Depois de implementar o aplicativo cliente/servidor, você poderá encontrar áreas em que gostaria de melhorar seu desempenho. Você pode ajustar o aplicativo para obter desempenho máximo, por exemplo, acelerando os formulários e consultas e aumentando a produtividade de dados.

Este capítulo aborda as estratégias de otimização para o desempenho de aplicativos no cliente, na rede e no servidor. Para obter informações sobre como implementar aplicativos cliente/servidor, consulte os capítulos anteriores deste manual.

Este capítulo aborda os seguintes tópicos:

- Otimizando o uso de conexões
- Acelerando a recuperação de dados
- Acelerando consultas e visualizações
- Acelerando formulários
- Melhorando o desempenho em atualizações e exclusões

Otimizando o uso de conexões

Estabelecer uma conexão gasta tempo e memória tanto no cliente quanto no servidor. Quando você otimiza as conexões, equilibra a necessidade de alto desempenho com as exigências de recursos do seu aplicativo.

O número de conexões usadas pelo Visual FoxPro depende do fato de você forçar ou não o fechamento de conexões não usadas e de como você define o espaço do tempo limite inativo de uma conexão.

Utilizando conexões compartilhadas

Você pode usar conexões de forma exclusiva ou compartilhar uma conexão.. Quando você usa uma conexão de forma exclusiva, o seu aplicativo não apresenta contenções para os recursos de conexão quando uma conexão é estabelecida. Se cada conjunto de resultados utilizar uma conexão exclusiva, você também poderá combinar o processamento assíncrono em vários conjuntos de resultados.

Quando você usa uma conexão compartilhada, tem uma conexão para vários conjuntos de resultados. Você deve serializar as operações de manipulação de dados nos conjuntos de resultados que compartilham a mesma conexão e criar o aplicativo para testar a conexão sempre que puder ocorrer um conflito. Para obter informações sobre como compartilhar uma conexão, consulte o capítulo 8, Criando visualizações.

Controlando o tempo limite da conexão

Se o seu aplicativo ficar inativo por longos períodos de tempo, você poderá reduzir o uso da conexão, definindo a propriedade IdleTimeout na conexão. A propriedade IdleTimeout controla o intervalo de tempo em que as conexões podem ficar inativas antes de serem fechadas pelo Visual FoxPro. Como padrão, as conexões esperam indefinidamente e não são desativadas até serem fechadas especificamente pelo usuário.

Estabeleça o tempo de inatividade de uma definição de conexão com a propriedade IdleTimeout da função DBSETPROP( ); você pode definir a propriedade IdleTimeout para uma conexão ativa com a função SQLSETPROP( ).

O Visual FoxPro fecha as conexões mesmo que formulários e janelas Pesquisar que exibam dados remotos ainda estejam abertos e, em seguida, os reconecta automaticamente quando a conexão é necessária novamente. No entanto, o Visual FoxPro não poderá fechar uma conexão se:

- Os resultados de uma consulta do servidor estiverem pendentes.
- A conexão estiver no modo de transação manual. Você deve gravar ou desfazer a transação e mudar para o modo de transação automático antes da conexão ser fechada.

O modo de transação é definido para uma conexão com a propriedade Transactions da função DBSETPROP( ); você pode definir o modo de transação para uma conexão ativa com a função SQLSETPROP( ).

Liberando conexões

Você pode melhorar o desempenho fechando as conexões que o aplicativo não está mais utilizando. As conexões são fechadas automaticamente quando você fecha uma visualização. Se a conexão for compartilhada por diversas visualizações, o Visual FoxPro fechará a conexão quando a última visualização que utiliza a conexão for fechada.

Você pode controlar manualmente a conexão para uma consulta se não quiser atualizar os dados em um cursor. Utilize uma consulta de passagem SQL para selecionar os dados que você precisa em um cursor local, depois feche a conexão.

Acelerando a recuperação de dados

Você pode acelerar a recuperação de dados gerenciando o número de linhas carregadas durante uma *carga progressiva*, controlando o tamanho de carga e utilizando a carga de Memo com pausa.

Você também pode utilizar a propriedade de visualização UtilizeMemoSize para retornar campos de caracteres como campos Memo e, em seguida, desativar FetchMemo para permitir que o aplicativo carregue seletivamente os campos de caracteres convertidos em campos Memo.

Utilizando carga progressiva

Quando você consulta uma fonte de dados remota, o Visual FoxPro recupera linhas completas de dados e constrói um cursor do Visual FoxPro. Para acelerar a recuperação de dados remotos, o Visual FoxPro utiliza a carga progressiva de cursores de visualização e cursores criados de forma assíncrona com a passagem SQL. Em vez de solicitar que você ou o aplicativo aguardem a recuperação de um conjunto de dados inteiro, o Visual FoxPro executa uma consulta e carrega apenas um pequeno subconjunto das linhas do conjunto de resultados para o cursor local. O tamanho deste subconjunto é 100 linhas como padrão.

Observação As instruções síncronas de passagem SQL não utilizam carga progressiva. O conjunto de resultados inteiro solicitado por uma instrução SQLEXEC( ) é recuperado antes que o controle volte para o aplicativo.

Como o Visual FoxPro recupera linhas adicionais de dados, o cursor local contém cada vez mais dados consultados. Como as linhas são recuperadas da fonte de dados em momentos diferentes, as informações nas linhas não são atualizadas automaticamente. Se a sua conexão estiver operando em modo assíncrono, o Visual FoxPro retornará o controle para você e para seu programas assim que carregar o primeiro subconjunto de dados. Durante o tempo inativo, o Visual FoxPro executa uma carga em segundo plano das linhas restantes nos dados consultados, um subconjunto de cada vez, para o cursor local. Este cenário permite que você utilize os dados já carregados no cursor sem ter que esperar pelo restante dos dados.

Observação Aumentar o número de linhas carregadas melhora o desempenho, mas diminui a velocidade de resposta da interface do usuário. Diminuir o número de linhas carregadas tem o efeito inverso.

Carregando dados quando solicitados

Você pode desativar o carregamento progressivo e carregar linhas apenas em uma base quando necessário utilizando a propriedade de banco de dados e cursores de visualização FetchAsNeeded. Isso pode resultar em uma recuperação de dados mais eficiente para visualizações remotas ou visualizações que recuperam conjuntos de resultados extremamente grandes.

A propriedade FetchAsNeeded é definida como padrão como falsa (.F.), o que significa que o carregamento progressivo é empregado como padrão. Quando você define a propriedade

FetchAsNeeded como verdadeira (.T.), as linhas são carregadas somente quando necessário. Quando a propriedade FetchAsNeeded é definida como verdadeira, não é possível realizar uma atualização até completar o carregamento, chamar a função SQLCANCEL( ) no identificador de conexão atual ou fechar a visualização.

Se desejar ver o impacto do uso da propriedade FetchAsNeeded, defina a propriedade FetchAsNeeded em uma visualização recuperando um grande conjunto de resultados como .T. e, em seguida, abra a janela de pesquisa na visualização e role para baixo. A barra de status é atualizada para exibir o número de linhas recuperadas ao se movimentar pela janela de pesquisa.

Controlando a carga do cursor

Se quiser carregar o cursor inteiro, poderá emitir o comando GOTO BOTTOM ou qualquer comando que exija acesso ao conjunto de dados inteiro.

Dica Embora você possa usar o comando GOTO BOTTOM para carregar o cursor inteiro, geralmente é mais eficiente construir uma visualização com parâmetros que carrega apenas uma única linha de cada vez e faz novas consultas à medida que o usuário altera os registros. Para obter maiores informações sobre como construir visualizações de alto desempenho, consulte o capítulo 8, Criando visualizações

Os programas não fornecem processamento em loop inativo. Para carregar os cursores de visualização utilizando a linguagem de programação, utilize os comandos GO *nNúmeroRegistro* ou GOTO BOTTOM. Para carregar cursores criados com a passagem SQL em modo assíncrono, chame a função assíncrona de passagem SQL uma vez para cada subconjunto de linha.

Cancelando uma instrução SQLEXEC( )

Você pode usar a função SQLCANCEL( ) para cancelar uma instrução SQLEXEC( ) ou uma visualização a qualquer momento. Porém, se o servidor tiver concluído a criação do conjunto de resultados remotos e o Visual FoxPro tiver começado a carregar o conjunto de resultados remotos no cursor local, a função SQLCANCEL( ) cancelará a instrução SQLEXEC( ) e sairá do cursor local. Se você deseja excluir o cursor local, emita o comando USE, que fechará o cursor e cancelará o carregamento.

O comando USE não cancelará uma instrução SQLEXEC( ) se a instrução ainda não tiver criado um cursor local. Para determinar se o Visual FoxPro criou um cursor local, você poderá chamar a função USED( ).

Controlando o tamanho da carga

Você controla o número de linhas carregadas de cada vez pelo aplicativo a partir de um servidor remoto, definindo a propriedade FetchSize na sua visualização. A propriedade FetchSize especifica quantos registros são carregados de uma vez no cursor local a partir do servidor remoto, através da carga progressiva ou das chamadas de passagem SQL assíncronas. O valor padrão é 100 linhas.

Para controlar o número de registros carregados de cada vez em uma visualização

- No **Criador de visualizações**, escolha **Opções avançadas** do menu **Consulta**. Na área **Carga de dados** da caixa de diálogo **Opções avançadas**, utilize o controle de rotação para definir um valor para o **Número de registros carregados a cada vez**.

  – Ou –

- Defina a propriedade FetchSize com a função DBSETPROP( ) para definir o tamanho da carga de definição da visualização.

  – Ou –

- Defina a propriedade FetchSize com a função CURSORSETPROP( ) para definir o tamanho de carga do cursor de visualização ativo.

Por exemplo, o código a seguir especifica a definição de visualização para carregar

progressivamente 50 linhas de cada vez em `Customer_remote_view`:

```
? DBSETPROP('Customer_remote_view', 'View', 'FetchSize', 50)
```

Utilizando carga de Memo com atraso

Um aplicativo bem projetado geralmente utiliza a carga de Memo com atraso para acelerar a transferência de conjuntos de resultados que contêm campos Memo ou Geral. A carga de Memo com atraso significa que o conteúdo dos campos Memo e Geral não são transferidos automaticamente quando você transfere um conjunto de resultados. Em vez disso, o restante dos campos na linha é rapidamente transferido e o conteúdo dos campos Memo e Geral só é carregado quando você os chama, abrindo o campo Memo ou Geral. A carga de Memo com atraso fornece a transferência mais rápida de linhas e permite que o conteúdo dos campos Memo ou Geral, que pode ser bem extenso, seja carregado apenas se o usuário tiver necessidade.

Por exemplo, o seu formulário pode incluir um campo Geral que exibe uma figura. Para acelerar o desempenho, você pode usar a carga de Memo com atraso para evitar a transferência da figura até que o usuário escolha um botão **Visualizar** para o seu formulário. O código por trás do botão **Visualizar** depois carrega o campo Geral e o exibe no formulário.

Para controlar a carga de Memo com atraso, você usa a propriedade FetchMemo na sua visualização ou cursor. A propriedade FetchMemo especifica se o conteúdo dos campos Memo ou Geral devem ser carregados quando a linha é transferida. O valor padrão é verdadeiro (.T.), o que significa que os campos Memo e Geral são transferidos automaticamente. Se os dados contiverem grande quantidade de dados do campo Memo ou Geral, você verificará uma melhoria de desempenho quando definir a propriedade FetchMemo como falsa (.F.).

Observação A visualização deve ser atualizável para permitir que a carga de Memo com atraso funcione, porque o Visual FoxPro usa os valores de campos-chave estabelecidos pelas propriedades de atualização para localizar a linha fonte no servidor quando recuperar o campo Memo ou Geral. Para obter informações sobre como tornar uma visualização atualizável, consulte o capítulo 8, Criando visualizações

Utilize DBSETPROP( ) para definir a propriedade FetchMemo em uma visualização e CURSORSETPROP( ) para definir a propriedade FetchMemo em um cursor.

Otimizando o desempenho do carregamento de dados

Você pode usar as recomendações a seguir para definir propriedades de conexão e visualização para otimizar o carregamento de dados. A propriedade PacketSize em sua conexão tem a maior influência no desempenho. Além disso, você pode otimizar o desempenho do carregamento utilizando conexões síncronas.

Objeto	Propriedade	Definição
Conexão	PacketSize	4K to 12K[1]
Conexão	Asynchronous[2]	.F.
Visualização	FetchSize[3]	máximo

[1] Defina um valor maior para as linhas que contêm mais dados; experimente localizar o melhor valor.

[2] Utilize conexões síncronas para melhorar o desempenho em até 50%, a não ser que queira ser capaz de cancelar instruções SQL ao executar no servidor.

[3] O efeito de FetchSize depende muito do tamanho do registro do conjunto de resultados carregado. No modo síncrono, isso não afeta de maneira significativa o desempenho, então defina-o como necessário para o processamento assíncrono de passagem SQL exiba o carregamento progressivo. FetchSize, se reduzida, fornece uma velocidade de resposta bem melhor enquanto carrega progressivamente, mas diminui a velocidade de carregamento. Se aumentado, ela aumentará o desempenho de carregamento de visualizações.

O desempenho real depende em grande parte da configuração do sistema e dos requisitos do aplicativo. Essas recomendações estão baseadas na máquina de um usuário que está executando o

Windows NT versão 3,5, com a versão ODBC 2,10 e uma versão de driver ODBC de Servidor SQL versão 2,05 e como Servidor uma máquina que está executando o Windows NT, versão 3,5 com um servidor de SQL versão 4,21e 6,0.

Acelerando consultas e visualizações

Você pode melhorar o desempenho de consultas e visualizações adicionando índices, otimizando o processamento local e remoto e otimizando expressões de parâmetros.

Adicionando índices a tabelas remotas

Os índices remotos podem aumentar de maneira significativa a velocidade das consultas. As consultas de várias tabelas são mais rápidas se as tabelas estiverem indexadas nos campos de associação. Possuir índices em campos incluídos na cláusula WHERE de uma consulta também pode melhorar o desempenho.

Os índices agrupados fornecem o melhor desempenho. No Servidor SQL, cada tabela pode ter um índice agrupado. O **Assistente de upsizing do SQL Server** cria automaticamente índices agrupados em tabelas que tinham uma chave primária no Visual FoxPro.

Dica Embora os índices dos campos de uma tabela usados em consultas possam acelerar o processamento, índices em conjuntos de resultados podem diminuir o desempenho. Cuidado ao usar índices em conjuntos de resultados.

Otimizando o processamento local e remoto

Se precisar processar uma combinação de dados locais e remotos, crie uma visualização remota que combine todos os dados remotos em uma única visualização. Depois, você pode associar a visualização remota com os dados locais em uma visualização local. Pelo fato de o Visual FoxPro carregar as visualizações completamente antes de associar e filtrar a visualização combinada, é importante limitar o tamanho do conjunto de resultado da visualização.

Para otimizar o processamento remoto, limite o conjunto de resultados remotos para a quantidade mínima de dados necessários para o seu aplicativo. Quando você recupera menos dados em um conjunto de resultados remotos, você minimiza o tempo exigido para transferir dados remotos para o cursor de visualização ou consulta local.

Otimizando visualizações com parâmetros

Você pode acelerar a recuperação de dados durante operações REQUERY( ) em uma visualização com parâmetros aberta, compilando a visualização antes de executá-la. Para pré-compilar ou “preparar” uma visualização, defina a propriedade Prepared na visualização como verdadeira (.T.).

Otimizando as expressões de parâmetros

Os parâmetros de visualização e passagem SQL são expressões do Visual FoxPro e são avaliados nele antes de serem enviados ao servidor remoto. O tempo de avaliação para a expressão é importante, porque aumenta o tempo de execução da consulta.

Acelerando formulários

Ao criar um formulário principalmente com base em dados do servidor, utilize uma abordagem minimalista para obter melhor desempenho. Determine os dados e funcionalidades necessários e faça um intervalo antes de pedir ao servidor estes dados e funcionalidades até que seja solicitado pelo usuário. Solicitar dados do servidor usa tempo de processamento e cria tráfego na rede. Para solicitar menos dados nos formulários:

- Peça o menor número possível de registros. Por exemplo, utilize um filtro ou consulta para limitar o tamanho do conjunto de registros. Certifique-se de que o servidor pode processar as restrições

utilizadas.

- Utilize o menor número possível de campos remotos em visualizações subjacentes aos seus formulários.
- Utilize o menor número possível de formulários que acessam visualizações remotas no conjunto de formulários. Quando você abrir um conjunto de formulários, todos os formulários no conjunto serão abertos e preenchidos com dados, conforme aplicável. Limitando o número de formulários no conjunto de formulários, especialmente os que devem se conectar a um servidor e recuperar dados remotos, você diminui o tempo que o conjunto de formulários leva para carregar.
- Utilize menos controles de ligação que acessam dados remotos. Cada caixa de combinação, caixa de listagem e grade que está ligada a uma tabela ou consulta remota requer uma consulta ao servidor separada quando o formulário é aberto. Evite controles que contenham totais ou caixas de listagem e de combinação que tenham grandes fontes de linhas.
- Se os usuários precisarem comparar vários conjuntos de dados, considere o armazenamento dos dados fornecidos pelo servidor em tabelas locais temporárias. Forneça um formulário no qual o usuário possa usar os dados armazenados anteriormente ou execute uma nova consulta.

Armazenando tabelas de pesquisa localmente

Geralmente, um aplicativo contém vários formulários que utilizam a mesma tabela remota. Se os dados na tabela não mudarem com freqüência, você poderá acelerar a carga de formulários e reduzir a carga do servidor utilizando uma das técnicas a seguir:

- Armazene tabelas que nunca mudam e que não sejam muito grandes (como os nomes e abreviações das regiões ou estados no seu país) no banco de dados do aplicativo do Visual FoxPro local. Se a tabela for associada em visualizações ou consultas com tabelas remotas, você também deve manter uma cópia da tabela no servidor para evitar associar dados remotos e locais.
- Armazene tabelas que mudam pouco (como uma lista dos prédios da empresa) no servidor e no banco de dados do aplicativo local. Proporcione uma opção para que o usuário transfira a tabela quando os dados forem efetivamente alterados.
- Armazene tabelas que mudam de vez em quando mas não diariamente (como uma lista de funcionários em uma pequena empresa ou departamento) tanto no servidor quanto no banco de dados do aplicativo local. O aplicativo deverá atualizar automaticamente a versão local sempre que for iniciado. Este método utiliza tempo extra quando o aplicativo é iniciado, mas acelera consultas durante o processamento do aplicativo.

Exibindo campos somente quando solicitado

Exiba campos que demoram para recuperar dados do servidor, como os campos Memo ou Geral, apenas quando solicitado. Você pode usar as seguintes técnicas:

- Se o formulário estiver baseado em uma visualização, coloque os campos Memo ou Geral fora da tela em outra página de formulário. Adicione um rótulo ao formulário, como “Percorra as páginas para ver notas e figuras”, que informe ao usuário como exibir as informações. Defina a propriedade FetchMemo na visualização ou cursor como falsa (.F.), de modo que o Visual FoxPro não recupere os campos Memo ou Geral até que sejam exibidos na tela.
- Defina a propriedade Visible como falsa (.F.) para controles ligados aos campos Memo ou Geral. Adicione um botão de alternância ou um botão de comando que defina a propriedade como verdadeira (.T.), de modo que o usuário possa optar por visualizar o conteúdo destes controles.
- Exiba os campos mais importantes em um formulário principal e forneça um botão rotulado “Mais informações” que abra outro formulário contendo outros campos. Baseie o segundo formulário em uma visualização com parâmetros pelo campo de chave primária no formulário principal. Por exemplo, vamos supor que você tenha um formulário principal baseado em uma visualização cuja instrução SQL SELECT inclua o seguinte código:

```
SELECT customer_id, company_name, address, city, region, country
FROM customers
```

No formulário anterior, `cust_id` está ligado a `thisform.txtCust_id`. Você pode basear o segundo formulário na visualização seguinte, que só é usada quando o usuário escolhe o botão **Mais informações**:

```
SELECT orders.order_id, orders.order_date, orders.shipper_id, ;
   employee.emp_id, employee.last_name, employee.first_name ;
FROM orders, employee ;
WHERE orders.cust_id = ?THISFORM.txtCust_id ;
AND orders.employee_id = employees.emp_id
```

Melhorando o desempenho em atualizações e exclusões

Você pode acelerar as instruções de Atualização e Exclusão das seguintes formas:

- Adicionando marcas de data e hora às tabelas remotas
- Utilizando a propriedade CompareMemo.
- Utilizando o modo de transação manual.
- Utilizando procedimentos armazenados no servidor
- Atualizações em lote

Adicionando marcas de data e hora

Você pode melhorar o desempenho quando atualizar, inserir ou excluir dados em uma tabela remota que contenha muitos campos, adicionando um campo de marca e data e hora à tabela remota, se o servidor fornecer o tipo de campo Timestamp.

A presença de um campo de marca de data e hora em uma tabela remota permite que você utilize a opção de atualização SQL WhereType DB_KEYANDTIMESTAMP do Visual FoxPro. Esta opção economiza tempo de processamento porque o Visual FoxPro compara apenas dois campos na sua visualização, o campo chave e o campo de marca de data e hora, com uma tabela remota para detectar conflitos de atualização. Ao comparar apenas dois campos, em vez de todos os campos atualizáveis (com a opção DB_KEYANDUPDATABLE) ou todos os campos modificados (com a opção DB_KEYANDMODIFIED), a opção DB_KEYANDTIMESTAMP reduz o tempo necessário para atualizar dados remotos. Para obter maiores informações sobre as opções WhereType, consulte o capítulo 8, Criando visualizações.

Observação A opção DB_KEYANDTIMESTAMP compara os campos- chave e marca de data e hora somente quando a tabela remota contiver um campo de marca de data e hora. Se você utilizar a opção DB_KEYANDTIMESTAMP com uma tabela remota que não contém um campo de marca de data e hora, o Visual FoxPro irá comparar somente os campos-chave.

O **Assistente de upsizing** poderá adicionar automaticamente campos de marca de data e hora adequados às tabelas exportadas. Para obter maiores informações, consulte “Colunas de identidade de marca de data e hora” no capítulo 20, Migrando bancos de dados do Visual FoxPro

Dica Se você fizer alguma coisa que altere a estrutura da tabela base de uma visualização, como adicionar um campo de marca de data e hora, talvez seja necessário recriar a visualização. Os campos em uma definição de visualização são armazenados no banco de dados e qualquer alteração nas tabelas base para uma visualização depois que a visualização tiver sido usada não será refletida na definição de visualização até que você recrie a visualização.

Excluindo campos Memo da cláusula WHERE de atualização

Sempre que for adequado, você poderá acelerar as atualizações evitando que os campos Memo da visualização (campos do tipo Memo, Geral ou Figura) sejam comparados com os seus correspondentes na tabela base. Como padrão, a propriedade CompareMemo é definida como verdadeira (.T.), o que inclui automaticamente campos Memo na cláusula SQL WHERE gerada ao criar uma visualização atualizável. Você pode definir a propriedade CompareMemo como falsa (.F.) para excluir memos da cláusula SQL WHERE.

Utilizando transações

Para obter o melhor desempenho possível, utilize o modo de transação manual e gerencie as transações por conta própria. O modo de transação manual permite que você controle a gravação de um grupo de transações, o que permite que o servidor processe mais instruções rapidamente.

O modo de transação automático é mais demorado, porque, como padrão, cada instrução de atualização única é quebrada em uma transação separada. Este método fornece controle máximo sobre cada instrução de atualização individual, mas também aumenta o volume.

Você pode melhorar o desempenho no modo de transação automático, aumentando a definição da propriedade BatchUpdateCount na visualização ou cursor. Quando você utiliza uma definição BatchUpdateCount extensa, muitas instruções de atualização são processadas em lote em uma instrução de atualização única, que então é quebrada em uma transação única. No entanto, se qualquer instrução no processamento em lote falhar, todo o lote será desfeito.

Dica A propriedade BatchUpdateCount não é suportada por alguns servidores; você deve testar essa propriedade contra cada servidor remoto antes de distribuí-la no aplicativo.

Utilizando procedimentos armazenados no servidor

Você pode criar procedimentos armazenados no servidor, que são pré-compilados e, portanto, executados muito rapidamente. Você pode executar procedimentos armazenados, enviar parâmetros com passagem SQL e mover processamento adicional para o servidor, conforme apropriado para o seu aplicativo.

Por exemplo, talvez você queira obter dados do usuário localmente e depois executar uma passagem SQL para enviar os dados ao servidor, chamando o procedimento armazenado adequado. Para isso, talvez você queira criar um formulário em um cursor ou matriz local para coletar dados, depois escrever código que construa uma instrução SQLEXEC( ), utilizando o nome do procedimento armazenado no servidor e os parâmetros a serem fornecidos. Você pode então adicionar este código ao evento Click de um botão de comando denominado **OK** ou **Gravar**. Quando o usuário escolhe o botão, a instrução SQLEXEC( ) é executada. Usar os procedimentos armazenados no servidor para atualizar dados remotos pode ser mais eficiente, porque os procedimentos armazenados são compilados no servidor.

Atualizações em lote

Se o aplicativo atualizar uma série de registros, você talvez queira atualizar em lote para que possam ser manipulados de forma mais eficiente pela rede e pelo servidor. As instruções de Atualização ou Inserção são processadas em lote antes de serem enviadas ao servidor, de acordo com a definição da propriedade BatchUpdateCount da visualização. O valor padrão é 1, que significa que cada registro é enviado ao servidor com uma instrução de atualização. Você pode reduzir o tráfego na rede, aumentando o valor para compactar atualizações múltiplas em uma instrução.

Dica A propriedade BatchUpdateCount não é suportada por alguns servidores; você deve testar essa propriedade com cada servidor remoto antes de distribuí-la no aplicativo.

Para usar este recurso de forma eficiente, a conexão da visualização deve ser definida para o modo de Utilização de Buffer 5, para utilização de buffer de tabela otimista. O ideal é que as alterações estejam restritas aos mesmos campos em cada linha do cursor. Você pode usar DBSETPROP( ) para definir a propriedade BatchUpdateCount para a definição da visualização; para alterar o valor para um cursor de visualização ativo, utilize CURSORSETPROP( ).

Otimizando o desempenho de atualizações e exclusões

Você pode utilizar as seguintes diretrizes para definir as propriedades de visualização e conexão e para otimizar o desempenho de atualizações e exclusões. A propriedade BatchSize na visualização tem a maior influência sobre o desempenho.

Objeto	Propriedade	Definição	Observações
Visualização	BatchUpdateCount	10 – 30 linhas	Define um valor maior para atualizações menores.[1] Defina para melhorar o desempenho em até 50%. O padrão é 1.
Conexão	Assíncrona	(.F.)	Utilize conexões síncronas para melhorar o desempenho em até 50%, a não ser que você deseje ser capaz de cancelar instruções SQL ao executar no servidor. O padrão é síncrono.
Conexão	WaitTime	N/A	Para melhorar o desempenho no modo assíncrono, utilize um tempo de espera menor; para reduzir o tráfico de rede, aumente o tempo de espera.
Conexão	PacketSize	4K a 12K	Tem pouco efeito no desempenho.

[1] O melhor valor também depende da velocidade do servidor.

O desempenho real depende em grande parte da configuração do sistema e dos requisitos do aplicativo. Experimente com os valores listados determinar as melhores definições para a configuração. As recomendações anteriores eram baseadas adequadamente na máquina de um cliente que executa o Windows NT versão 3.5 com ODBC 2.10 e SQL Server Driver de 2.05; e uma máquina servidora executando o Windows NT, Versão 3.5 com Microsoft SQL Server 4.21 e 6.0.