para relançar o movimento. Nos BDs utilizando versioning, a única proteção para isso é criar um mirror do BD em outro dispositivo (HD).

Voce pode utilizar o recurso de shadowing do Interbase para fazer isso, use drives RAID ou use um sistema de mirror do próprio sistema operacional.

Ampliando o Banco de dados

Enquanto alguns bancos de dados requerem a paralisação do BD para aumentar o tamanho do arquivo de log e para descarregar e recarregar os dados, o Interbase não necessita disso. Bancos de dados Interbase se expandem automaticamente até o tamanho de máximo permitido pelo sistema de arquivo, conforme os dados vão sendo inseridos. Se o BD precisa de mais espaço, voce pode adicionar um arquivo secundário através do comando ALTER DATABASE. Enquanto o ALTER DATABASE requer uso exclusivo do banco de dados, ele é muito rápido pois não exige que se descarregue e recarregue os dados.

Outros Assuntos

No princípio pode parecer que um banco de dados de versioning tem uma desvantagem significante pois as múltiplas versões dos registros farão com que o tamanho do banco de dados aumente rapidamente comparado com um BD que utiliza o modelo de travas. Enquanto que isso é verdade, não esqueça que esses outros BDs também crescem na medida que seus arquivos de LOGs aumentam.

Porém, bancos de dados de versioning certamente crescerão rapidamente se algo não for feito para controlar a proliferação de versões de registro. O gerenciador de banco de dados executa automaticamente um pouco da administração doméstica para você. Cada vez que um registro é acessado, o gereciador do banco de dados confere para ver se quaisquer das versões anteriores daquele registro não é mais necessária. Uma versão não é mais necessária se uma transação foi cancelada (rolled back) ou se existe uma versão mais nova já commitada e não há nenhuma transação ativa com um número de transação menor que o número de transação da versão commitada mais nova. Versões que não são mais necessárias são apagadas automaticamente e o espaço que elas ocuparam nas páginas do banco de dados é re-utilizado.

Muitos registros em muitos bancos de dados são acessados infreqüentemente. Para remover versões desnecessárias destes registros, o banco de dados deve ser varrido periodicamente. Uma operação de varredura visita todos os registros em todas as tabelas no banco de dados e apaga versões desnecessárias. Você pode rodar a varredura enquanto o banco de dados estiver em uso mas ela implicará numa queda de performance enquanto estiver sendo executada.

O Interbase, pode default, começará uma varredura automaticamente depois de 20.000 transações. Isso não é o melhor modo para administrar um sweep porque você não tem nenhum controle de quando a varredura começará e o usuário que começa a transação que disparou a varredura é travado até o final da varredura. É melhor ativar o sweep manualmente quando ninguém estiver utilizando o BD, ou através de um BACKUP/RESTORE. Isto é porque o Interbase não removerá versões de registros mais velhos que a mais Velha Transação Interessante (OIT). A OIT é a mais velha transação cujo estado não seja o de commitado. O estado de todas as transações desde que a OIT é mantida

na TIP e as versões de registros para estas transações é retido no banco de dados. Executando uma varredura quando você tem uso exclusivo do banco de dados ou executando um backup/restore reajustará o OIT à última transação commitada e removerá todas as transações anteriores da TIP e removerá as versões delas do banco de dados. Isto não só libera o máximo de espáco no banco de dados, mas também reduz o tamanho da TIP. Desde que a TIP deve ser copiada para cada transação quando ela se inicia, fazer com que a TIP seja pequena fará com que as transações iniciem mais rapidamente.

O Interbase requer menos de 10 megabytes de espaço de disco e é muito fácil de instalar e configurar. Há só dois parâmetros que você precisa fixar. O primeiro é o tamanho de página do banco de dados e o segundo é o tamanho do cache das páginas. Uma vez que estes parâmetros são definidos, o Interbase se ajusta para prover máximo desempenho. Instalação fácil e manutenção fazem do Interbase uma boa escolha para instalações onde não existe um administrador de banco de dados. Finalmente, o Interbase roda em uma grande variedade de plataformas que o tornam facilmente escalável.

O Interbase também suporta múltiplos character sets por tabela, o que o faz uma escolha ideal para aplicações internacionais. O Interbase foi o pioneiro no armazenamento eficiente de dados de BLOBs e suporta múltiplos campos BLOB por banco de dados bem como VIEWs atualizáveis.

Conclusão

Selecionar o banco de dados certo para sua aplicação requer uma compreensão clara dos tipos de transações, isolamentos e atualizações. O sistema de versioning tem uma vantagem clara porque pode processar transações de leitura e escrita concorrentemente e ainda provê o isolamento serializável para garantir segurança. O versioning também provê uma rápida recuperação depois de falhas pois não há arquivos de LOG para serem processados. Quando um BD versioning re-starta, ele só marca as transações abertas e que não foram commitadas como rolled-back e está pronto para continuar.

Quando esse artigo foi escrito, o IB era o mais novo servidor de BD e o único à utilizar o modelo de versioning. Além das vantagens do modelo de versioning, o Interbase tem o menor tamanho e necessita de menos memória, é auto-configurável e está disponível para o Netware, Windows NT, Windows 9x, Linux, Solaris sendo altamente escalável.