

Se você criar um aplicativo que será executado em várias máquinas em um ambiente de rede ou se várias instâncias de um formulário acessarem os mesmos dados, será necessária uma programação para acesso compartilhado. Acesso compartilhado significa fornecer formas eficientes de utilizar e compartilhar dados entre usuários, bem como restringir o acesso quando for necessário.

O Visual FoxPro fornece suporte ao acesso compartilhado ou exclusivo a dados, opções de bloqueio, sessões de dados, utilização de buffer de registro e de tabela e transações. Embora esses recursos sejam particularmente úteis em ambientes compartilhados, você poderá também utilizá-los em ambientes monousuário.

Este capítulo aborda os tópicos a seguir:

- [Controlando o acesso a dados](#)
- [Atualizando dados](#)
- [Gerenciando conflitos](#)

Controlando o acesso a dados

Como você acessa dados em arquivos, o gerenciamento eficaz de dados começa com o controle sobre o ambiente desses arquivos. Você deve escolher como acessar os dados e como e quando limitar esse acesso.

Acessando dados

Em um ambiente compartilhado, você pode acessar os dados de duas maneiras: a partir de arquivos exclusivos ou de arquivos compartilhados. Se você abrir uma tabela para acesso compartilhado, outros usuários também terão acesso ao arquivo. Se abrir para acesso exclusivo, nenhum outro usuário poderá ler ou gravar esse arquivo. Como a utilização exclusiva invalida muitas das vantagens de compartilhamento de dados em uma rede, ele deve ser utilizado com moderação.

Utilizando tabelas com acesso exclusivo

A maneira mais restritiva de abrir um arquivo é a abertura exclusiva. Como padrão, ao abrir uma tabela através da interface, essa tabela será aberta para utilização exclusiva. Você pode também abrir explicitamente uma tabela para utilização exclusiva, utilizando os comandos do Visual FoxPro.

► Para abrir uma tabela para utilização exclusiva

- Digite os comandos a seguir na janela **Comando**:

```
SET EXCLUSIVE ON  
USE cMyTable  
– Ou –
```

- Digite o comando a seguir na janela **Comando**:

```
USE cMyTable EXCLUSIVE
```

Os comandos a seguir exigem que você abra uma tabela para utilização exclusiva:

- [ALTER TABLE](#)
- [INDEX](#) ao criar, adicionar ou excluir uma marca de índice composto.
- [INSERT \[BLANK\]](#)
- [MODIFY STRUCTURE](#)

Para utilizar este comando a fim de alterar a estrutura de uma tabela, você deve abrir a tabela de forma exclusiva. Contudo, pode-se utilizar este comando no modo somente para leitura quando abrir a tabela para utilização compartilhada.

- [PACK](#)
- [REINDEX](#)
- [ZAP](#)

O Visual FoxPro retornará a mensagem de erro “Abertura exclusiva de arquivo obrigatória” se você tentar executar um desses comandos em uma tabela compartilhada.

Você pode restringir o acesso a uma tabela, utilizando a função [FLOCK\(\)](#). Se utilizar [FLOCK\(\)](#) para bloquear a tabela, outros usuários não poderão fazer gravações nela, mas poderão usá-la para leitura.

Utilizando tabelas com acesso compartilhado

Quando você abrir uma tabela para utilização compartilhada, mais de uma estação de trabalho poderá utilizar a mesma tabela simultaneamente. Ao abrir uma tabela através da interface, você poderá substituir a definição padrão ativada (ON) de [SET EXCLUSIVE](#). Você pode abrir explicitamente uma tabela para utilização compartilhada, utilizando os comandos do Visual FoxPro.

► Para abrir uma tabela para utilização compartilhada

- Digite os comandos a seguir na janela **Comando**:

```
SET EXCLUSIVE OFF  
USE cMyTable  
– Ou –
```

- Digite o comando na janela **Comando**:

```
USE cMyTable SHARED
```

Ao adicionar ou alterar dados em uma tabela compartilhada, você deve primeiro bloquear o registro afetado ou a tabela inteira. Você pode bloquear um registro ou uma tabela aberta para utilização compartilhada das maneiras a seguir:

- Utilize um comando que executa um bloqueio automático de registro ou de tabela. Consulte a tabela de comandos que executam o bloqueio automático na seção [Escolhendo o bloqueio automático ou manual](#).
- Bloqueie manualmente um ou mais registros ou uma tabela inteira com as funções de bloqueio de registro e de tabela.
- Inicie a utilização de buffer com a função [CURSORSETPROP\(\)](#).

Arquivos de memo e de índice associados são sempre abertos com o mesmo status de compartilhamento que suas tabelas.

Se seu aplicativo utilizar uma tabela somente para consultas e todos os usuários do aplicativo a acessarem, você poderá melhorar o desempenho, marcando a tabela como somente para leitura.

Bloqueando dados

Se compartilhar o acesso a arquivos, você poderá também gerenciar o acesso a dados, bloqueando tabelas e registros. Os bloqueios, ao contrário das permissões de acesso, podem fornecer controle de dados a longo e a curto prazo. O Visual FoxPro fornece bloqueio automático e manual.

Escolhendo bloqueios de registro ou de tabela

O bloqueio de registro, automático ou manual, impede que um usuário grave um registro que esteja sendo gravado no momento por outro usuário. O bloqueio de tabela impede que outros usuários gravem na tabela inteira, mas permite sua leitura. Como o bloqueio da tabela proíbe que os usuários atualizem registros em uma tabela, ele deve ser utilizado com moderação.

Escolhendo o bloqueio automático ou manual

Além do bloqueio de registro ou de tabela, você pode também escolher se ele será automático ou manual. Vários comandos do Visual FoxPro tentam bloquear automaticamente um registro ou uma tabela antes que o comando seja executado. Se o registro ou a tabela for bloqueado com êxito, o comando será executado e o bloqueio liberado.

Comandos que bloqueiam automaticamente registros e tabelas

Comando	Escopo do bloqueio
ALTER TABLE	A tabela inteira
APPEND	O cabeçalho da tabela
APPEND BLANK	O cabeçalho da tabela
APPEND FROM	O cabeçalho da tabela
APPEND FROM ARRAY	O cabeçalho da tabela
APPEND MEMO	O registro atual
BLANK	O registro atual
BROWSE, CHANGE e EDIT	O registro atual e todos os registros dos campos com aliases em tabelas relacionadas quando a edição de um campo começa
CURSORSETPROP()	Depende dos parâmetros
DELETE	O registro atual
DELETE NEXT 1	O registro atual
DELETE RECORD <i>n</i>	O registro <i>n</i>
DELETE de mais de um registro	A tabela inteira
DELETE - SQL	O registro atual
GATHER	O registro atual
INSERT	A tabela inteira
INSERT - SQL	O cabeçalho da tabela
MODIFY MEMO	O registro atual quando a edição começa
READ	O registro atual e todos os registros dos campos com aliases
RECALL	O registro atual
RECALL NEXT 1	O registro atual
RECALL RECORD <i>n</i>	O registro <i>n</i>
RECALL de mais de um registro	A tabela inteira
REPLACE	O registro atual e todos os registros de campos com aliases
REPLACE NEXT 1	O registro atual e todos os registros de campos com aliases
REPLACE RECORD <i>n</i>	O registro <i>n</i> e todos os registros de campos com aliases
REPLACE de mais de um registro	A tabela inteira e todos os arquivos de campos com aliases
SHOW GETS	O registro atual e todos os registros referenciados por campos com aliases
TABLEUPDATE()	Depende do buffer
UPDATE	A tabela inteira
UPDATE - SQL	A tabela inteira

Características do bloqueio de registro

Os comandos que tentam bloquear um registro são menos poderosos do que os comandos que

bloqueiam tabelas. Quando você bloqueia um registro, outros usuários ainda podem adicionar ou excluir outros registros. Se um registro ou tabela já estiver bloqueado por outro usuário, uma tentativa de bloquear um registro ou tabela falhará. Os comandos que tentarem bloquear o registro atual retornarão a mensagem de erro "O registro está sendo utilizado por outro usuário", se o registro não puder ser bloqueado.

Os comandos **BROWSE**, **CHANGE**, **EDIT** e **MODIFY MEMO** não bloqueiam um registro até que você o edite. Se você estiver editando campos a partir de registros em tabelas relacionadas, os registros relacionados serão bloqueados, se possível. A tentativa de bloqueio falhará se o registro atual ou qualquer um dos registros relacionados também for bloqueado por outro usuário. Se a tentativa de bloqueio tiver êxito, será possível editar o registro; o bloqueio será liberado quando você se movimentar para um outro registro ou ativar outra janela.

Características do bloqueio de cabeçalho e de tabela

Alguns comandos do FoxPro bloqueiam uma tabela inteira enquanto outros bloqueiam somente um cabeçalho da tabela. Os comandos que bloqueiam toda a tabela são mais intrusivos do que os comandos que bloqueiam somente os cabeçalhos das tabelas. Quando você bloqueia o cabeçalho da tabela, outros usuários não conseguem adicionar registros, mas ainda podem alterar os dados nos campos.

Os usuários podem compartilhar a tabela sem causar conflitos quando você emite o comando **APPEND BLANK**, porém pode ocorrer um erro enquanto um outro usuário também estiver incluindo um registro BLANK na tabela. Você pode interceptar o erro "O arquivo está sendo utilizado por outro usuário", que é retornado quando dois ou mais usuários executam **APPEND BLANK** simultaneamente. Os comandos que bloqueiam uma tabela inteira retornarão a mensagem de erro "O arquivo está sendo utilizado por outro usuário", se a mesma não puder ser bloqueada. Para cancelar a tentativa de bloqueio, pressione ESC.

Exemplo: bloqueio automático

No exemplo a seguir, o usuário bloqueia automaticamente o cabeçalho de tabela `customer` incluindo os registros de outra tabela, mesmo se `customer` tiver sido aberta como um arquivo compartilhado.

```
SET EXCLUSIVE OFF
USE customer
APPEND FROM oldcust FOR status = "OPEN"
```

Bloqueando manualmente

Você pode bloquear manualmente um registro ou uma tabela com as funções de bloqueio.

► Para bloquear manualmente um registro ou tabela

- Utilize um dos comandos a seguir:

```
RLOCK()
LOCK()
FLOCK()
```

RLOCK() e **LOCK()** são idênticos e bloqueiam um ou mais registros. **FLOCK()** bloqueia um arquivo. As funções **LOCK()** e **RLOCK()** podem ser aplicadas a um cabeçalho de tabela. Se você fornecer 0 como o registro para **LOCK()** ou **RLOCK()** e o teste indicar que o cabeçalho está desbloqueado, a função bloqueará o cabeçalho e retornará verdadeiro (.T.).

Depois de bloquear um registro ou tabela, certifique-se de liberar o bloqueio utilizando o comando **UNLOCK** o mais rápido possível para fornecer acesso a outros usuários.

Essas funções de bloqueio manual bloqueiam as ações a seguir:

- Testam o status de bloqueio do registro ou tabela.
- Bloqueiam o registro ou a tabela e retornam verdadeiro (.T.), se o teste indicar que o registro está desbloqueado.

- Tentam o bloqueio da tabela ou registro novamente, dependendo da definição atual de **SET REPROCESS**, se o registro ou tabela não puderem ser bloqueados.
- Retornam verdadeiro (.T.) ou falso (.F.), indicando se a tentativa de bloqueio teve êxito.

Dica Se você quiser testar o status de bloqueio de um registro sem bloqueá-lo, utilize a função **ISRLOCKED()** ou **ISFLOCKED()**.

Se a tentativa de bloqueio de um registro ou tabela falhar, o comando **SET REPROCESS** e a rotina de erro atual determinarão se o bloqueio será tentado novamente. **SET REPROCESS** afeta o resultado de uma tentativa de bloqueio sem êxito. Você pode controlar o número de tentativas de bloqueio ou o intervalo de tempo em que é feita uma tentativa de bloqueio com o **SET REPROCESS**.

Exemplo: bloqueio manual

O exemplo a seguir abre a tabela `customer` para acesso compartilhado e utiliza **FLOCK()** para tentar bloqueá-la. Se a tabela for bloqueada com sucesso, **REPLACE ALL** atualizará todos os registros na tabela. **UNLOCK** libera o bloqueio do arquivo. Se o arquivo não puder ser bloqueado porque outro usuário já bloqueou o arquivo ou um registro dentro dele, será exibida uma mensagem.

```
SET EXCLUSIVE OFF
SET REPROCESS TO 0
USE customer    && Abre tabela de forma compartilhada
IF FLOCK( )
    REPLACE ALL contact ;    && Substitui e desbloqueia
    WITH UPPER(contact)
    UNLOCK
ELSE    && Exibe mensagem
    WAIT " Arquivo em utilização por outro usuário." WINDOW NOWAIT
ENDIF
```

Desbloqueando dados

Após estabelecer um bloqueio de registro ou de arquivo e completar uma operação de dados em ambiente compartilhado, você deve liberar o bloqueio o mais rápido possível. Existem muitas maneiras de se fazer isso. Em alguns casos, a simples movimentação para o próximo registro é suficiente para desbloquear os dados. Outras situações exigem comandos explícitos.

Para desbloquear um registro automaticamente bloqueado, você precisa somente movimentar o ponteiro do registro, mesmo se tiver definido **MULTILOCKS** como ativado (ON). Você deve, entretanto, remover explicitamente o bloqueio manual de um registro; a simples movimentação do ponteiro do registro não será suficiente.

A tabela a seguir descreve os efeitos dos comandos em bloqueios manuais e automáticos de registros e tabelas.

Comando	Efeito
UNLOCK	Libera o bloqueio de registro e arquivo na área de trabalho atual.
UNLOCK ALL	Libera todos os bloqueios em todas as áreas de trabalho na sessão atual.
SET MULTILOCKS OFF	Ativa a liberação automática do bloqueio atual quando um novo bloqueio é assegurado.
FLOCK()	Libera todos os bloqueios de registros no arquivo afetado antes de bloquear o arquivo.
CLEAR ALL, CLOSE ALL, USE, QUIT	Libera todos os bloqueios de registros e arquivos.
END TRANSACTION	Libera os bloqueios automaticamente.
TABLEUPDATE()	Libera todos os bloqueios antes de atualizar

a tabela.

Importante Se um registro foi automaticamente bloqueado em uma função definida pelo usuário e você mover o ponteiro para fora e para dentro do registro, o bloqueio será liberado. Utilize o buffer de tabela para evitar esse problema.

Utilizando sessões de dados

Para se certificar de que cada usuário em um ambiente compartilhado possua uma duplicata segura e exata do ambiente de trabalho e que várias instâncias de um formulário possam operar de modo independente, o Visual FoxPro oferece as sessões de dados.

Uma sessão de dados é uma representação do ambiente de trabalho dinâmico atual. Você pode considerar uma sessão de dados como um ambiente de dados em miniatura que executa uma sessão aberta do Visual FoxPro em uma máquina. Cada sessão de dados contém:

- Uma cópia dos itens no ambiente de dados do formulário
- Cursores representando as tabelas abertas, seus índices e relacionamentos.

O conceito de uma sessão de dados é facilmente compreendido ao considerar o que acontece quando um mesmo formulário é aberto simultaneamente a partir de estações de trabalho diferentes em um aplicativo multiusuário. Nesse caso, cada estação de trabalho executa uma sessão separada do Visual FoxPro e, logo, possui o seu próprio conjunto de áreas de trabalho, cursores representando tabelas base abertas, índices e relacionamentos.

No entanto, se você abrir várias instâncias do mesmo formulário em um único projeto, em uma máquina, na mesma sessão do Visual FoxPro, os formulários compartilharão a Sessão de dados padrão, representando um único ambiente de trabalho dinâmico. Cada instância do formulário aberto aberta na mesma sessão do Visual FoxPro utiliza o mesmo conjunto de áreas de trabalho, e as ações em uma instância de um formulário que movem o ponteiro do registro em uma área de trabalho afetam automaticamente outras instâncias do mesmo formulário.

Utilizando sessões de dados privadas

Se você desejar ter maior controle sobre várias instâncias do formulário, poderá implementar Sessões de dados privadas. Quando o formulário utiliza sessões de dados privadas, o Visual FoxPro cria uma nova sessão de dados para cada instância do controle de formulário, conjunto de formulários ou barra de ferramentas criada pelo seu aplicativo. Cada sessão de dados privada contém:

- Uma cópia separada de cada tabela, índice e relacionamento no ambiente de dados do formulário.
- Um número ilimitado de áreas de trabalho.
- Ponteiros do registro para cada cópia de cada tabela que são independentes das tabela base do formulário.

O número de sessões de dados disponíveis é limitado somente pelo espaço em disco e memória do sistema disponível.

Você implementa sessões de dados privadas, definindo a propriedade `DataSession` para o formulário. A propriedade `DataSession` possui duas definições:

- 1 – Sessão de dados padrão (a definição padrão).
- 2 – Sessão de dados privada.

Como padrão, a propriedade `DataSession` de um formulário é definida como 1.

► Para ativar sessões de dados privadas

Escolha uma das opções a seguir:

- No **Criador de formulários**, defina a propriedade `DataSession` do formulário como **2 – Sessão**

de dados privada.

– Ou –

- No código, defina a propriedade `DataSession` como 2.

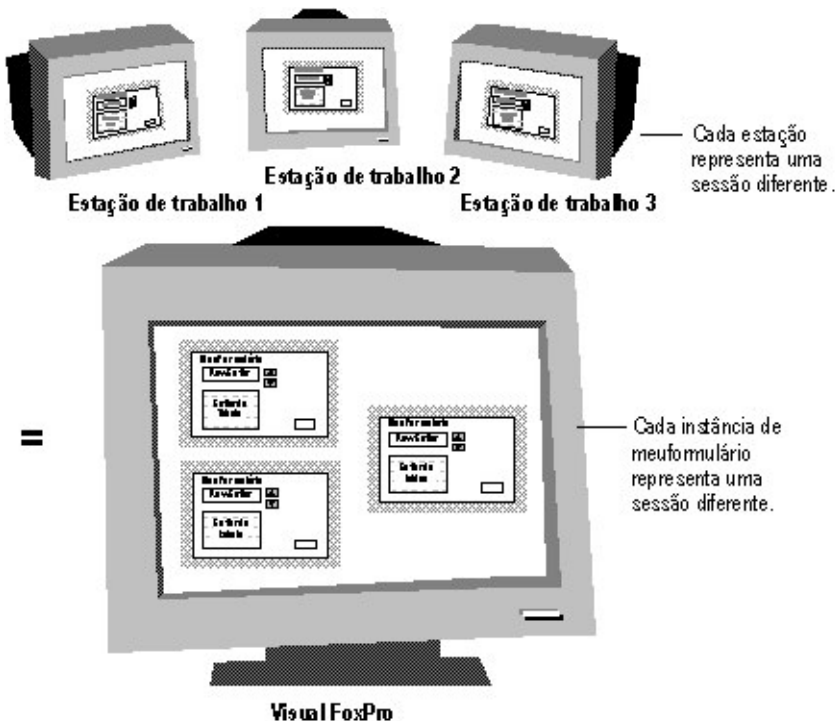
Por exemplo, digite:

```
frmFormName.DataSession = 2
```

Observação Você somente pode definir a propriedade `DataSession` no momento da criação. A propriedade `DataSession` fica definida como somente para leitura em tempo de execução.

Quando um formulário utiliza sessões de dados privadas, cada instância de um formulário aberta em uma única máquina, em uma única sessão do Visual FoxPro utiliza o seu próprio ambiente de dados. A utilização das sessões de dados privadas é semelhante à execução simultânea do mesmo formulário a partir de estações de trabalho diferentes.

Várias sessões de dados equivalentes



Identificando sessões de dados

Cada sessão de dados privada é identificada separadamente. Você pode ver o conteúdo de cada sessão de dados na janela **Sessão de dados**. Além disso, você pode alterar a descrição da sessão de dados utilizando os comandos do código de evento Load.

Você pode visualizar o número de identificação de cada sessão de dados, utilizando a propriedade em tempo de execução `DataSessionID`. O exemplo a seguir exibe a propriedade `DataSessionID` de um formulário denominado `frmMyForm`.

```
DO FORM frmMyForm  
? frmMyForm.DataSessionID
```

Se você ativar o formulário utilizando a cláusula `NAME`, poderá utilizar o nome do formulário para acessar a propriedade `DataSessionID`, como no código a seguir:

```
DO FORM MyForm NAME one  
? one.DataSessionID
```

A propriedade `DataSessionID` é projetada somente para identificar uma determinada sessão de

dados. Evite alterar a propriedade DataSessionID de uma instância do formulário, pois os controles ligados a dados perdem suas fontes de dados quando você altera DataSessionID.

Atualizando dados utilizando várias instâncias de formulários

Enquanto as sessões de dados privadas geram áreas de trabalho separadas que contêm cópias separadas das tabelas abertas, índices e relacionamentos de um formulário, cada cópia do formulário faz referência aos mesmos arquivos de índice base e às mesmas tabelas base subjacentes. Quando um usuário atualiza um registro de uma instância de um formulário, a tabela base referenciada pelo formulário é atualizada. Você verá as alterações feitas a partir de outra instância do formulário ao navegar para o registro alterado.

Os bloqueios feitos em registros ou tabelas em uma sessão de dados privada são respeitados por outras sessões de dados privadas. Por exemplo, se o usuário da sessão de dados 1 fizer um bloqueio em um registro, o usuário na sessão de dados 2 não poderá bloquear o registro. Se o usuário na sessão 1 abrir uma tabela com exclusividade, o usuário na sessão de dados 2 não poderá abrir a tabela. Respeitando os bloqueios efetuados por outras sessões de dados, o Visual FoxPro protege a integridade das atualizações feitas nas tabelas base subjacentes.

Personalizando o ambiente de uma sessão de dados

Pelo fato das sessões de dados controlarem o escopo de alguns comandos SET, você poderá utilizar sessões de dados privadas para estabelecer definições personalizadas do comando SET em uma única sessão do Visual FoxPro.

Por exemplo, o comando **SET EXACT**, que controla as regras utilizadas ao comparar seqüências de caracteres de tamanhos diferentes, será dimensionado para a sessão de dados atual. A definição padrão para SET EXACT é desativada especificando que, para serem equivalentes, as expressões devem ser correspondentes, caractere a caractere, até o final da expressão no lado direito ser alcançado. É possível que você queira ativar procuras “imprecisas” ou equivalentes, deixando SET EXACT definida como desativada (OFF) para a sessão de dados padrão; no entanto, é possível que o aplicativo contenha um formulário específico que requeira correspondências exatas. Você pode definir a propriedade DataSession para o formulário exigindo correspondências exatas como 2, para ativar sessões de dados privadas e, em seguida, definir SET EXACT como ativada (ON) para esse formulário. Ao emitir um comando SET somente para o formulário utilizando sessões de dados privadas, as definições da sessão global do Visual FoxPro são preservadas e as definições da sessão personalizada são ativadas para um formulário específico.

Alterando a atribuição automática de sessão de dados privada

Quando as sessões de dados privadas de um formulário estiverem sendo utilizadas, as alterações feitas nos dados de um formulário não serão representadas automaticamente em outras instâncias do mesmo formulário. Se você desejar que todas as instâncias de um formulário acessem os mesmos dados e reflitam imediatamente as alterações aos dados comuns, poderá alterar a atribuição automática da sessão de dados.

► Para substituir a atribuição automática da sessão de dados

- Utilize um desses comandos:

```
SET DATASESSION TO 1
```

– Ou –

```
SET DATASESSION TO
```

Os dois comandos permitem que a sessão de dados padrão seja controlada pela janela **Comando** e pelo **Gerenciador de projetos**.

Utilizando o buffer para acessar dados

Se você deseja proteger os dados durante as atualizações, utilize os buffers. A utilização do buffer

de registro e de tabela do Visual FoxPro ajuda a preservar a atualização dos dados e as operações de manutenção dos dados em registros únicos ou múltiplos de dados em ambientes multiusuário. Os buffers podem testar, bloquear e liberar os registros ou tabelas automaticamente.

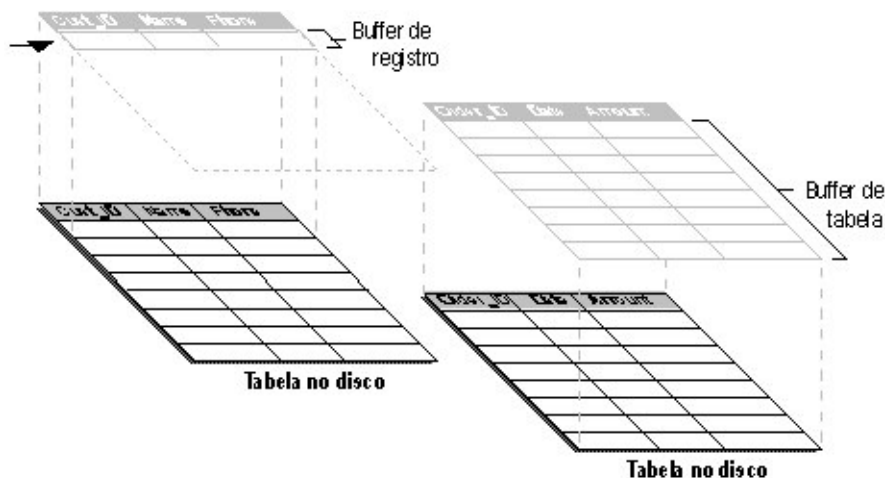
Com a utilização do buffer, você pode facilmente detectar e solucionar os conflitos nas operações de atualização de dados: o registro atual é copiado para uma localização na memória ou no disco gerenciada pelo Visual FoxPro. Ainda assim, outros usuários poderão acessar o registro original simultaneamente. Quando você se movimenta do registro ou tenta atualizá-lo através da linguagem de programação, o Visual FoxPro tenta bloquear o registro, verifica se nenhuma alteração foi feita por outros usuários e, em seguida, grava as edições. Após a tentativa de atualização dos dados, você deverá também solucionar os conflitos que impedem as edições de serem gravadas na tabela original.

Escolhendo um método de utilização de buffer

Antes de ativar o buffer, avalie o ambiente de dados para escolher o método de utilização de buffer e as opções de bloqueio que melhor se adequam às necessidades de edição do seu aplicativo, os tipos e tamanhos de registro e tabela, como as informações são utilizadas e atualizadas, entre outros fatores. Depois de ativar a utilização de buffer, ela permanecerá habilitada até que você desative o buffer ou feche a tabela.

O Visual FoxPro tem dois tipos de utilização de buffer: de registro e de tabela.

Utilização de buffer de registro e tabela do Visual FoxPro



- Para acessar, modificar e gravar um único registro ao mesmo tempo, escolha utilização de buffer de registro.

A utilização de buffer de registro fornece a validação de processo adequada com o mínimo de impacto nas operações de atualização de dados de outros usuários em um ambiente multiusuário.

- Para utilizar o buffer nas atualizações de diversos registros, escolha utilização de buffer de tabela.

A utilização de buffer de tabela oferece a maneira mais eficaz de gerenciar diversos registros em uma tabela ou registros filho em um relacionamento um-para-n.

- Para obter o máximo de proteção aos dados existentes, utilize as transações do Visual FoxPro.

Você pode utilizar as transações sozinhas, mas ganhará um rendimento adicional se utilizá-las como empacotadoras para os comandos de buffer de registros ou tabelas. Para obter maiores detalhes, consulte a seção, [Gerenciando atualizações com transações](#), posteriormente neste capítulo.

Escolhendo um modo de bloqueio

O Visual FoxPro fornece a utilização de buffer em dois modos de bloqueio: pessimista e otimista.

Essas opções determinam quando um ou mais registros serão bloqueados e como serão liberados.

Buffer pessimista

A utilização de buffer pessimista evita que outros usuários em um ambiente multiusuário acessem um determinado registro ou tabela enquanto você faz alterações nele. O bloqueio pessimista oferece o ambiente mais seguro para a alteração de registros individuais, mas pode tornar as operações do usuário mais lentas. Esse modo de buffer é o mais semelhante ao mecanismo de bloqueio padrão das versões anteriores do FoxPro, com o benefício adicional de um buffer de dados interno.

Buffer otimista

A utilização de buffer otimista é um modo eficaz de atualização de registros porque os bloqueios somente acontecem no momento em que o registro é gravado, diminuindo, desse modo, o tempo que um único usuário monopoliza o sistema em um ambiente multiusuário. Quando você utiliza um buffer de registro ou de tabela em visualizações, o Visual FoxPro impõe o bloqueio otimista.

O valor da propriedade Buffering, definido com a função [CURSORSETPROP\(\)](#), determina os métodos de utilização de buffer e de bloqueio.

A tabela a seguir resume os valores válidos para a propriedade Buffering.

Para ativar	Utilize este valor
A não utilização de buffer. O valor padrão.	1
Bloqueios pessimistas de registro que bloqueiam o registro agora e atualizam quando o ponteiro se move ou com TABLEUPDATE() .	2
Bloqueios otimistas de registro que esperam até que o ponteiro se mova para então bloquear e atualizar.	3
Bloqueios pessimistas de tabela que bloqueiam o registro agora e atualizam mais tarde com TABLEUPDATE() .	4
Bloqueios otimistas de tabela que esperam o TABLEUPDATE() para depois bloquearem e atualizarem os registros modificados.	5

O valor padrão para a propriedade Buffering é 1 para tabelas e 3 para visualizações. Se você utilizar o buffer para acessar dados remotos, a propriedade Buffering será 3, para o buffer de linha otimista, ou 5, para o buffer de tabela otimista. Para obter maiores informações sobre o acesso a dados em tabelas remotas, consulte o capítulo 6, [Consultando e atualizando várias tabelas](#), no *Guia do Usuário*.

Observação Defina MULTILOCKS como ativado (ON) em todos os modos de utilização de buffer acima de 1.

Ativando a utilização de buffer de registro

Ative a utilização de buffer de registro com a função [CURSORSETPROP\(\)](#).

► Para ativar o bloqueio pessimista de registro na área de trabalho atual

- Utilize esta função e valor:

```
CURSORSETPROP("Buffering", 2)
```

O Visual FoxPro tenta bloquear o registro na localização do ponteiro. Se o bloqueio tiver êxito, o Visual FoxPro colocará o registro em um buffer e permitirá a edição. Quando você movimentar o ponteiro do registro ou emitir um comando [TABLEUPDATE\(\)](#), o Visual FoxPro grava na tabela original o registro do buffer.

► Para ativar o bloqueio otimista de registro na área de trabalho atual

- Utilize esta função e valor:

`CURSORSETPROP("Buffering", 3)`

O Visual FoxPro grava o registro na localização do ponteiro em um buffer e permite as edições. Quando você movimenta o ponteiro do registro ou emite um comando `TABLEUPDATE()`, o Visual FoxPro tenta bloquear o registro. Se o bloqueio tiver êxito, o Visual FoxPro irá comparar o valor atual do registro no disco com o valor original no buffer. Se esses valores forem os mesmos, as edições serão gravadas na tabela original; se forem diferentes, o Visual FoxPro gerará um erro.

Ativando a utilização de buffer de tabela

Ative a utilização de buffer de tabela com a função `CURSORSETPROP()`.

► Para ativar o bloqueio pessimista de vários registros na área de trabalho atual

- Utilize esta função e valor:

`CURSORSETPROP("Buffering", 4)`

O Visual FoxPro tenta bloquear o registro na localização do ponteiro. Se o bloqueio tiver êxito, o Visual FoxPro colocará o registro em um buffer e permitirá a edição. Utilize o comando `TABLEUPDATE()` para gravar os registros do buffer na tabela original.

► Para ativar o bloqueio otimista de vários registros na área de trabalho atual

- Utilize esta função e valor:

`CURSORSETPROP("Buffering", 5)`

O Visual FoxPro grava os registros em um buffer e permite as edições até que você emita um comando `TABLEUPDATE()`. O Visual FoxPro executa, em seguida, a sequência a seguir em cada registro no buffer:

- Tenta um bloqueio em cada registro editado.
- Com um bloqueio bem-sucedido, compara o valor atual de cada registro no disco com o valor original no buffer.
- Grava as edições na tabela original se a comparação mostrar que os valores são idênticos.
- Gera um erro se os valores forem diferentes.

Quando a utilização de buffer de tabela for ativada, o Visual FoxPro tentará fazer as atualizações somente depois de um comando `TABLEUPDATE()`.

Incluindo e excluindo registros dos buffers de tabela

Você pode incluir e excluir registros enquanto a utilização de buffer de tabela estiver ativada: os registros incluídos serão adicionados ao final do buffer. Para acessar todos os registros no buffer, além dos registros incluídos, utilize a função `RECNO()`. A função `RECNO()` retorna números sequenciais negativos nos registros que você anexa a um buffer de tabela. Por exemplo, se você iniciar a utilização de buffer de tabela, editar os registros 7, 8 e 9 e, em seguida, incluir três registros, o buffer terá os valores `RECNO()` 7, 8, 9, -1, -2 e -3.

Buffer após a edição e inclusão de registros

BUFFER DE TABELA	
7	Editar Registro
8	Editar Registro
9	Editar Registro
-1	Incluir Registro
-2	Incluir Registro
-3	Incluir Registro

Os registros incluídos no buffer somente podem ser removidos com o comando `TBLEREVERT()`.

Para qualquer registro incluído, tanto o TABLEUPDATE() quanto o TABLEREVERT() excluirão o valor RECNO() negativo para o registro enquanto a seqüência for mantida.

Buffer após a edição, exclusão de um registro incluído e inclusão de outro

BUFFER DE TABELA	
7	Editar Registro
8	Editar Registro
9	Editar Registro
-1	Incluir Registro
-2	Excluir Registro
-3	Incluir Registro
-4	Incluir Registro

Quando utilizar um buffer de tabela, você poderá utilizar o comando GO com o valor RECNO() negativo para acessar um registro incluído específico. Com base no exemplo anterior, você pode digitar:

```
GO 7      && move para o primeiro registro no buffer
GO -3     && move para o sexto registro no buffer (o terceiro registro incluído)
```

► Para incluir registros em um buffer de tabela

- Utilize o comando **APPEND** ou **APPEND BLANK** depois de ativar a utilização de buffer de tabela. Os registros incluídos possuem números RECNO() negativos crescentes e seqüenciais.

► Para remover um registro incluído de um buffer de tabela

- 1 Utilize o comando **GO** com um valor negativo para posicionar o ponteiro no registro a ser excluído.
- 2 Utilize o comando **DELETE** para marcar o registro para exclusão.
- 3 Utilize a função **TABLEREVERT()** para remover o registro do buffer.

Observação A função TABLEREVERT() também afeta o status das linhas excluídas e alteradas.

► Para remover todos os registros incluídos de um buffer de tabela

- Utilize a função **TABLEREVERT()** com um valor verdadeiro (.T.).

TABLEREVERT() remove de um buffer de tabela os registros incluídos sem gravá-los na tabela.

TABLEUPDATE() grava todos os registros de buffer atuais em uma tabela, mesmo se tiverem sido marcados para exclusão.

Atualizando dados

Para atualizar dados, pode-se utilizar buffers, transações ou visualizações.

Fazendo atualizações com buffers

Depois de escolher o método de utilização de buffer e o tipo de bloqueio, você deverá ativar a utilização do buffer de registro ou de tabela.

► Para ativar a utilização do buffer

Escolha uma das opções a seguir:

- No **Criador de formulários**, defina a propriedade BufferModeOverride do cursor no ambiente de dados do formulário.

– Ou –

- No código, defina a propriedade Buffering.

Por exemplo, você pode ativar o buffer de linha pessimista inserindo o código a seguir no procedimento Init de um formulário.

```
CURSORSETPROP('Buffering', 2)
```

Em seguida, poderá colocar o código para as operações de atualização no código de método adequado para seus controles.

Para gravar as edições na tabela original, utilize **TABLEUPDATE()**. Para cancelar as edições depois de uma operação de atualização mal-sucedida em uma tabela restrita por regras, utilize **TABLEREVERT()**. **TABLEREVERT()** é válido mesmo se a utilização explícita de buffer de tabela não estiver ativada.

O exemplo a seguir demonstra como atualizar os registros quando a utilização de buffer de registro pessimista estiver ativada.

Exemplo de atualização com os buffers de registro e de tabela

Código	Comentário
<pre>OPEN DATABASE testdata USE customers CURSORSETPROP('Buffering', 2)</pre>	No código Init do formulário, abre a tabela e ativa a utilização de buffer de registro pessimista.
<pre>lModified = .F. FOR nFieldNum = 1 TO FCOUNT() IF GETFLDSTATE(nFieldNum) = 2 lModified = .T. EXIT ENDIF ENDIF ENDFOR</pre>	Verifica se algum campo foi modificado. Observação: Este código pode estar no evento Click de um botão de comando Salvar ou "Atualizar".
<pre>IF lModified nResult = MESSAGEBOX(("R Registro modificado. Salvar?", ; 4+32+256, "Alteração de Dados") IF nResult = 7 TABLEREVERT (.F.) ENDIF ENDIF</pre>	Localiza o próximo registro modificado. Apresenta o valor atual e fornece ao usuário a opção de reverter a modificação do campo atual.
<pre>SKIP IF EOF() MESSAGEBOX("Fim da tabela") SKIP -1 ENDIF THISFORM.Refresh</pre>	O SKIP garante que a última alteração foi gravada.

Gerenciando atualizações com transações

Mesmo com a utilização de buffer, algo pode sair errado. Se você deseja proteger as operações de atualização e recuperar uma sessão de código inteira como uma unidade, utilize as transações.

Ao adicionar as transações ao seu aplicativo, será oferecida uma proteção além da utilização de buffer de registro e de tabela do Visual FoxPro, colocando uma sessão de código inteira em uma unidade protegida e recuperável. Você poderá aninhar as transações e usá-las para proteger as atualizações nos buffers. As transações do Visual FoxPro estão disponíveis somente com as tabelas e visualizações contidas em um banco de dados.

Quebrando segmentos de código

Uma transação funciona como um empacotador que armazena as operações de atualização de dados na memória ou em disco, em vez de aplicar as atualizações diretamente no banco de dados. A verdadeira atualização do banco de dados é executada ao final da transação. Se por algum motivo o sistema não puder executar as operações de atualização no banco de dados, você poderá reverter

toda a transação e nenhuma operação será atualizada.

Observação As operações de atualização com buffers feitas fora de uma transação são ignoradas em uma transação na mesma sessão de dados.

Comandos que controlam transações

O Visual FoxPro oferece três comandos e uma função para gerenciar uma transação:

Para	Utilize
Iniciar uma transação	<code>BEGIN TRANSACTION</code>
Determinar o nível da transação atual	<code>TXNLEVEL()</code>
Inverter todas as alterações feitas desde a instrução <code>BEGIN TRANSACTION</code> mais recente	<code>ROLLBACK</code>
Bloquear registros, gravar no disco todas as alterações feitas nas tabelas do banco de dados desde a instrução <code>BEGIN TRANSACTION</code> mais recente e, em seguida, desbloquear os registros	<code>END TRANSACTION</code>

Você pode utilizar as transações para quebrar as modificações nas tabelas, nos arquivos .CDX estruturais e arquivos de memo associados às tabelas dentro do banco de dados. As operações que envolvem variáveis de memória e outros objetos não respeitam as transações; logo, você não poderá reverter ou confirmar tais operações.

Observação Ao utilizar dados armazenados em tabelas remotas, os comandos de transação somente controlam as atualizações feitas nos dados da cópia local do cursor de visualização, as atualizações feitas em tabelas base remotas não são afetadas. Para ativar as transações manuais em tabelas remotas, utilize `SQLSETPROP()` e, em seguida, controlar as transações com `SQLCOMMIT()` e `SQLROLLBACK()`.

Geralmente, você deve utilizar as transações com buffers de registro em vez de utilizar o buffer de tabela, exceto para quebrar as chamadas `TABLEUPDATE()`. Se você colocar um comando `TABLEUPDATE()` em uma transação, poderá reverter uma atualização mal-sucedida, identificar o motivo da falha e tentar novamente o `TABLEUPDATE()` sem perder os dados. Isso garante que a atualização acontecerá como uma operação “tudo ou nada”.

Apesar de o processamento de transação simples fornecer operações de atualização de dados seguras em situações normais, não oferece proteção total contra as falhas do sistema. Se ocorrer falta de energia ou algum outro tipo de interrupção do sistema durante o processamento do comando `END TRANSACTION`, a atualização dos dados poderá falhar.

Utilize o modelo de código a seguir para transações.

```
BEGIN TRANSACTION
* Atualiza registros
IF !Success = .F.    && ocorre um erro
    ROLLBACK
ELSE    && confirma as alterações
    * Valida os dados
    IF    && ocorre um erro
        ROLLBACK
    ELSE
        END TRANSACTION
    ENDIF
ENDIF
```

Utilizando transações

As regras a seguir se aplicam às transações:

- Uma transação começa com o comando `BEGIN TRANSACTION` e termina com o comando `END TRANSACTION` ou `ROLLBACK`. Uma instrução `END TRANSACTION` sem uma instrução `BEGIN`

TRANSACTION anterior gera um erro.

- Uma instrução ROLLBACK sem uma instrução BEGIN TRANSACTION anterior gera um erro.
- Uma transação, depois de começada, permanece em vigor até que o END TRANSACTION correspondente comece (ou até que seja emitido um comando ROLLBACK), mesmo em programas e funções, a menos que o aplicativo termine, o que causará uma reversão.
- O Visual FoxPro utiliza os dados armazenados no buffer da transação antes de utilizar os dados no disco para consultar os dados envolvidos nas transações. Isso garante a utilização dos dados mais atualizados.
- Se o aplicativo terminar durante uma transação, todas as operações serão revertidas.
- Uma transação funciona somente em um recipiente de banco de dados.
- Você não poderá utilizar o comando INDEX se ele sobrescrever um arquivo de índice existente ou se qualquer arquivo de índice .CDX estiver aberto.
- As transações são dimensionadas para sessões de dados.

As transações apresentam os comportamentos de bloqueio a seguir:

- Em uma transação, o Visual FoxPro exige um bloqueio no momento em que este é chamado direta ou indiretamente por um comando. Qualquer comando de desbloqueio direto ou indireto do usuário ou do sistema é armazenado até que a transação seja encerrada com os comandos ROLLBACK ou END TRANSACTION.
- Se você utilizar um comando de bloqueio tal como FLOCK() ou RLOCK() dentro de uma transação, a instrução END TRANSACTION não liberará o bloqueio. Nesse caso, você deverá explicitamente desbloquear qualquer bloqueio feito explicitamente dentro de uma transação. Além disso, as transações contendo comandos FLOCK() ou RLOCK() devem ser as mais curtas possíveis; caso contrário, os usuários podem ficar bloqueados nos registros durante muito tempo.

Aninhando transações

As transações aninhadas fornecem grupos lógicos de operações de atualização de tabela que são isolados dos processos simultâneos. Os pares BEGIN TRANSACTION...END TRANSACTION não precisam estar na mesma função ou procedimento. As regras a seguir se aplicam às transações aninhadas:

- Você pode aninhar até cinco pares BEGIN TRANSACTION...END TRANSACTION.
- As atualizações feitas em uma transação aninhada não são confirmadas até que se chame o END TRANSACTION mais afastado.
- Nas transações aninhadas, END TRANSACTION opera somente na transação iniciada pelo último BEGIN TRANSACTION emitido.
- Nas transações aninhadas, uma instrução ROLLBACK opera somente na transação iniciada pelo último BEGIN TRANSACTION emitido.
- A atualização mais interna em um conjunto de transações aninhadas nos mesmos dados tem precedência sobre todas as outras no mesmo bloco de transações aninhadas.

Observe no exemplo a seguir que, como as alterações em uma transação aninhada não são gravadas no disco e sim no buffer da transação, a transação interna irá sobrescrever as alterações feitas nos mesmos campos STATUS da transação anterior.

```
BEGIN TRANSACTION    && transação 1
    UPDATE EMPLOYEE ;    && primeira alteração
    SET STATUS = "Contract" ;
    WHERE EMPID BETWEEN 9001 AND 10000
    BEGIN TRANSACTION    && transação 2
        UPDATE EMPLOYEE ;
        SET STATUS = "Exempt" ;
        WHERE HIREDATE > {1/1/93}    &&  sobrescreve
    END TRANSACTION    && transação 2
```


END TRANSACTION && transação 1

O exemplo de transação aninhada a seguir exclui um registro do cliente e todas as faturas relacionadas a ele. A transação será revertida se ocorrerem erros durante um comando **DELETE**. Este exemplo exhibe as operações de atualização de tabela em grupos para proteger as atualizações de termos parciais e evitar conflitos de concorrência.

Exemplo de modificação de registros em transações aninhadas

Código	Comentários
DO WHILE TXNLEVEL() > 0 ROLLBACK ENDDO	Limpeza de outras transações.
CLOSE ALL SET MULTLOCKS ON SET EXCLUSIVE OFF	Estabelece o ambiente para a utilização de buffer.
OPEN DATABASE test USE mrgtest1 CURSORSETPROP('buffering',5) GO TOP	Ativa a utilização de buffer de tabela otimista.
REPLACE fld1 WITH "alterado" SKIP REPLACE fld1 WITH "outra alteração" MESSAGEBOX("modifica primeiro campo dos dois" + "registros em outra máquina")	Altera um registro. Altera um outro registro.
BEGIN TRANSACTION lSuccess = TABLEUPDATE(.T.,.F.) IF lSuccess = .F. ROLLBACK AERROR(aErrors) DO CASE CASE aErrors[1,1] = 1539 ... CASE aErrors[1,1] = 1581 ... CASE aErrors[1,1] = 1582	Inicia a transação 1 e tenta atualizar todos os registros modificados sem forçar. Se a atualização falhou, reverte a transação. Detecta o erro do AERROR(). Determina a causa da falha. Se um disparador falhou, trata esse erro.
CASE aErrors[1,1] = 1585 nNextModified = getnextmodified(0) DO WHILE nNextModified <> 0 GO nNextModified RLOCK() FOR nField = 1 to FCOUNT() cField = FIELD(nField) if OLDVAL(cField) <> CURVAL(cField)	Se um campo não aceitar valores nulos, trata esse erro. Se uma regra do campo foi violada, trata esse erro.
nResult = MESSAGEBOX; ("Dados alterados por " + "outro usuário. Manter "+	Se um registro foi alterado por outro usuário, localiza o primeiro registro modificado. Efetua um loop por todos os registros modificados, começando pelo primeiro. Bloqueia cada registro para garantir a sua atualização. Verifica em cada campo qualquer alteração.
)	Compara o valor no buffer com o valor no disco e apresenta uma caixa de diálogo ao usuário.

<pre> "alterações?", 4+48, ; "Registro Modificado") IF nResult = 7 TABLEREVERT(.F.) UNLOCK record nNextModified ENDIF EXIT ENDIF ENDFOR ENDDO </pre>	<p>Se o usuário respondeu 'Não', inverte e desbloqueia o registro.</p> <p>Sai do loop 'FOR nField...'. Vai para o próximo registro modificado.</p>
<pre> BEGIN TRANSACTION TABLEUPDATE(.T.,.T.) END TRANSACTION UNLOCK </pre>	<p>Inicia a transação 2 e força a atualização de todos os registros não invertidos. Encerra a transação 2. Libera o bloqueio.</p>
<pre> CASE aErrors[1,1] = 1700 ... CASE aErrors[1,1] = 1583 ... CASE aErrors[1,1] = 1884 ... OTHERWISE MESSAGEBOX("Mensagem de erro "+ "desconhecida: " + STR(aErrors[1,1])) ENDCASE </pre>	<p>Se o registro estiver sendo utilizado por outro usuário, trata esse erro.</p> <p>Se uma regra de linha foi violada, trata esse erro.</p> <p>Se houve uma violação de índice único, trata esse erro.</p> <p>Caso contrário, apresenta uma caixa de diálogo ao usuário.</p>
<pre> ELSE END TRANSACTION ENDIF </pre>	<p>Encerra a transação 1.</p>

Protegendo atualizações remotas

As transações podem evitar erros gerados pelo sistema durante as atualizações de dados em tabelas remotas. O exemplo a seguir utiliza uma transação para quebrar as operações de gravação de dados em uma tabela remota.

Exemplo de transação em uma tabela remota

Código	Comentário
<pre> hConnect = CURSORGETPROP('connecthandle') SQLSETPROP(hConnect, 'transmode', DB_TRANSMANUAL) BEGIN TRANSACTION </pre>	<p>Detecta o identificador de conexão e ativa as transações manuais.</p> <p>Começa a transação manual.</p>
<pre> lSuccess = TABLEUPDATE(.T.,.F.) IF lSuccess = .F. SQLROLLBACK(hConnect) ROLLBACK </pre>	<p>Tenta atualizar todos os registros sem forçar. Se a atualização falhou, reverte a transação na conexão para o cursor.</p>
<pre> AERROR(aErrors) DO CASE CASE aErrors[1,1] = 1539 ... </pre>	<p>Detecta o erro do AERROR().</p> <p>Se um disparador falhou, trata esse erro.</p>


```
ELSE
    SQLCOMMIT(hConnect)
END TRANSACTION
ENDIF
```

erro.

Se todos os erros foram tratados e toda a transação foi bem-sucedida, envie uma confirmação e termine a transação.

Gerenciando o desempenho

Se você possuir um aplicativo multiusuário em operação, poderá utilizar as sugestões a seguir para melhorar o desempenho:

- Colocar os arquivos temporários em uma unidade de disco local.
- Optar entre os arquivos de índice e de classificação.
- Planejar o acesso exclusivo aos arquivos.
- Cronometrar o bloqueio dos arquivos.

Colocar os arquivos temporários em uma unidade de disco local

O Visual FoxPro cria seus arquivos temporários em um diretório temporário padrão do Windows. As sessões de edição de texto podem também criar, temporariamente, uma cópia completa do arquivo em edição (um arquivo .BAK).

Se as estações de trabalho locais tiverem unidades de disco rígido com muito espaço livre, você poderá melhorar o desempenho, colocando esses arquivos temporários em uma unidade local ou de RAM. O redirecionamento desses arquivos para uma unidade local ou de RAM aumentará o desempenho, pois reduzirá o acesso à unidade de disco da rede.

Você pode especificar uma localização alternativa para esses arquivos, incluindo as instruções EDITWORK, SORTWORK, PROGWORK e TMPFILES no seu arquivo de configuração CONFIG.FPW. Para obter maiores informações sobre como gerenciar arquivos, consulte o capítulo 4, [Otimizando o sistema](#), no *Guia de Instalação e Índice Principal*.

Optar entre arquivos de índice e de classificação

Quando os dados contidos em uma tabela forem relativamente estáticos, o processamento de tabelas classificadas seqüencialmente, sem uma definição de ordem, melhora o desempenho. Isso não significa que as tabelas classificadas não podem ou devem tirar proveito dos arquivos de índice — o comando [SEEK](#), que requer um índice, é incomparável para localizar registros rapidamente. Entretanto, depois que você localiza um registro com SEEK, pode desativar a ordenação.

Planejar acesso exclusivo aos arquivos

Os comandos que são executados quando nenhum outro usuário precisa acessar os dados, como atualizações feitas à noite, podem se beneficiar da abertura dos arquivos de dados para utilização exclusiva. Quando os arquivos são abertos para utilização exclusiva, o desempenho melhora porque o Visual FoxPro não precisa testar o status dos bloqueios de registros ou de arquivos.

Cronometrar o bloqueio dos arquivos

Para reduzir a disputa entre usuários em relação ao acesso à gravação em uma tabela ou registro, diminua o intervalo de tempo do bloqueio de um registro ou tabela. Você pode fazer isso bloqueando o registro somente depois da sua edição, em vez de durante a edição. A utilização de buffer de linha otimista fornece a você o menor tempo de bloqueio.

Para obter maiores informações sobre como melhorar o desempenho, consulte o capítulo 4, [Otimizando o sistema](#), no *Guia de Instalação e Índice Principal*. Você pode também encontrar

informações sobre como melhorar o desempenho dos seus aplicativos cliente/servidor no capítulo 22, [Otimizando o desempenho do cliente/servidor](#), neste manual.

Gerenciando atualizações com visualizações

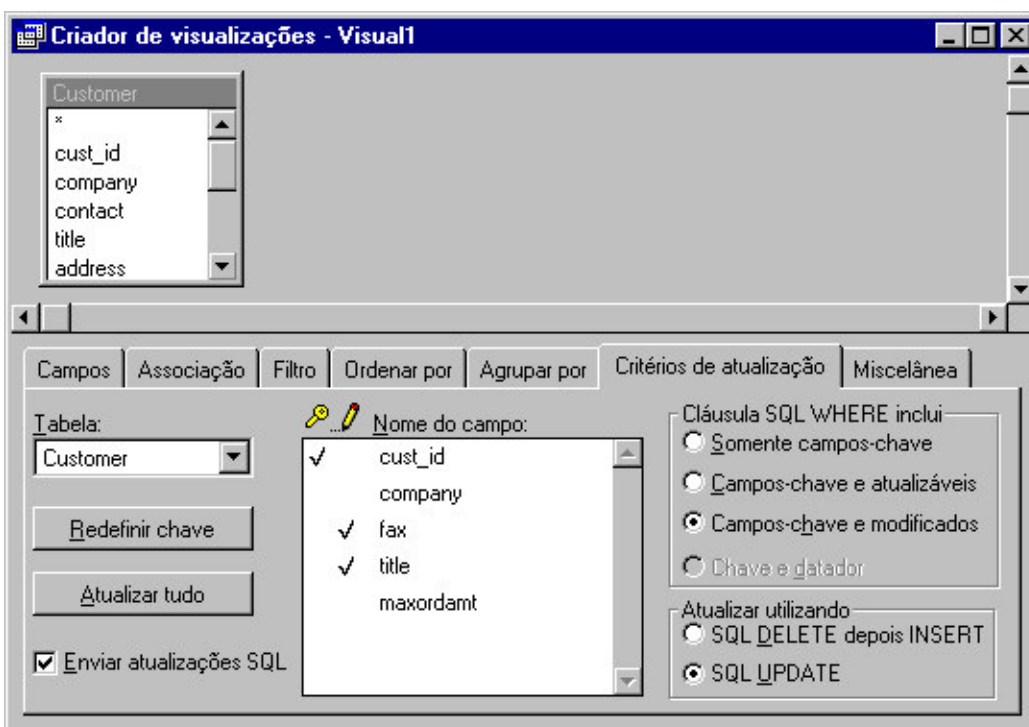
Você pode utilizar a tecnologia de gerenciamento de conflito de atualização encontrada nas visualizações Visual FoxPro para gerenciar o acesso de vários usuários aos dados. As visualizações controlam o que é enviado para as tabelas base subjacentes à visualização, utilizando a propriedade WhereType. Você pode definir essa propriedade para visualizações remotas e locais. A propriedade WhereType fornece quatro definições:

- DB_KEY
- DB_KEYANDUPDATABLE
- DB_KEYANDMODIFIED (padrão)
- DB_KEYANDTIMESTAMP

Ao escolher uma dessas quatro definições, você controla como Visual FoxPro constrói a cláusula WHERE para a instrução SQL Update enviada para as tabelas base da visualização. Você pode escolher a definição desejada, utilizando a guia **Critérios de atualização** do **Criador de visualizações** ou pode utilizar `DBSETPROP()` para definir WhereType para uma definição de visualização. Para alterar a definição de WhereType de um cursor de visualização ativo, utilize `CURSORSETPROP()`.

Por exemplo, suponha que você tem uma visualização remota simples baseada na tabela Customer que inclui sete campos: `cust_id`, `company`, `phone`, `fax`, `contact`, `title` e `timestamp`. A chave primária da visualização é `cust_id`.


A guia Critérios de atualização exibe os campos atualizáveis na visualização.



Somente dois campos foram tornados atualizáveis: `contact_name` e `contact_title`. Você deseja que o usuário possa alterar o contato da empresa e seu título a partir da visualização. No entanto, se outros fatores sobre a empresa mudarem, como o endereço, você desejará que as alterações sejam passadas para um coordenador que identificará o impacto causado pelas

alterações em sua empresa, como se a região de vendas para o cliente mudará. Agora que a visualização foi configurada para enviar atualizações, você poderá escolher WhereType de acordo com as suas preferências.

Suponha que você tenha alterado o nome no campo `contact` de um cliente, mas não alterou o valor no outro campo atualizável, `title`. Com base nesse exemplo, a seção a seguir abordará como a definição de WhereType teria impacto sobre a cláusula WHERE que o Visual FoxPro constrói para enviar o novo nome de contato para as tabelas base.

Para obter uma versão animada, clique aqui. 

Comparando somente o campo chave

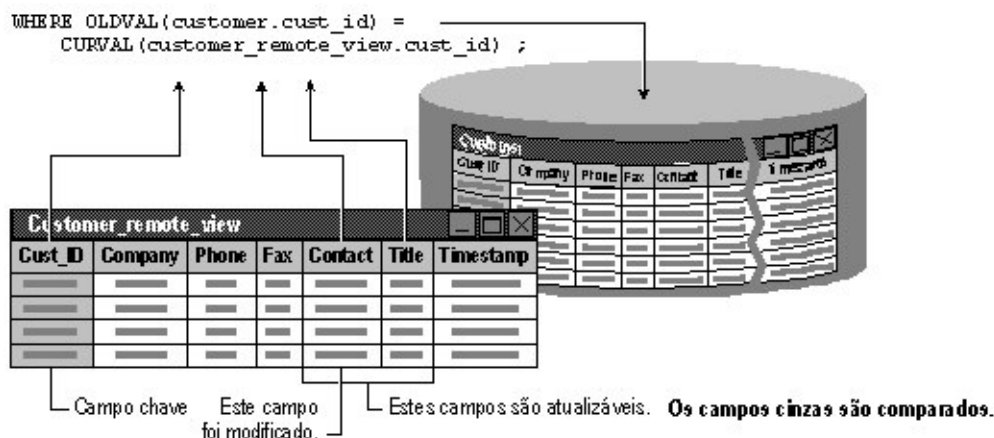
A atualização menos restritiva utiliza a definição `DB_KEY`. A cláusula WHERE utilizada para atualizar tabelas remotas é composta somente pelo campo chave primária especificado com a propriedade `KeyField` ou `KeyFieldList`. A menos que o valor no campo chave primária tenha sido alterado ou excluído na tabela base desde que o registro foi recuperado, a atualização se realizará.

No caso do exemplo anterior, o Visual FoxPro prepararia uma instrução de atualização com a cláusula WHERE que compara o valor no campo `cust_id` com o campo `cust_id` na linha da tabela base:

```
WHERE OLDVAL(customer.cust_id) = CURVAL(customer_remote_view.cust_id)
```

Quando a instrução de atualização é enviada para a tabela base, somente o campo chave da tabela base é verificado.

O campo chave na visualização é comparado com o seu correspondente na tabela base.



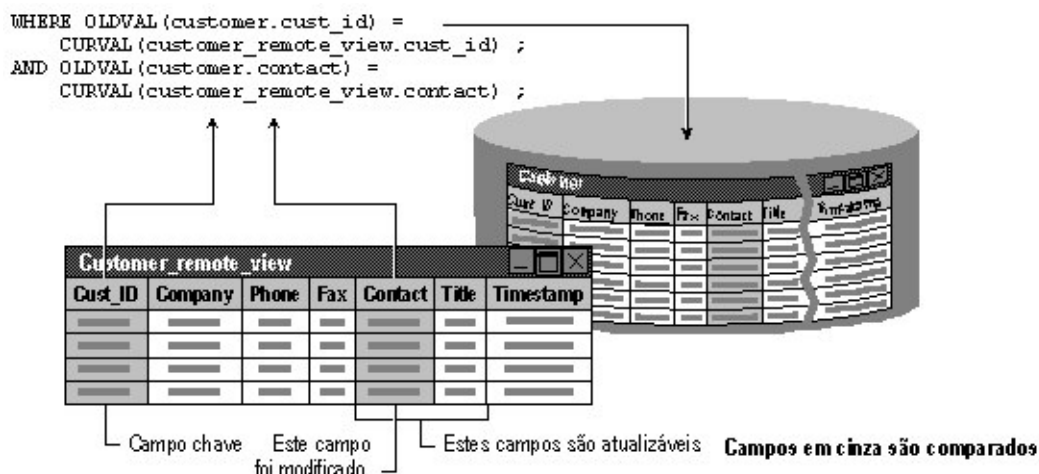
Comparando o campo chave e os campos modificados na visualização

A definição `DB_KEYANDMODIFIED`, a padrão, é ligeiramente mais restritiva do que a `DB_KEY`. `DB_KEYANDMODIFIED` compara somente o campo chave e qualquer campo atualizável modificado na visualização com os correspondentes na tabela base. Se você modificar um campo na visualização, mas o campo não for atualizável, os campos não serão comparados com os dados da tabela base.

A cláusula WHERE utilizada para atualizar tabelas base consiste em campos primários especificados com a propriedade `KeyFieldList` e qualquer outro campo modificado na visualização. No caso do exemplo anterior, o Visual FoxPro prepararia uma instrução de atualização que compara o valor no campo `cust_id` por ser o campo chave com o campo `contact`, pois o nome de contato foi alterado. Mesmo que o campo `title` seja atualizável, `title` não será incluído na instrução de atualização pois não foi modificado.

Os campos chave e modificados na visualização são comparados com os seus

correspondentes na tabela base.

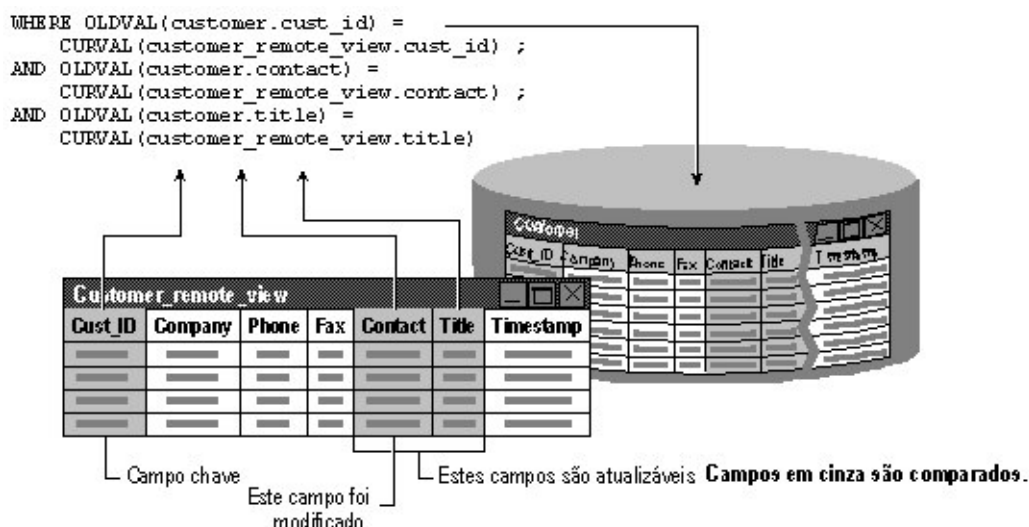


Comparando o campo chave e todos os campos atualizáveis

A definição DB_KEYANDUPDATABLE compara o campo chave e qualquer campo atualizável (se modificado ou não) na visualização com os seus correspondentes na tabela base. Se o campo for atualizável, mesmo se não foi alterado na visualização, se qualquer pessoa alterou esse campo na tabela base, a atualização falhará.

A cláusula WHERE utilizada para atualizar tabelas base consiste em campos primários especificados com a propriedade Key Field ou KeyFieldList e qualquer outro campo que seja atualizável. No caso do exemplo, o Visual FoxPro prepararia uma instrução de atualização que compara os valores dos campos `cust_id`, `contact` e `title` com os mesmos campos na linha da tabela base.

Todos os campos atualizáveis na visualização são comparados com os seus correspondentes na tabela base.



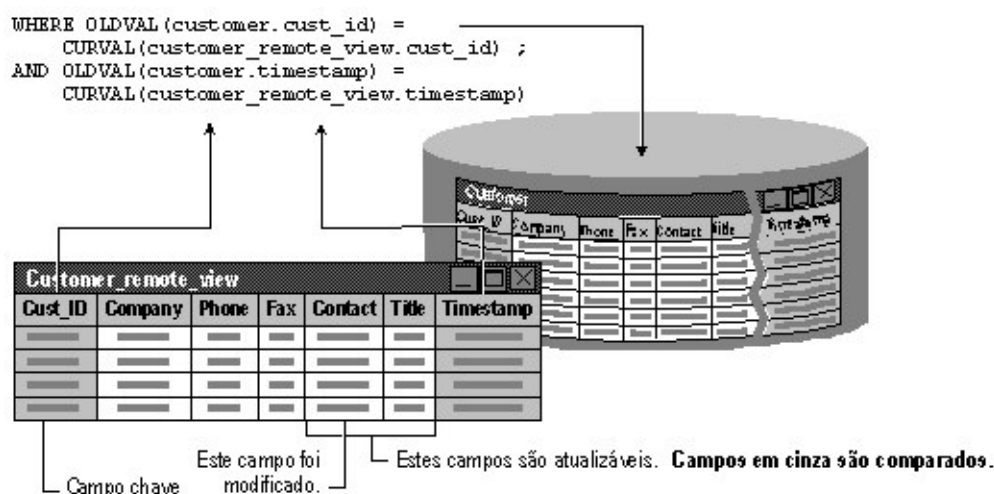
Comparando a marca de data e hora para todos os campos no registro da tabela base

A definição DB_KEYANDTIMESTAMP é o tipo mais restritivo de atualização e somente estará disponível se a tabela base tiver uma coluna de marca de data e hora. O Visual FoxPro compara a

marca de data e hora atual no registro da tabela base com a marca de data e hora no momento em que os dados são carregados para a visualização. Se qualquer campo no registro da tabela base foi alterado, mesmo se não for um campo que você esteja tentando alterar, ou mesmo um campo na visualização, a atualização falhará.

No caso do exemplo, o Visual FoxPro prepara uma instrução de atualização que compara os valores no campo `cust_id` e o valor no campo `timestamp` com os campos na linha da tabela base.

A marca de data e hora para o registro da visualização é comparada com a marca de data e hora no registro da tabela base.



Para haver uma atualização bem-sucedida dos dados utilizando a definição `DB_KEYANDTIMESTAMP` com uma visualização multitabela, é necessário que você inclua o campo de marca de data e hora na visualização para cada tabela atualizável. Por exemplo, se há três tabelas em uma visualização e você somente deseja atualizar duas utilizando a definição `DB_KEYANDTIMESTAMP`, é necessário levar os campos de marca de data e hora das duas tabelas atualizáveis para o seu conjunto de resultados. Você pode também utilizar valores lógicos na propriedade `CompareMemo` para determinar se os campos memo serão incluídos na detecção de conflitos.

Gerenciando conflitos

Se você utilizar buffers, transações ou visualizações, terá que gerenciar conflitos durante o processo de atualização.

Gerenciando conflitos no buffer

Você pode tornar as operações de atualização de dados mais eficientes se escolher cuidadosamente como e quando abrir, utilizar buffer e bloquear os dados em um ambiente multiusuário. Você deve limitar o tempo em que um registro ou tabela é submetido a conflitos de acesso. No entanto, os conflitos inevitáveis resultantes devem ser previstos e controlados. Um *conflito* ocorre quando um usuário tenta bloquear um registro ou tabela que já está bloqueado por outro usuário no momento. Dois usuários não podem bloquear o mesmo registro ou tabela ao mesmo tempo.

Seu aplicativo deve conter uma rotina para gerenciar esses conflitos. Se o aplicativo não tiver essa rotina de conflitos, o sistema poderá travar. Um *bloqueio perpétuo* ocorre quando um usuário bloqueou um registro ou tabela e tenta bloquear outro registro bloqueado por um segundo usuário. Este, por sua vez, está tentando bloquear o registro bloqueado pelo primeiro usuário. Tais situações são raras, porém, quanto mais tempo um registro ou tabela fica bloqueado, maiores serão as chances de ocorrer um bloqueio perpétuo.

Interceptando erros

A criação de uma aplicativo multiusuário ou a adição de suporte à rede para um sistema monousuário requerem que você lide com colisões ou interceptações de erros. A utilização de buffers de tabela e registro do Visual FoxPro simplifica, em parte, esse trabalho.

Se você tentar bloquear um registro ou tabela já bloqueado por outro usuário, o Visual FoxPro retornará uma mensagem de erro. Você pode utilizar o **SET REPROCESS** para lidar automaticamente com uma tentativa de bloqueio sem êxito. Esse comando, combinado com uma rotina **ON ERROR** e o comando **RETRY**, permite que você continue ou cancele as tentativas de bloqueio.

O exemplo a seguir demonstra o reprocessamento automático de uma operação mal-sucedida, utilizando o SET REPROCESS.

Utilizando SET REPROCESS e ON ERROR para gerenciar colisões de usuários

Código	Comentário
<pre>ON ERROR DO err_fix WITH ERROR(),MESSAGE() SET EXCLUSIVE OFF SET REPROCESS TO AUTOMATIC USE customer IF !FILE('cus_copy.dbf') COPY TO cus_copy ENDIF</pre>	<p>Esta rotina será executada se ocorrer um erro.</p> <p>Abre os arquivos de modo não exclusivo.</p> <p>O reprocessamento dos bloqueios sem êxito é automático.</p> <p>Abre a tabela.</p> <p>Cria uma tabela APPEND FROM , se necessário.</p>
<pre>DO app_blank DO rep_next DO rep_all DO rep_curr DO add_recs ON ERROR</pre>	<p>A rotina principal começa aqui. Estes comandos são exemplos de códigos que podem ser executados ao longo do programa.</p> <p>A rotina principal termina aqui.</p>
<pre>PROCEDURE app_blank APPEND BLANK RETURN ENDPROC</pre>	<p>Rotina para incluir um registro vazio.</p>
<pre>PROCEDURE rep_next REPLACE NEXT 1 contact WITH ; PROPER(contact) RETURN ENDPROC</pre>	<p>Rotina para substituir os dados no registro atual.</p>
<pre>PROCEDURE rep_all REPLACE ALL contact WITH ; PROPER(contact) GO TOP RETURN ENDPROC</pre>	<p>Rotina para substituir os dados em todos os registros.</p>
<pre>PROCEDURE rep_curr REPLACE contact WITH PROPER(contact) RETURN ENDPROC</pre>	<p>Rotina para substituir os dados no registro atual.</p>
<pre>PROCEDURE add_recs APPEND FROM cus_copy RETURN</pre>	<p>Rotina para incluir registros de outro arquivo.</p>

ENDPROC

O exemplo a seguir demonstra um procedimento de erro que começa quando o usuário pressiona ESC.

Gerenciamento de erros utilizando a tecla ESC

Código	Comentário
<pre>PROCEDURE err_fix PARAMETERS errnum, msg</pre>	Este programa é chamado quando se encontra um erro e o usuário interrompe o processo de espera.
<pre>DO CASE CASE errnum = 108 line1 = "O arquivo não pode ser bloqueado." line2 = "Tente novamente depois..." CASE errnum = 109 .OR. errnum = 130 line1 = "O registro não pode ser bloqueado." line2 = "Tente novamente depois." OTHERWISE line1 = msg + " " line2 = ; " Consulte o administrador do sistema." ENDCASE</pre>	Tenta descobrir que tipo de erro é este. É "O arquivo está sendo utilizado por outro usuário"? Ou "O registro está sendo utilizado por outro usuário"? Ou é desconhecido?
<pre>=MESSAGEBOX(line1 + line2, 48, "Erro!") RETURN</pre>	Exibe a mensagem de erro em uma caixa de diálogo com um ponto de exclamação e um botão OK .

Detectando e resolvendo conflitos

Durante as operações de atualização dos dados, especialmente em ambientes compartilhados, você pode querer determinar quais campos foram alterados ou quais são os valores originais e atuais nos campos alterados. A utilização de buffer do Visual FoxPro e as funções [GETFLDSTATE\(\)](#), [GETNEXTMODIFIED\(\)](#), [OLDVAL\(\)](#) e [CURVAL\(\)](#) permitem a você determinar quais campos foram alterados, encontrar os dados modificados e comparar os valores atuais, originais e editados. Desse modo, você poderá decidir como gerenciar um erro ou conflito.

► Para detectar uma alteração em um campo

- Após uma operação de atualização, utilize a função [GETFLDSTATE\(\)](#).

[GETFLDSTATE\(\)](#) funciona em dados que não estão em buffer; no entanto, esta função opera ainda melhor quando você ativa a utilização de buffer de registro. Por exemplo, utilize [GETFLDSTATE\(\)](#) no código de um botão **Ignorar** em um formulário. Quando o ponteiro do registro é movimentado, o Visual FoxPro verifica os status de todos os campos no registro, como no exemplo a seguir:

```
lModified = .F.
FOR nFieldNum = 1 TO FCOUNT( )    && Verifica todos os campos
    if GETFLDSTATE(nFieldNum) = 2    && Campo foi modificado
        lModified = .T.
        EXIT    && Insira para atualizar/salvar rotina aqui.
    ENDIF    && Veja o exemplo a seguir
ENDFOR
```

► Para detectar e localizar um registro alterado em dados de buffer

- Utilize a função [GETNEXTMODIFIED\(\)](#).

[GETNEXTMODIFIED\(\)](#), com zero como parâmetro, encontra o primeiro registro modificado. Se

outro usuário fizer alterações na tabela de buffer, qualquer alteração encontrada por um comando **TABLEUPDATE()** no seu buffer causará conflitos. Você pode avaliar os valores conflitantes e solucioná-los, utilizando as funções **CURVAL()**, **OLDVAL()**, e **MESSAGEBOX()**. **CURVAL()** retorna o valor atual do registro no disco, enquanto **OLDVAL()** retorna o valor do registro no momento em que foi para o buffer.

► **Para determinar o valor original de um campo de buffer**

- Utilize a função **OLDVAL()**.

OLDVAL() retorna o valor de um campo de buffer.

► **Para determinar o valor atual em disco de um campo de buffer**

- Utilize a função **CURVAL()**.

CURVAL() retorna o valor atual em disco de um campo de buffer antes que qualquer edição tenha sido feita.

Você pode criar um procedimento de gerenciamento de erros que compara os valores atuais com os originais, permitindo que você confirme a alteração atual ou aceite uma alteração anterior nos dados em um ambiente compartilhado.

O exemplo a seguir utiliza **GETNEXTMODIFIED()**, **CURVAL()** e **OLDVAL()** para fornecer informações ao usuário a fim de que ele possa fazer opções em uma operação de atualização. Esse exemplo continua com a detecção do primeiro registro modificado e pode estar incluído em um botão **Atualizar** ou **Salvar** de um formulário.

Código de evento Click para um botão Atualizar ou Salvar

Código	Comentário
DO WHILE GETNEXTMODIFIED(nCurRec) <> 0	Efetua um loop pelo buffer.
GO nCurRec	
RLOCK()	Bloqueia o registro modificado.
FOR nField = 1 TO FCOUNT(cAlias)	Procura conflitos.
cField = FIELD(nField)	
IF OLDVAL(cField) <> CURVAL(cField)	Compara o valor original com o atual em disco e pergunta ao usuário o que fazer com o conflito.
nResult = MESSAGEBOX("Dados ; alterados por outro usuário. ; Manter alterações?", 4+48+0, ; "Registro Modificado")	
IF nResult = 7	Se o usuário selecionar 'Não', reverte este registro e remove o bloqueio.
TABLEREVERT(.F.)	
UNLOCK RECORD nCurRec	
ENDIF	
ENDIF	
ENDFOR	
nCurRec = GETNEXTMODIFIED(nCurRec)	
ENDDO	Encontra o próximo registro modificado.
TABLEUPDATE(.T., .T.)	Força a atualização de todos os registros.

Detectando conflitos utilizando campos Memo

Você pode utilizar a propriedade **CompareMemo** para controlar quando os campos memo serão utilizados para detectar conflitos de atualização. Essa propriedade de visualização e cursor determina se os campos memo (tipos M ou G) serão incluídos na cláusula **WHERE** de atualização. A definição padrão, **Verdadeiro (.T.)**, significa que os campos memo são incluídos na cláusula **WHERE**. Se você definir essa propriedade como **Falso (.F.)**, os campos memo não participarão da cláusula **WHERE** de atualização, independentemente das definições de **UpdateType**.

A detecção de conflito otimista em campos Memo é desativada quando CompareMemo é definida como Falso. Para obter detecção de conflitos em valores memo, defina CompareMemo como Verdadeiro (.T.).

Regras para gerenciamento de conflitos

Os gerenciamento de conflitos encontrado em ambientes multiusuário pode requerer código repetitivo e extensivo. Uma rotina de gerenciamento de conflitos completa executa os procedimentos a seguir:

- Detecta um conflito
- Identifica a natureza e a localização do conflito
- Fornece informações suficientes para que o usuário possa solucionar o conflito de forma inteligente

Para obter um exemplo de uma rotina de gerenciamento de conflitos, consulte a classe de verificação de dados em SAMPLES.VCX, localizada em VFP\SAMPLES\CLASSES. Basta adicionar a classe a um formulário e chamar o método CheckConflicts antes de realizar qualquer operação de registro de dados de buffer na tabela, por exemplo, a movimentação do ponteiro do registro se você estiver utilizando buffer de linha, fechando uma tabela ou emitindo TABLEUPDATE().