

Porque o `pack` comando é um problema (ver [O Comando Pack](#)), um substituto deve ser encontrado. reciclagem Record é a resposta.

Independentemente dos perigos da `pack`, a finalidade é remover registros excluídos. No entanto, a maioria dos arquivos de dados crescer ao longo do tempo e, eventualmente, estabilizar-se em um determinado tamanho como os registros são adicionados ou removidos durante o exercício de uma actividade normal.

O princípio por trás da reciclagem registro é a reutilização dos registros excluídos, eliminando assim a necessidade de `pack -los`.

O Método

Em vez de usar o `append` comando para adicionar um outro registro em uma tabela, o programa deve procurar primeiro por um registro excluído e apresentá-lo como o "record" de novo. Esta abordagem economiza tempo porque o aplicativo não precisa perguntar o sistema operacional a alocar mais espaço para a tabela.

reciclagem Record requer duas funções principais, um para adicionar registros (aqui ele é chamado `AddRec()`) e outro para remover registros (`EraseRec()`).

Nesta implementação, o `EraseRec()` função não só apaga o registro, mas também apaga todos os campos. Isso fornece um pouco de segurança, mas também torna mais fácil para o `AddRec()` função para determinar se um registro é um candidato para a reciclagem. Assim, os registros que só são apagados (mas não apagado com `EraseRec()`) não vão ser reciclados.

O `EraseRec()` função é chamada sempre que `delete` é usado agora. A diferença é que `EraseRec()` bloqueia e desbloqueia o registro para você.

O `AddRec()` função é chamada sempre que quiser um novo recorde:

```
se AddRec ()
    / / Tela de entrada de vídeo, etc ..
    / / ...
    dbunlock ()
outro
    ErrorMessage ("Não é possível adicionar um novo recorde!")
endif
```

Embora `AddRec()` irá funcionar na maioria das vezes, você precisa decidir o que fazer se ele falhar. Isso depende da sua aplicação.

O Código

Vamos olhar o código fonte para as funções de reciclagem de registro. Na discussão a seguir, o arquivo de programa (**RECYCLE.PRG**) é dividido para que ele possa ser examinado em seções, mas você pode colá-lo de volta juntos antes de usá-lo.

possíveis melhorias para o código-fonte são apresentados no final.

```
*=====
* Nome do arquivo | RECYCLE.PRG
* Descrição | Código de reciclagem de registros.
* | Autor Greg Holmes, gregh@ghservices.com
*-----
* Rotinas | AddRec ()
* | EraseRec ()
* | RecIsEmpty ()
* | EmptyType (CType)
*=====

# Include "dbstruct.ch "
```

Esta seção do arquivo de programa contém o cabeçalho descritivo, que indica os nomes das funções que se seguem. Além disso, o arquivo de cabeçalho **dbstruct.ch** está incluído. Define constantes que são usadas como desvios para a matriz estrutura da tabela (que é usado no `EraseRec()` função).

```
*-----
* AddRec | Função
* Objetivo | Adicionar um registro por encontrar um ou anexando.
* Parâmetros |
* Retorna .T. | / .F. - Se bem sucedida.
* Assume | Esta função executa mais rápido se um índice
* | É aberta (registros vazios flutuar para cima).
* | Se não houver nenhum índice de abrir, em seguida, 'append
* | Em branco "é executado.
* Efeito Colateral | O ponteiro do registro está apontando para a esquerda
* | * Novo registro (se for bem sucedida), ou em
```

```

| * Fim-de-arquivo (se não for bem sucedida).
* | Se for bem sucedido, então novo registro está bloqueado.
*-----
AddRec função ()
    lResultado local: =. F., lDeleted

    lDeleted: = set (_SET_DELETED. F.)

    if. não. empty (ordkey ()) // Não é um índice.
        dbgotop () // float recs Empty 'para cima.
        se RLOCK () // registro de bloqueio de integridade.
            se excluído ()
                se RecIsEmpty ()
                    dbrecall ()
                    dbcommit ()
                    lResultado: =. T.
                outro
                    dbunlock ()
            endif
        outro
            dbunlock ()
        endif
    endif

    if. não. lResultado // Não é capaz de reciclar.
        dbappend () // Standard 'append' em branco.
        se neterr ()
            dbgobottom ()
            dbskip () // Vai para o fim-de-arquivo.
        outro
            lResultado: =. T.
        endif
    endif

    set (_SET_DELETED, lDeleted)

lResultado retorno

```

O `AddRec()` função é uma das duas funções principais para a reciclagem de registro, e é responsável por encontrar registros disponíveis. Um registro é um candidato se ele é excluído e todos os campos estão vazios.

Primeiro, a função torna visível com registros excluídos `set (_SET_DELETED, .F.)` e salva o estado anterior do pavilhão.

Como os valores em branco sempre aparecem em primeiro lugar em uma tabela indexada (a menos que o índice é decrescente), o código verifica se existe um índice ativo. Se assim for, os movimentos de código para a parte superior do índice em branco onde os registros são mais prováveis de aparecer. Para garantir que os dados de registro permanece consistente para as poucas linhas de código seguinte, o registro está bloqueado.

Para a velocidade, o código determina então se o registro mais alto é suprimido. Esta verificação é realizada antes do mais lento `RecIsEmpty()` função é chamada. A `RecIsEmpty()` função é descrita abaixo.

Se um excluído, registro em branco é encontrado, em seguida, recorde-se, ea mudança está empenhada em disco. A bandeira resultado é então definido para indicar sucesso.

A última parte da função realiza um padrão de `append blank` se a reciclagem de registro foi bem sucedido. Se o `append blank` falhar, o ponteiro do registro é movido para o fim do arquivo, caso contrário, o pavilhão resultado é definido para indicar sucesso.

Finalmente, a bandeira é definida excluído de volta ao estado que tinha aquando da entrada para a função. Note-se que o novo registro é deixada trancada se a função foi bem-sucedida em encontrar ou adicionar um novo registro. Isso copia o comportamento do padrão de `append blank` comando que *adiciona e bloqueia* o registro.

```

*-----
* EraseRec | Função
* Objetivo | todos os campos em branco e registro como marca que foi excluída.
* Parâmetros |
* Retorna .T. | / .F. - Se bem sucedida.
*-----
EraseRec função ()
    lResultado local: =. F., nLen, NL, aStruct

```

```

se RLOCK ()
  aStruct: = dbstruct ()
  nLen: = len (aStruct)
  para NL: = 1 para nLen
    fieldput (NL, EmptyType (aStruct [NL] [DBS_TYPE]))
  próximo
  dbdelete ()
  dbcommit ()
  dbunlock ()
  lResultado: =. T.
endif

lResultado retorno

```

O `EraseRec()` função apaga todos os campos do registro, usando um `for..next` loop para visitar cada campo. O `EmptyType()` função é descrito a seguir.

```

*-----
* RecIsEmpty | Função
* Objetivo | Procurar se os campos deste registro são todos vazios.
* Parâmetros |
* Retorna .T. | / .F.
*-----
RecIsEmpty função ()
  lResultado local: =. T., nLen, NL

  nLen: = fcount ()
  para NL: = 1 para nLen
    lResultado: = lResultado. e. vazio (fieldget (NL))
    if. não. lResultado
      saída
    endif
  próximo NL

  lResultado retorno

```

Esta função de visitas de cada campo em um registro para determinar se todos os campos estão vazios. Uma otimização foi adicionado verificações que se `lResult` é falsa cada vez através do loop. Isso faz com que `for..next` laço para parar assim como um campo vazio não for encontrado.

```

*-----
* EmptyType | Função
* Objetivo | Voltar um valor vazio do tipo desejado.
* Parâmetros pcType | - do tipo para criar.
* Retorna xResult | - um valor vazio.
*-----
função EmptyType (pcType)
  xResult local

  do caso
    caso pcType == 'A'
      xResult: = ()
    caso pcType == 'B'
      xResult: = (| | NIL)
    caso pcType == 'C'
      xResult: = ''
    caso pcType == 'D'
      xResult: = CTOD ('')
    caso pcType == 'M'
      xResult: = ''
    caso pcType == 'N'
      xResult: = 0
    caso pcType == 'L'
      xResult: =. F.
    caso contrário
      xResult: = NIL
  endcase

  xResult retorno

```

Esta última função retorna um valor vazio simples do tipo especificado. A maioria das CA Clipper tipos de dados possível, estão representados no `do case` comunicado.

Melhorias

Existem várias alterações que podem ser feitas para as funções anteriores.

- *Não registros em branco, basta apagar*

Esta é realmente uma simplificação do código que permitiria acelerar o `AddRec()` função porque ele não teria para verificar se os campos vazios. Além disso, o `aStruct` matriz e `for..next` ciclo pode ser removida do `EraseRec()` função.

- *Olhe mais para registros excluídos*

O código-fonte acima não tente muito difícil encontrar registros excluídos que são candidatos para a reciclagem de registro. Código pode ser adicionado para verificar "vários" registros no início do arquivo para os campos em branco. Pode ser razoável para saltar através da mesa para até 2 segundos, enquanto olhando para os candidatos. Além disso, o `rlock()` e `dbappend()` chama a função deve ser realizada mais de uma vez para aumentar a possibilidade de sucesso.

- *Utilização de um índice específico para registros excluídos*

O aprimoramento mais elegante que incluem um índice específico (para cada tabela) que armazena apenas registros excluídos. O índice poderia ser criada com qualquer tecla, mas com uma condição:

```
index on str(recno()) for deleted() to FILENAME.NTX
```

Em seguida, o `AddRec()` função que define a ordem para o índice de especial e chamar `dbgotop()`. Se `eof()` é encontrado, então não há registros apagados na tabela. Este método é mais fácil de gerir, se você usar um RDD que suporta arquivos de índice de chave múltipla, como o driver DBFCDX. No formato de arquivo DBFCDX, cada ordem do índice é dado um nome ou *marca*, que é utilizado quando da escolha da ordem do índice. Assim, cada tabela pode ter um fim índice chamado "Eliminado" e não haveria conflito, porque cada tabela teria seu próprio arquivo de índice CDX.