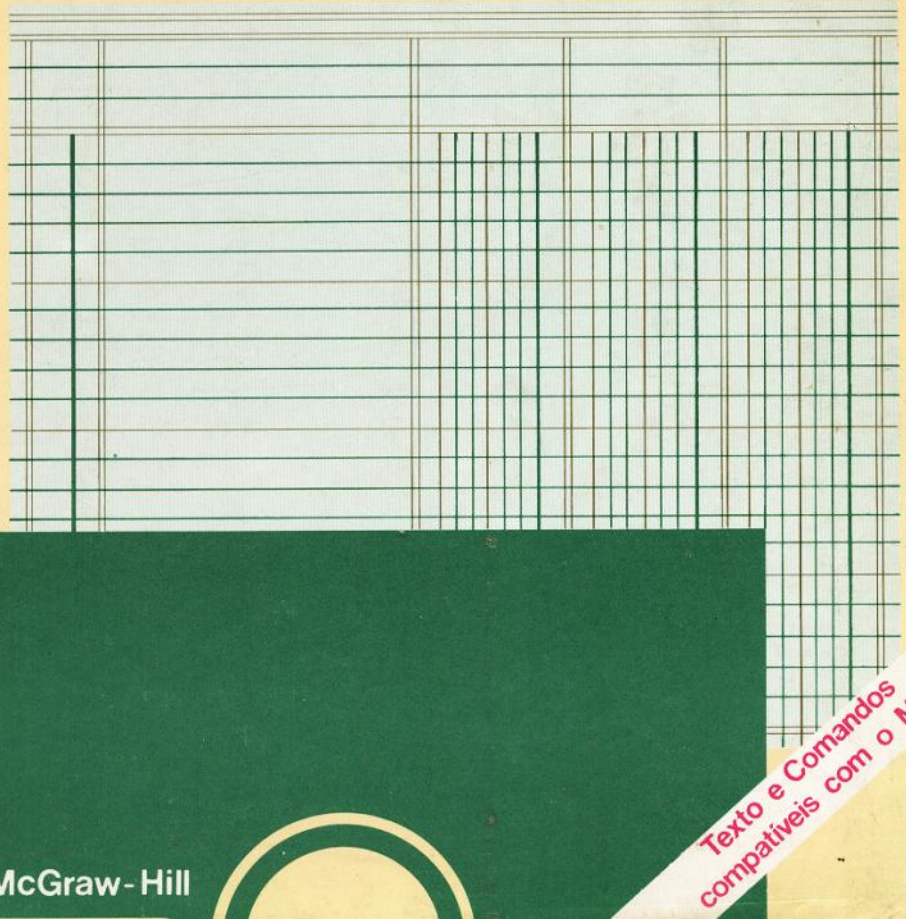
SÉRIE McGRAW-HILL/DATALÓGICA

dBASE II

ROBERT A. BYERS

APLICAÇÕES COMERCIAIS



Texto e Comandos compatíveis com o MSX

McGraw-Hill

dBASE II

APLICAÇÕES COMERCIAIS

dBASE II

APLICAÇÕES COMERCIAIS

Robert A. Byers

Tradução e Revisão Técnica
Equipe Datalógica

DATALÓGICA
São Paulo • Rio de Janeiro
Avenida Paulista, 2028, 16º andar
São Paulo – CEP 01098
(011) 283-0355

McGraw-Hill
São Paulo
Rua Tabapuã, 1.105, Itaim-Bibi
CEP 04533
(011) 881-8604 e (011) 881-8528

Rio de Janeiro • Lisboa • Porto • Bogotá • Buenos Aires • Guatemala • Madrid • México • New York • Panamá • San Juan • Santiago

Auckland • Hamburg • Kuala Lumpur • London • Milan • Montreal • New Delhi • Paris • Singapore • Sydney • Tokyo • Toronto

Coordenadora de Revisão: Daisy Pereira Daniel
Supervisor de Produção: Edson Sant'Anna
Capa: Criação: DATALÓGICA/McGraw-Hill
Arte Final: Cyro Giordano

CIP-Brasil. Catalogação-na-Publicação
Câmara Brasileira do Livro, SP

B997d

Byers, Robert A.
dBASE II : aplicações comerciais / Robert A. Byers ; tradução e revisão técnica Equipe DATALÓGICA. – São Paulo : McGraw-Hill do Brasil : DATALÓGICA , 1985
(Série McGraw-Hill – DATALÓGICA)

1. dBASE II (Programa de computador) 2. Processamento eletrônico de dados – Administração de empresas I. Título

17. CDD-651.8
18. -001.6425
17. -658.506
18. CDD-658.05425

85-1472

Índices para catálogo sistemático:

1. dBASE II : Computadores : Programas : Processamento 651.8 (17.) 001.6225 (18.)
2. Processamento de dados : Sistemas de "software" financeiro : Administração de empresas 658.506 (17.) 658.05425 (18.)
3. Sistemas de "software" : Aplicações financeiras : Processamento de dados : Administração de empresas 658.506 (17.) 658.05425 (18.)

AGRADECIMENTOS ESPECIAIS À EQUIPE DATALÓGICA:

Ademir Avila
André Noschese
Denise T. de Souza
Eduardo José S. Engelmann
Eduardo P. da Conceição
Eliana De S. Figueiredo
José Rubens S. Toledo
Marcio Arruda Cunha
Marcos B. Prestes Cesar
Maria Raquel Luperi

Não Fraudarás!

Seja Como For, é **Errado**

Pessoas que jamais entrariam em uma loja para furtar um produto de software, não hesitariam em fazer várias cópias do mesmo software. O resultado é o mesmo. A atitude é igualmente errada.

Muitas pessoas não percebem o dispendioso impacto causado ao criador do software e à comunidade de clientes. Em uma transação envolvendo o software, existe entre o autor e o cliente uma relação de confiança mútua. O cliente confia em que o autor criou um produto que proporcionará o resultado desejado, funcionará de acordo com as especificações e está apropriadamente documentado e com suporte. O autor confia em que o cliente usará apenas as cópias pelas quais pagou uma licença, mesmo que seja relativamente fácil fazer cópias adicionais não autorizadas. A duplicação e uso não autorizado de software, além de imoral, viola nossa **Legislação de Direitos Autorais** (Lei nº 5988, de 14 de dezembro de 1973) e constitui crime contra a propriedade intelectual (Código Penal, artigo 184), além de privar injustamente os criadores de software do benefício que têm direito a receber por seu trabalho.

Quem Comete o Crime?

O **Softfurto** é executado por indivíduos que fazem cópias ilegais para seu próprio uso, ou para um amigo; a **Pirataria de Software** ocorre quando organizações escolhem conscientemente encorajar, ou inconscientemente permitir que funcionários realizem e usem cópias ilegais de software.

Ambas as práticas violam a Lei Brasileira de Direitos Autorais, e expõem os indivíduos e empresas envolvidas às penalidades previstas em lei.

Cuidado! Alguém Pode Estar Expondo sua Organização

Embora acreditemos que a maioria das pessoas não encoraja a **pirataria de software** e o **softfurto**, sugerimos que verifique, por meio de uma *auditoria interna contínua*, a fim de assegurar-se de que ninguém os esteja praticando, evitando assim as ações previstas pela lei e, conseqüentemente, futuros embaraços para sua empresa.

Agindo dessa forma você estará colaborando para uma melhor conscientização do problema da violação da Legislação de Direitos Autorais no Brasil.

NOTA DOS EDITORES

O mundo está mudando com a velocidade da informação.

Bancos eletrônicos, compras e vendas sem dinheiro nem cheque, notícias cruzando o mundo em segundos. Vídeo jogos, vídeo texto, telefone sem fio.

São os computadores tomando conta do nosso presente, reorganizando nossa vida, alterando nosso comportamento.

É preciso adequar nossa linguagem ultrapassada às novas linguagens do futuro. Como? Adquirindo cultura, a nova cultura da informática.

É preciso que você, executivo, homen de informática, homem comum, entenda esse processo em toda a sua amplitude, assimilando suas novas linguagens e se conscientizando da influência dos computadores no seu dia a dia social e dos negócios.

Este é o propósito da série "McGRAW-HILL/DATALÓGICA", antecipando o seu futuro.

São Paulo, setembro de 1985

MILTON MIRA DE ASSUMPÇÃO FILHO
McGraw-Hill – Diretor Geral

OCTAVIO AUGUSTO SLEMER
DATALÓGICA – Diretor Geral

AGRADECIMENTOS

Escrever um livro está longe de ser uma tarefa para uma única pessoa. Embora o autor receba todo o crédito, é necessário uma equipe e um trabalho de equipe para produzir um livro. Poucas pessoas trabalharam muito para produzir este. Meus agradecimentos a todos. Em especial, gostaria de agradecer a Monet Thomson que fez a maior parte do trabalho de edição, a Steve Kurasch por sua revisão dos programas e a Mary Lincoln que deu o máximo para que eu me mantivesse dentro de uma certa programação de prazos. Jane Mellin merece o crédito por ter ajudado a desenvolver o trabalho de modo eficiente e organizado. Meu agradecimento também a meu filho Ken, minha filha Lauren e minha amiga muito especial Charlene Ebert por sua paciência nestes últimos meses.

Robert A. Byers

SUMÁRIO

INTRODUÇÃO

Este livro é destinado a pessoas que desejam desenvolver programas usando o dBASE II. É, na verdade, uma seqüência do *Everyman's Database Primer*. Para aproveitá-lo ao máximo, você deverá ou ter completado o *Primer* ou ter algum conhecimento do dBASE II.

O livro está dividido em três partes. Os três capítulos da Primeira Parte fazem uma revisão dos fundamentos do dBASE II e fornecem a base para os demais capítulos. Não há referências ao *Primer* ou ao manual do dBASE II.

A segunda e maior parte consiste em sete capítulos que descrevem a utilização do dBASE II em vários problemas financeiros. Nós montamos um esquema básico para um sistema de software financeiro, com uma série de programas para arquivos de clientes, listas e etiquetas de endereçamento e gerenciamento de inventário. Com este esquema, abordamos os problemas mais dinâmicos vinculados à entrada de pedidos de venda, preenchimento de pedidos e faturamento.

Os programas na segunda parte possuem uma função dupla. Cada um deles resolve um problema simples de contabilidade e demonstra alguns dos recursos da linguagem de programação do dBASE II. Com isso, eles fornecem ao leitor exemplos de programação. O objetivo principal dos programas é o de mostrar ao leitor como lidar com o desenvolvimento de programas com o dBASE II para resolver um problema específico. O sistema de software em si não tem a intenção de ser completo.

O texto e os programas são exemplificados por telas e figuras. Quando é necessário interromper linhas do programa no texto, um sinal de ■ foi colocado onde a linha do programa está interrompida. No programa, as variáveis de memória foram colocadas em letras minúsculas para que você possa distingui-las de nomes de campo e de arquivo. No texto, as variáveis de memória foram inicializadas.

Use sempre o sinal de aspas simples (') ou de aspas duplas (") em ambos os lados de suas *strings* de caracteres, ao fazer o programa. Nossa impressão apresenta aspas fechadas e abertas nas partes de texto, ilustrando aspectos do problema. Entretanto, você deverá usar as aspas em seu teclado do computador ao escrever seus programas. *Nunca* use o sinal de acentuação.

A última parte consiste em quatro capítulos. Ela lida com tópicos e artifícios especiais, bem como explicações genéricas de questões como, por exemplo, como aproveitar ao máximo o comando REPORT e como lidar com a data como um tipo de dados. O leitor deverá buscar esses artifícios ao desenvolver seus próprios programas.

O método do livro é realmente o do "aprender fazendo" – a maioria de nós aprende mais facilmente quando está tentando efetivamente fazer algo. Um dos melhores métodos de aprender é examinar o método usado por alguma outra pessoa para resolver um determinado problema e então adaptar a solução para resolver o nosso problema específico. A nossa adaptação será exclusiva e, por sua vez, se tornará modelo para alguma outra pessoa.

Parte Um

FUNDAMENTOS DO dBASE II

ELEMENTOS BÁSICOS SOBRE OS BANCOS DE DADOS

Quase todas as profissões ou empresas requerem algum sistema bem organizado para o armazenamento e recuperação de informações. Já se foram os dias dos cálculos mentais; mesmo a microempresa pode se beneficiar de um eficiente sistema gerenciador de dados. E o gerenciamento de dados é, sem dúvida, um dos mais úteis empregos que se pode dar ao microcomputador.

O banco de dados é simplesmente um conjunto de dados organizados de forma a serem usados com um determinado objetivo. Um exemplo comum de banco de dados é a lista telefônica, um conjunto de informações organizadas de modo a possibilitar a fácil localização de números telefônicos. Outro exemplo comum é o sistema de índice por cartões, usado em bibliotecas. Esses bancos de dados têm a finalidade mais ampla de tornar eficientes os sistemas telefônicos e bibliotecários.

Empresas e outras organizações têm seus próprios sistemas de bancos de dados: arquivos de clientes, arquivos com informações pessoais, inventários, registros de vendas, tabelas de impostos etc. As escolas têm currículos dos alunos, listas de chamadas, arquivos de funcionários, relatórios de freqüência e inventários. Todos esses bancos de dados possibilitam o bom funcionamento de uma empresa ou organização.

A maioria dos bancos de dados é montada como tabelas com linhas e colunas – este é um modo de organizar informações que todos usamos e intuitivamente compreendemos. A lista telefônica, por exemplo, é organizada dessa forma:

NOME	ENDEREÇO	TELEFONE
Cardoso, B.	r 2 av. B. Leme	389-8923
Dias, E. Z.	r 123 al. Franca	555-1234
Junqueira, A.	r 8931 L. Ferreira	789-4309

Antes da era do computador, todos os bancos de dados eram armazenados em papel, livros-razão ou em fichários. Para criar, modificar, armazenar e recuperar dados nesses bancos, em papel, era necessária a efetiva manipulação física dos dados computadorizados, a informação era armazenada como áreas magnéticas em um disco. Usando o computador não é possível o acesso *físico* aos dados; assim, torna-se necessário um método para se lidar com eles. Nosso veículo será então uma forma de software para computador – um Sistema Gerenciador de Bancos de Dados – Database Managements System (DBMS). O dBASE II, que examinamos nesse livro, é um DBMS bastante conhecido. Uma vez que o usuário conheça os elementos ou comandos básicos, com o sistema gerenciador de dados ele poderá manipular vastas quantidades de dados muito mais facilmente que com aqueles armazenados em papel.

Como a maioria dos sistemas gerenciadores de bancos de dados, o dBASE II organiza os dados em linhas e colunas. As colunas são chamadas *campos*. As linhas são chamadas *registros*. Os títulos das colunas são chamados *nomes de campos*. O DBMS é composto por várias partes funcionais, como mostra a Figura 1.

Freqüentemente, usa-se dois ou mais bancos de dados em conjunto; por exemplo, se tivermos um banco de dados de estudantes e/ou outro de professores, como mostra a Figura 2. Podemos saber quem é o professor de Tomás Juliano por meio de uma referência cruzada às duas tabelas do banco de dados, usando como referência a sala de aula de Tomás. Ao fazer isso, estamos relacionando a informação, de uma das tabelas, à informação de uma outra. Esta é uma noção simples que utilizamos o tempo todo.

O sistema gerenciador de dados, que os utiliza organizados como linhas e colunas para possibilitar ao usuário relacionar (em referência cruzada) dados de tabelas separadas em bancos de dados, é denominado *relacional*. O dBASE II é um sistema gerenciador de dados relacional.

Em um certo sentido, o sistema gerenciador de banco de dados é semelhante a um rapaz do estoque em um supermercado: ele cuida de todas as tarefas rotineiras vinculadas ao armazenamento, manutenção, exclusão, modificação e recuperação de dados. Há geralmente três "linguagens" incorporadas para possibilitar a comunicação com esse trabalhador versátil.

- A linguagem de definição de dados – *Data Definition Language* (*DDL*) – permite *criar e modificar a estrutura* de um banco de dados. Em um sistema relacional como o dBASE II, a estrutura consiste nos nomes de campo, no tamanho de cada um deles e no tipo de dados que serão armazenados em cada campo (tipo de campo).

- A linguagem de manipulação de dados – *Data Manipulation Language* (*DML*) – possibilita *modificar o conteúdo* do banco de dados. O acréscimo, a edição e a exclusão de registros são tarefas da DML.

- A linguagem de consulta permite *localizar e recuperar dados* com a finalidade desejada. Ela também permite conectar o sistema a um programa aplicativo.

- O gerador de relatórios permite extrair dados do banco de dados para produzir saídas tanto impressas como no vídeo.

No dBASE II, os três elementos "linguagem" e o gerador de relatórios são combinados em um pacote integrado chamado "Linguagem de Desenvolvimento Aplicativo" – *Applications Development Language* (*ADL*). Para iniciar no dBASE II é necessário apenas conhecer um pouco de sua terminologia.

A Terminologia Básica do dBASE II

Quando for inserir os dados no computador, o usuário precisará estabelecer um arquivo, dando um nome ao arquivo, estabelecendo a estrutura de registro e informando ao computador o tipo e tamanho de cada campo. O dBASE II tem, incorporadas, mensagens do sistema operacional que auxiliarão o usuário a criar os arquivos.

Os Nomes dos Bancos de Dados

Os bancos de dados do dBASE II são armazenados em arquivos em disco. Cada arquivo tem um nome específico, chamado "nome de arquivo". Como o dBASE II utiliza o sistema operacional do computador para armazenar ou recuperar informações do banco de dados, ele deve ser compatível com as regras para nomes de arquivos do sistema operacional. Felizmente, para o usuário, a maioria dos sistemas operacionais tem regras compatíveis para os nomes de arquivos.

Os nomes de arquivos podem ter *oito* caracteres ou menos. Devem ser iniciados por uma letra e não podem ter espaços em branco no meio.

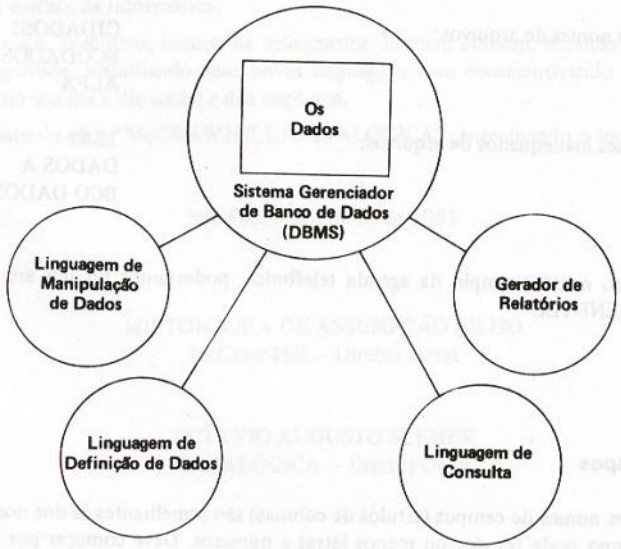


Figura 1. Esquema do sistema gerenciador de banco de dados (DBMS)

Nome do Aluno	Sala	Classe
Ângela, Aguiar	13	1
Borges, Edson	26	1
Cintra, Carlos	15	2
Juliano, Tomás	20	3

Nome do Professor	Sala	Classe
Bonfim, Walter	25	1
Neves, Maria	16	2
Prado, Plínio	13	1
Wilson, Jorge	20	3
"	"	"

Figura 2. Exemplo de banco de dados relacional

Bons nomes de arquivos:

CJDADOS1
BCODADOS
ALFA

Nomes inadequados de arquivos:

12A1
DADOS A
BCO DADOS

Voltando ao nosso exemplo da agenda telefônica, poderíamos ter um arquivo desse tipo sob o nome "AGENDTEL".

Nome de Campos

As regras para os nomes de campos (títulos de colunas) são semelhantes às dos nomes de arquivos. O nome de campo pode ter *dez* ou menos letras e números. Deve começar por uma letra e não pode ter intercalados espaços em branco. Em nosso exemplo da agenda telefônica, os nomes de campos seriam o "nome", "endereço" e "telefone".

Tipos de Campos

O tipo de campo determina o tipo de dados que podem ser armazenados em um determinado campo. Há três tipos de campos: de caracteres, numéricos e lógicos.

Os campos de caracteres podem conter tudo que for possível digitar: letras, números, espaços em branco e símbolos. Um número telefônico poderá ser um campo de caracteres já que com eles não serão efetuados cálculos.

Os campos numéricos podem conter números, um ponto decimal e um sinal de subtração. Os números podem ser inteiros ou decimais. Os campos numéricos são usados quando há cálculos aritméticos a serem efetuados.

Os campos lógicos podem conter T e F ou Y e N (Verdadeiro ou Falso, Sim ou Não). São usados nos campos onde o valor armazenado pode ser apenas uma entre duas possibilidades. Por exemplo, com o arquivo AGENDTEL, podemos estabelecer um campo denominado "atualizado". Então, se perguntarmos ao computador se ele está atualizado, poderemos recuperar os dados desse campo como Y ou N (atualizado ou não-atualizado), dependendo do que contiver aquele registro.

Tamanho de Campo

O tamanho de campo é simplesmente o número de espaços equivalentes aos de uma máquina de escrever em uma coluna. No caso de um campo lógico, ele é sempre um. Nos demais, a determinação do tamanho do campo depende do que o usuário calcular como o tamanho máximo necessário para o campo em questão.

Vocabulário de Comandos Básicos

Uma das vantagens do dBASE II é que ele pode ser usado com eficiência sem que o usuário realmente conheça muito a seu respeito. O dBASE II é um DBMS orientado por comandos. Isso significa que o usuário deverá informar-lhe que tarefa deseja que ele cumpra. Ele poderá operar o arquivo interativamente, por meio do teclado – executando todas as funções básicas do banco de dados – com o uso de apenas um vocabulário bastante limitado: o vocabulário de comandos básicos consiste em apenas 20 palavras.

CREATE	SORT	DELETE
USE	INDEX	RECALL
APPEND	LOCATE	PACK
DISPLAY	FIND	REPORT
LIST	EDIT	QUIT
SUM	COUNT	COPY
FOR	REPLACE	

Essas palavras-chave fazem exatamente o que cada uma delas sugere. Para que você perceba melhor como são usadas e o que realmente fazem, nós criaremos um banco de dados com os

dados apresentados na Figura 3. Esse banco de dados, que serve como exemplo, contém somente cinco registros, mas é bem apropriado para demonstrar o uso do vocabulário básico. As operações do banco de dados são sempre as mesmas, seja com cinco registros, seja com 50.000.

NOME PECA	QTIDADE	ESTOQ MIN	PRECO UNIT
MARTELO,DE ORELHA 3/4"	6	6	10880.00
CHAVE DE PARAFUSO,CAIXA 5/8"	4	6	4290.00
CHAVE DE PARAFUSO,CAIXA 3/4"	8	5	6190.00
MARTELO,DE BOLA 1 1/2"	5	3	8150.00
ALICATE,DE FIO 3/4"	11	5	3890.00

Figura 3. Exemplo de banco de dados

O Sistema Operacional

O sistema gerenciador de banco de dados de seu micro é um programa que opera sob o controle do sistema operacional do computador. Ele segue as regras determinadas pelo sistema operacional, como a estrutura dos nomes dos bancos de dados (nomes de arquivos). Também opera em conjunto com o sistema operacional e, na maioria dos casos, isola o usuário das peculiaridades de um determinado sistema.

O dBASE II trabalha com os seguintes sistemas operacionais de micros:

CP/M®
CP/M-86™
PC DOS™
MS DOS™
TRSDOS™
CONCURRENT CP/M

Na maior parte do tempo, ao usar o dBASE II, o usuário não perceberá as diferenças entre esses sistemas operacionais, entretanto, há algumas delas que deverão ser conhecidas. Sempre que tivermos uma área em que as diferenças entre os sistemas operacionais afetem o dBASE II, elas serão indicadas neste manual.

O Uso do dBASE II

Em todos os sistemas operacionais acima, o indicador aparecerá no lado esquerdo de seu terminal de vídeo, da seguinte forma:

A>

Para usar o dBASE II, primeiramente digite os caracteres DBASE, após o indicador do sistema operacional.

A > DBASE

Então pressione < RETURN > (ou ENTER etc., dependendo de seu computador). Com isso, o programa DBASE.COM é carregado na memória do computador. Após o dBASE II estar carregado, a tela exibirá a mensagem:

DIGITAR A DATA OU TECLAR RETURN
■ PARA NENHUMA
(MM/DD/AA) :

Depois de fornecida a data, a tela exibirá (na versão 2.4, por exemplo):

*** dBASE II Ver 2.43 26 Mar 1985

O "ponto" abaixo dos asteriscos é chamado indicador de ponto. Esse é o modo de o dBASE II informar que está pronto para receber comandos.

Para que o dBASE II faça algo de útil é necessário primeiramente um banco de dados. Para criar um banco de dados com o dBASE II precisamos digitar a palavra CREATE após o indicador de ponto; o computador responderá solicitando um nome de arquivo. Na Tela 1, mostramos o diálogo com o dBASE II que define o nosso banco de dados. O "B:" antes do nome do arquivo EXEMPLO indica ao dBASE que o arquivo será armazenado na unidade de discos B.

Primeiramente é necessário determinar a estrutura do arquivo. Cada um dos campos do banco de dados é definido digitando-se o nome, o tipo e o tamanho de campo – cada um deles separado por vírgulas e sem espaços. O tipo de campo é designado por um C de Caractere, N, de Numérico, e L, de Lógico (Observe que digitamos uma casa decimal apenas no Campo 4, PREÇO). O dBASE II considera como não havendo decimais, a menos que haja instruções em contrário.

Indicamos que está terminada a definição dos campos, teclando < RETURN > ao recebermos uma mensagem do sistema para inserir um campo. Nesse caso, teclamos < RETURN >, quando o sistema solicitou a definição do Campo 5.

No exemplo, indicamos que vamos fornecer os dados imediatamente. Assim que teclarmos Y em resposta a INPUT DATA NOW? (INSERIR OS DADOS AGORA?), a tela ficará limpa e aparecerá o display da Tela 2.

O cursor, que indicaremos com um "^", é posicionado para que o usuário insira o nome da peça. Para fornecer os dados, basta apenas digitá-los conforme desejado. O cursor se moverá somente nas áreas da tela delimitadas pelas vírgulas, que são as larguras que o usuário delimitou ao estabelecer o arquivo. Se for pressionado < RETURN > em vez de se inserir o nome da peça, o processo de inserção de dados será interrompido e o programa retornará ao nível do comando dBASE II (i.é, um indicador de ponto). Se cometer um erro ao inserir os dados, o usuário poderá deslocar o cursor na tela, utilizando a tecla Control, juntamente com as teclas

```
. CREATE
Entre o nome do arquivo: B:EXEMPLO
Entre a estrutura do registro na forma seguinte:
Campo,   Nome, Tipo, Tamanho, Casas Decimais
 001     NOMEPECA,C,30
 002     QTIDADE,N,3
 003     ESTOQMIN,N,3
 004     PRECO,N,10,2
 005
Deseja comecar entrada de dados agora?
```

Tela 1. Exemplo da criação de um banco de dados

E, S, D e X. A tecla Control é usada como a tecla Shift. Se for pressionada juntamente com a tecla E, o cursor se moverá um campo para cima na tela. Control X fará o cursor mover-se um campo para baixo na tela. Control S fará o cursor mover-se um caractere para a esquerda e Control D fará o cursor mover-se um caractere para a direita.

Quando o processo de inserção de dados associado a CREATE se encerra, o arquivo do banco de dados é fechado. Nesse ponto, não está mais sendo usado nenhum banco de dados. Para usar um banco de dados já existente, é necessário digitar USE seguido pela identificação da unidade de discos de um nome de arquivo.

```
USE B: EXEMPLO
```

O comando USE encerra qualquer arquivo que esteja sendo usado e abre aquele que for nomeado. Neste caso, foi aberto o arquivo EXEMPLO, localizado na unidade de disco B.

Para acrescentar registros a um banco de dados já existente, usa-se o comando APPEND.

```
APPEND
```

O comando APPEND apresenta uma tela idêntica à que mostramos na Tela 2, exceto pelo fato de que o número do registro será o daquele que está sendo acrescentado.

Os comandos DISPLAY e LIST permitem examinar os registros no banco de dados. LIST apresenta todos os registros no banco de dados que está sendo usado. Isso é demonstrado

na Tela 3. Os números da coluna da extrema esquerda são os números de registro no banco de dados. O comando DISPLAY ALL produz um resultado semelhante ao de LIST, exceto que a apresentação na tela se interrompe a cada 15 registros. O comando DISPLAY exibe o registro atual. Neste caso, o banco de dados é posicionado no último registro devido ao uso de LIST. A Tela 4 ilustra o comando DISPLAY.

LIST, DISPLAY e muitos dos outros comandos podem ser melhorados ou tornar-se mais específicos, como mostra a Tela 5. Nos dois exemplos apresentados nessa tela, ela foi limitada aos registros que satisfizeram uma condição específica. No primeiro, examinamos somente os registros em que o conteúdo do campo NOMEPEÇA começa por MARTELO. Observe que MARTELO vem entre apóstrofos. O apóstrofo, neste caso, é chamado DELIMITADOR. No segundo exemplo, estamos comparando os conteúdos dos dois campos QTIDADE E ESTOQMIN. A tela mostra somente o registro em que o conteúdo do campo QTIDADE é menor que o do campo ESTOQMIN.

Podemos também apresentar os campos escolhidos nomeando os que serão exibidos. Essa técnica pode ser combinada com o último exemplo; ver Tela 6.

Observe que, nesse exemplo, apresentamos inclusive os resultados de uma operação efetuada em dois campos. Multiplicamos (*) o conteúdo do campo QTIDADE pelo conteúdo do campo PREÇO e apresentamos o resultado na coluna da direita.

```
RECORD # 00001
NOMEPECA  :
QTIDADE   :   :
ESTOQMIN  :   :
PRECO     :         :
```

Tela 2. Exemplo de uma tela normal de entrada de dados

```
. LIST
00001   MARTELO,DE ORELHA 3/4"          6   6   10880.00
00002   CHAVE DE PARAFUSO,CAIXA 5/8"    4   6    4290.00
00003   CHAVE DE PARAFUSO,CAIXA 1/4"    8   5    6190.00
00004   MARTELO,DE BOLA 1 1/2"          5   3    8150.00
00005   ALICATES,DE FIO 3/4"           11   5    3890.00
```

Tela 3. Listagem do conteúdo do exemplo de banco de dados

```
. DISPLAY
00005   ALICATES,DE FIO 3/4"           11   5    3890.00
.
```

Tela 4. Exemplo do comando DISPLAY


```
. DISPLAY FOR NOMEPECA="MARTELO"
00001   MARTELO,DE ORELHA 3/4"          6   6   10880.00
00004   MARTELO,DE BOLA 1 1/2"          5   3    8150.00
. DISPLAY FOR QTIDADE < ESTOQMIN
00002   CHAVE DE PARAFUSO,CAIXA 5/8"    4   6    4290.00
.
```

Tela 5. Exemplo do comando DISPLAY FOR

```
. DISPLAY ALL NOMEPECA,QTIDADE,PRECO,QTIDADE*PRECO
00001   MARTELO,DE ORELHA 3/4"          6   10880.00    65280.00
00002   CHAVE DE PARAFUSO,CAIXA 5/8"    4    4290.00    17160.00
00003   CHAVE DE PARAFUSO,CAIXA 1/4"    8    6190.00    49520.00
00004   MARTELO,DE BOLA 1 1/2"          5    8150.00    40750.00
00005   ALICATES,DE FIO 3/4"           11    3890.00    42790.00
.
```

Tela 6. Exemplo de uma tela de campo seletivo

Observe também que os registros estão dispostos na ordem em que foram digitados. Para reorganizá-los em um arquivo de banco de dados numa determinada ordem – quer alfabética quer numérica – pode-se utilizar os comandos SORT e INDEX.

A Tela 7 mostra um bom exemplo de classificação (SORT) em ordem alfabética no campo NOMEPEÇA. A classificação cria um novo arquivo de banco de dados, que neste caso denominamos INVENTAR e que está localizado no disco B. Observe que os itens estão agora em ordem alfabética pelo nome da peça e que os números de registros para cada peça são diferentes do que eram em nosso banco de dados original, EXEMPLO.

A Tela 8 mostra um método alternativo para classificar os dados. Essa técnica é chamada INDEXAÇÃO. A indexação cria um novo arquivo chamado arquivo de ÍNDICE (INDEX), porém ela mantém uma relação entre os itens em que estamos interessados e o número do registro. Observe que nesse exemplo os nomes de peças também aparecem em ordem alfabética e, mesmo assim, os números de registros são os mesmos do arquivo original, EXEMPLO.

Se ao trabalhar com um arquivo, o usuário não puder se lembrar do nome de um campo, ele poderá usar o comando DISPLAY STRUCTURE, como mostra a Tela 9, para obter os nomes de campo no banco de dados. Observe que DISPLAY STRUCTURE também apresenta o número de registros, a última vez em que o banco de dados foi modificado (se foi digitada a data como solicitado ou se o usuário tem um IBM PC) e o número total de bytes em um registro. Note que o número total de bytes pode ser um a mais que a soma de bytes para o campo, a fim de permitir a adição do asterisco de exclusão de dados. Isso pode ser importante quando existe um grande número de registros ou quando se está copiando registros em um outro disco.

O comando SUM acrescentará o conteúdo de um campo numérico ou uma expressão que envolva campos, como mostra a Tela 10. Ele também pode ser usado em uma condição do tipo SUM FOR NOMEPEÇA = "CHAVE DE PARAFUSO".

O comando COUNT contará o número de registros que atendem a uma condição específica. Isso é ilustrado pela Tela 11.

O comando COPY copiará todo ou parte de um banco de dados em um outro. Isso é ilustrado pela Tela 12. Esse comando é importante quando se quer refazer a estrutura de um campo sem perder qualquer dos registros do arquivo.

O comando REPORT é um método eficiente de preparar relatórios formatados formais com os conteúdos do banco de dados. A Figura 4 mostra um exemplo desse comando. O relatório é preparado respondendo-se a uma série de perguntas simples. A Tela 13 mostra a preparação do relatório. Um relatório pode ser direcionado a uma impressora simplesmente, digitando-se REPORT TO PRINT. O comando REPORT será explicado com mais detalhes na Parte III.

Para editar um registro, o usuário deve conhecer o seu número, ou então, já estar posicionado nesse registro. Pode-se determinar o número do registro com o uso do comando DISPLAY ou então dos comandos LOCATE ou FIND. O comando FIND só pode ser usado com um banco de dados indexado. A Tela 14 ilustra esses dois comandos.

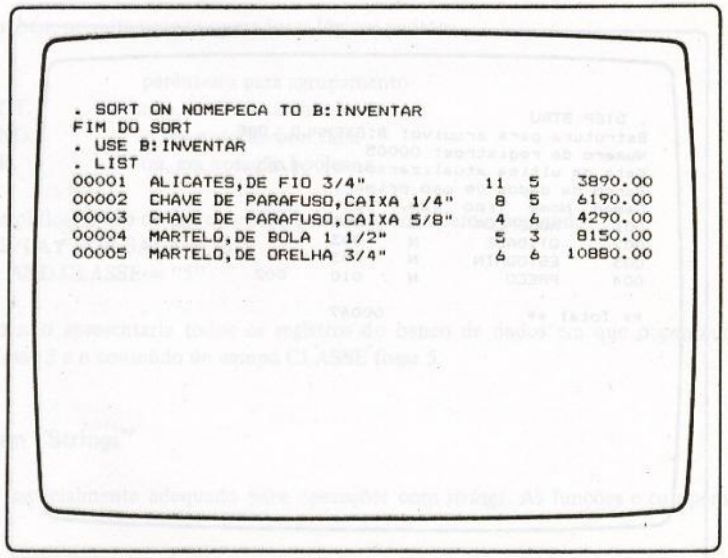
Observe que o comando LOCATE forneceu o número do registro desejado. O banco de dados é então posicionado no Registro 5. Se o registro não fosse localizado, a resposta do computador seria END OF FILE ENCOUNTERED (FINAL DO ARQUIVO ENCONTRADO).

O comando FIND forneceu um simples indicador de ponto. Isso indica que o registro foi encontrado e que o banco de dados está adequadamente posicionado. Se o registro não tivesse sido encontrado, a resposta teria sido NO FIND (NÃO ENCONTRADO). Em ambos os casos, podemos editar o registro por meio do comando EDIT A, do dBASE II, que permite a edição do registro atual. Se souber o número do registro, o usuário poderá simplesmente digitar EDIT seguido pelo número. EDIT 3, por exemplo, fará com que o dBASE II edite o Registro 3.

O comando EDIT apresenta uma tela idêntica à do comando APPEND. O cursor pode ser deslocado na tela por meio das teclas Control, como descrito anteriormente. Além disso, pode-se retroceder um registro por vez com o uso de Control R ou pode-se avançar, também um registro por vez, usando-se Control C. Control W encerra o processo de edição.

Um outro comando de nosso vocabulário básico, que modificará os conteúdos do banco de dados que usamos como exemplo, é REPLACE. Esse comando pode modificar o conteúdo de um registro individual, de todos os registros, ou de alguns que obedecem a uma determinada condição. Para ilustrar isso, suponhamos que seja necessário aumentar o preço unitário (PREÇO) das chaves de parágrafo em 10%. O processo necessário para isso é exemplificado pela Tela 15.

A exclusão de registros é um processo em dois estágios. O comando DELETE marca um registro para exclusão, o comando PACK apaga efetivamente esse registro. Se mudar de idéia, o usuário poderá "desmarcar" o registro por meio do comando RECALL. A Tela 16 demonstra esse procedimento.



```
. SORT ON NOMEPECA TO B:INVENTAR
FIM DO SORT
. USE B:INVENTAR
. LIST
00001  ALICATES,DE FIO 3/4"              11  5   3890.00
00002  CHAVE DE PARAFUSO,CAIXA 1/4"      8  5   6190.00
00003  CHAVE DE PARAFUSO,CAIXA 5/8"      4  6   4290.00
00004  MARTELO,DE BOLA 1 1/2"            5  3   8150.00
00005  MARTELO,DE ORELHA 3/4"            6  6  10880.00
```

Tela 7. Exemplo de classificação de um campo – NOMEPEÇA

```
. USE B:EXEMPLO
. INDEX ON NOMEPECA TO INVENTAR
00005
 registros indexados
. LIST
00005  ALICATES,DE FIO 3/4"          11  5   3890.00
00003  CHAVE DE PARAFUSO,CAIXA 1/4"  8  5   6190.00
00002  CHAVE DE PARAFUSO,CAIXA 5/8"  4  6   4290.00
00004  MARTELO,DE BOLA 1 1/2"        5  3   8150.00
00001  MARTELO,DE ORELHA 3/4"        6  6  10880.00
```

Tela 8. Exemplo de um banco de dados indexado

```
. DISP STRU
Estrutura para arquivo: B:EXEMPLO .DBF
Numero de registros: 00005
Data da ultima atualizacao: 01/01/80
Banco de dados de uso primario
Campo  Nome  Tipo  Tam.  Dec.
001     NOMEPECA     C    030
002     QTIDADE      N    003
003     ESTOQMIN     N    003
004     PRECO        N    010    002

** Total **                00047
.
```

Tela 9. Exemplo de DISPLAY STRUCTURE


```
. SUM QTIDADE*PRECO
 215500.00
. SUM QTIDADE
  34
.
```

Tela 10. Exemplo do comando SUM

```
. COUNT FOR NOMEPECA="CHAVE DE PARAFUSO"
Contagem = 00002
.
```

Tela 11. Exemplo do comando COUNT

```
. COPY FIELDS NOMEPECA,PRECO TO ARQTEMP
00005
 registros copiados
.
```

Tela 12. Exemplo do comando COPY

```
Pag. No. 00001
     01/01/80

                    RELATORIO DO BANCO DE DADOS EXEMPLO

          NOME DAS PECAS          QTI ESTOQ PRECO UNIT VALORESTOQ
                                  DAD  MIN
                                   E

     ALICATES,DE FIO 3/4"          11     5    3890.00   42790.00
     CHAVE DE PARAFUSO,CAIXA 1/4"   8     5    6190.00   49520.00
     CHAVE DE PARAFUSO,CAIXA 5/8"   4     6    4290.00   17160.00
     MARTELO,DE BOLA 1 1/2"         5     3    8150.00   40750.00
     MARTELO,DE ORELHA 3/4"         6     6   10880.00   65280.00

** Total **                        34                    215500.00
.
.
.
```

Figura 4. Exemplo de um relatório do dBASE II


```
. REPORT
Entre o nome do relatorio: B:EXEMPLO
Entre opcoes: m=margem esq., l=linhas/pag., w=tamanho pag.
Cabecalho? (sim=y/nao=n)Y
Entre o cabecalho da pagina: RELATORIO DO BANCO DE DADOS EXEMPLO
Relatorio com espaco duplo? (sim=y/nao=n)N
Deseja totais? (sim=y/no=n)Y
Deseja subtotais no relatorio? (sim=y/nao=n)N
Col   Tamanho, Conteudo
001     30,NOMEPECA
Entre o cabecalho: NOME DAS PECAS
002     3,QTIDADE
Entre o cabecalho: QTIDADE
Deseja totais? (sim=y/no=n)Y
003     5,ESTOQMIN
Entre o cabecalho: ESTOQ MIN
Deseja totais? (sim=y/no=n)N
004     10,PRECO
Entre o cabecalho: PRECO UNIT
Deseja totais? (sim=y/no=n)N
005     10,QTIDADE*PRECO
Entre o cabecalho: VALORESTOQ
Deseja totais? (sim=y/no=n)Y
006
```

Tela 13. Exemplo da preparação do relatório

```
. LOCATE FOR NOMEPECA="ALICATE"
Registro: 00005
. FIND ALICATE
.
```

Tela 14. Exemplo da localização de um registro

Ao terminar de usar o dBASE II, o usuário, digitando QUIT após o indicador de ponto, fará com que o programa encerre todos os arquivos abertos e retorne ao sistema operacional, como mostra a Tela 17.

dBASE II como uma Linguagem de Programação

Como vimos, nesse rápido panorama do vocabulário básico de dBASE II, os comandos individuais são simples e eficientes. O termo técnico para esses comandos é *Não-processual*. As linguagens convencionais de programação, como BASIC, PL/1, FORTRAN e COBOL são chamadas *processuais*; isto é, o usuário tem de fornecer a seqüência de instruções (um programa) para obter os resultados que aqui obtivemos, usando comandos simples de uma linha.

dBASE II tem características tanto processuais como não-processuais. É uma linguagem estruturada semelhante, em alguns aspectos, às linguagens PL/1 e Pascal. Por outro lado, também possui muitas das características "orientadoras" da linguagem BASIC. É um intérprete, no sentido de que podemos montar partes ou seções de nosso programa por meio do teclado para testar nossa metodologia. Essa capacidade de admitir comandos pelo teclado torna o dBASE II especialmente fácil de aprender.

Alguns especialistas calculam que entre 50% e 80% dos programas tradicionais são dedicados às tarefas de armazenar e gerenciar dados, preparar relatórios e manipular telas. Em dBASE II, grande parte dessas tarefas já estão prontas, o que reduz a tarefa de programá-lo a apenas 20%. Ele também fornece uma grande variedade de instrumentos para se usar durante a programação.

Quais são, então, as desvantagens? Há duas. Primeiramente, o dBASE II não lida com matemática de alto nível, somente com matemática financeira (pela versão 2.4). Se as funções mais sofisticadas são importantes para você, talvez o dBASE II não possa corresponder as suas expectativas.

A segunda desvantagem está no fato de que o dBASE II é um programa lento, se comparado ao FORTRAN e ao PL/1. Isso é parte da troca que se tem de fazer: troca-se a velocidade pela capacidade e facilidade de uso. A velocidade é de extrema importância no mundo tradicional e vasto dos computadores de grande porte. No mundo do micro, a velocidade reduzida pode ser às vezes irritante, mas é de pouca importância. Além disso, ela é relativa. A soma de dois números pode levar 7 milissegundos no dBASE II. De acordo com muitos padrões, isso é demasiado lento. Por outro lado, para o usuário do computador, parece virtualmente instantâneo.

Tudo somado, para o principiante na área de computação, dBASE II é uma excelente linguagem. Grande parte do trabalho mecânico da programação já foi realizada: a parte da linguagem controlada pelo DBMS controla os arquivos de gerenciamento de dados, a manipulação da tela é simples e direta e o gerador de relatórios produzirá grande parte da saída impressa necessária — assim, pode-se concentrar em aprender aqueles procedimentos que possibilitarão a produção de aplicações especializadas. O usuário pode aprender um pouco por vez e ainda obter um resultado significativamente útil, a qualquer estágio do aprendizado. Mais cedo ou mais tarde, aprenderá a utilizar os instrumentos de dBASE II para fazer com que o computador realmente trabalha para ele. Com a programação, o usuário obtém resultados mais rapidamente e com menor esforço.

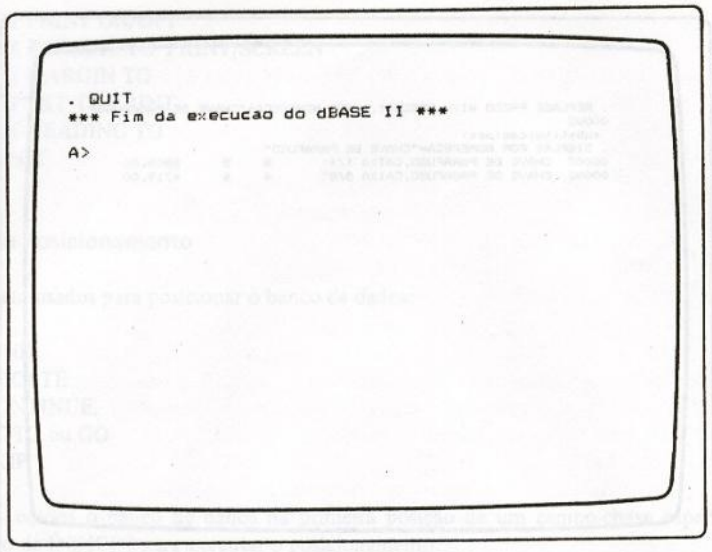

```
. REPLACE PRECO WITH PRECO*1.1 FOR NOMEPECA="CHAVE DE PARAFUSO"
00002
 substituicao(oes)
. DISPLAY FOR NOMEPECA="CHAVE DE PARAFUSO"
00003  CHAVE DE PARAFUSO,CAIXA 1/4"      8    5     6809.00
00002  CHAVE DE PARAFUSO,CAIXA 5/8"      4    6     4719.00
```

Tela 15. Exemplo do comando REPLACE

```
. DELETE FOR NOMEPECA="MARTELO"
00002
 eliminacao(oes)
. RECALL FOR NOMEPECA='MARTELO,DE ORELHA 3/4"'
00001
 recuperacao(oes)
. PACK
PACK (compactacao) terminado00004
 registros copiados

REINDEXANDO ARQUIVO    - B:INVENTAR.NDX
00004
 registros indexados
.
```

Tela 16. Exemplo de exclusão de registro



Tela 17. Saindo do dBASE II

Encaremos esse fato: a idéia toda está em fazer com que o computador realmente o ajude em sua contabilidade rotineira e não apenas conseguir programas bem feitos. O dBASE II pode aumentar sua produtividade, mesmo que você tenha pouca experiência na área. Como tempo é dinheiro, esse aumento na produtividade é dinheiro em seu bolso. Mesmo que seja necessária a recodificação em uma outra linguagem mais rápida, ainda assim você pode usar dBASE II como uma linguagem de estrutura. Quase todas as linguagens de estrutura são na verdade conferidores de sintaxe – dBASE II pode ser usado para testar os procedimentos lógicos. A estrutura e a verificação são os elementos da programação que mais consomem tempo. A codificação (especialmente a recodificação) tem conseqüências relativamente insignificantes.

No Capítulo 2, vamos examinar a programação para o principiante.

Um Resumo da História do dBASE II

A história do dBASE II leva-nos até meados da década de 60 e a um sistema gerenciador de informações chamado RETRIEVE, que foi comercializado pela Tymshare Corporation. O Jet Propulsion Laboratory, em Pasadena, Califórnia, usou o RETRIEVE até o final da década de 60 quando adquiriu os computadores UNIVAC 1108. Jeb Long, um novo programador no JPL, recebeu a incumbência de escrever um programa que pudesse executar as mesmas funções que o RETRIEVE. O novo sistema gerenciador de arquivos, que Long desenvolveu, chamou JPLDIS. Esse sistema

continuou a evoluir nos vários anos seguintes e ainda é usado em muitos dos computadores do tipo UNIVAC 1100.

No final da década de 70, Wayne Ratliff, que trabalhava no JPL, sob contrato. com Martin Marietta, começou a se interessar por microcomputadores. Ratliff era um dos amigos de Jeb e começou a desenvolver um sistema que, para o usuário, era muito semelhante ao JPLDIS. Conta a história que Wayne desenvolveu a versão inicial de seu sistema de banco de dados com o objetivo de ganhar apostas de futebol! Wayne comercializou essa versão JPLDIS sob o nome de VULCAN.

Embora rapidamente se desenvolvesse até o ponto de adquirir seu próprio caráter e personalidade, o VULCAN não conseguiu muitas vendas: em 1980 ele possuía somente sessenta clientes. Então um empresário, George Tate, entrou em cena após ver um anúncio do VULCAN em uma publicação sobre computadores. Logo deu-lhe um novo nome – dBASE II – e uma nova companhia, a Ashton-Tate, que foi formada para comercializá-lo. O resto é história.

Capítulo Dois

COMO COMEÇAR A PROGRAMAR

O principiante em programação verificará que, na verdade, não é difícil.

Para se ter uma idéia, suponhamos que temos um banco de dados chamado ESCOLA, que consiste em informações sobre estudantes do primeiro grau. Neste capítulo, vamos examinar programas que contam estudantes. Se já estiver familiarizado com os fundamentos da programação com o dBASE II, deixe-o de lado e prossiga no Capítulo 3.

Naturalmente, um dos métodos de contar os estudantes, classe por classe, consiste em usar diretamente o teclado, como na Tela 1.

Outra alternativa seria escrever um programa em dBASE II simples, como o Programa 1.

Não há qualquer diferença prática entre digitar os comandos um por vez, usando o teclado, e digitá-los em um programa.

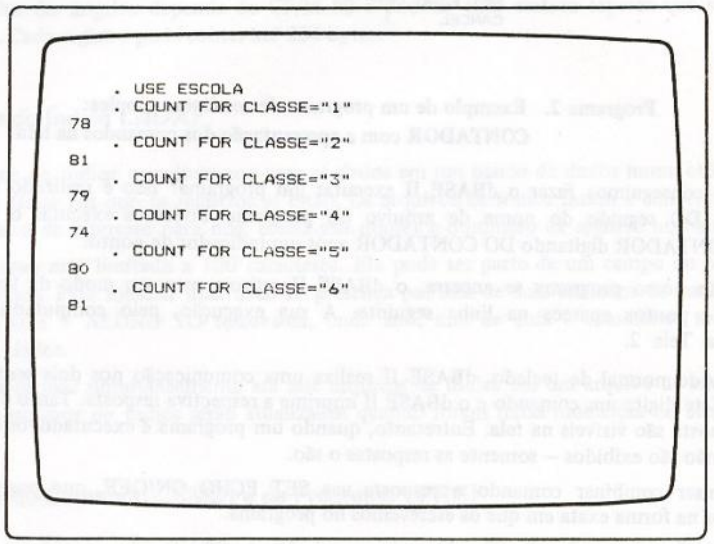
Os programas em dBASE II são chamados COMMAND FILES (ARQUIVOS DE COMANDO). Para escrever um programa, simplesmente digitam-se os comandos no computador, usando um programa editor de texto como o WordStar®, ou o comando do DBASE II, fornecido exatamente com esse objetivo – O MODIFY COMMAND (COMANDO MODIFY).

Quando se usa o MODIFY COMMAND, o computador solicita um nome para o programa o qual deve estar de acordo com as regras para nomes de arquivo. O MODIFY COMMAND acrescentará os caracteres .PRG (para os computadores do tipo IBM PC) ou .CMD (para computadores com o sistema CP/M), após o nome do arquivo. Esses caracteres indicam ao dBASE II que o arquivo é um programa.

Quando o nome do arquivo é digitado, a tela fica limpa e o cursor é posicionado em sua parte superior. Podemos então teclar os comandos, uma linha por vez, como mostra o Programa 1. CANCEL indica o fim do programa.

Ao terminar de digitar os comandos, tecle Control-W para gravar o arquivo no disco.

Outro modo de escrever programas consiste em usar um programa editor de texto. A maioria dos editores de texto tem um modo especial para escrever os programas. Em WordStar, esse modo é o "não-documento" – Opção N. Poderão ocorrer efeitos estranhos se você, acidentalmente, escrever um programa usando o modo de documento normal – Opção D. Suspeitando-se ter escrito um programa usando o modo documento, poder-se-á copiá-lo novamente sobre ele mesmo, usando PIP com a opção z. Se usarmos um outro editor de texto, deveremos consultar o manual do usuário para nos assegurar de que estamos usando o modo apropriado.



Tela 1. Exemplo do comando COUNT, do dBASE II

```
USE ESCOLA
COUNT FOR CLASSE="1"
COUNT FOR CLASSE="2"
COUNT FOR CLASSE="3"
COUNT FOR CLASSE="4"
COUNT FOR CLASSE="5"
COUNT FOR CLASSE="6"
CANCEL
```

Programa 1. Exemplo de um programa de contagem simples: CONTADOR

```
SET ECHO ON
USE ESCOLA
COUNT FOR CLASSE="1"
COUNT FOR CLASSE="2"
COUNT FOR CLASSE="3"
COUNT FOR CLASSE="4"
COUNT FOR CLASSE="5"
COUNT FOR CLASSE="6"
SET ECHO OFF
CANCEL
```

Programa 2. Exemplo de um programa de contagem simples: CONTADOR com a apresentação dos comandos na tela

Como conseguimos fazer o dBASE II executar um programa? Isso é realizado usando-se o comando DO seguido do nome de arquivo do programa. Podemos executar o programa chamado CONTADOR digitando DO CONTADOR após um indicador de ponto.

Quando nosso programa se encerra, o dBASE II retorna para seu modo de teclado e o indicador de pontos aparece na linha seguinte. A sua execução, pelo computador, teria a aparência da Tela 2.

No modo normal de teclado, dBASE II realiza uma comunicação nos dois sentidos com o usuário. Este digita um comando e o dBASE II imprime a respectiva resposta. Tanto o comando como a resposta são visíveis na tela. Entretanto, quando um programa é executado, os comandos geralmente não são exibidos – somente as respostas o são.

Se quiser combinar comando e resposta use SET ECHO ON/OFF, que possibilita ver os comandos na forma exata em que os escrevemos no programa.

O Programa 2 gerará a Tela 3.

Neste ponto, vamos deixar de lado nossos exemplos de desenvolvimento de programação e resumidamente discutir a sintaxe dos comandos dBASE II.

Estrutura dos Comandos do dBASE II

Os comandos devem ser fornecidos na sintaxe correta. Cada comando dBASE II inicia com um verbo (o comando básico). Muitos comandos também podem ter complementos para adaptá-los de modo a poderem atender a um objetivo específico. Por exemplo:

DISPLAY [scope] [< exp list >] [FOR < exp >]
- [OFF]

Essa é a sintaxe do comando DISPLAY. Os colchetes indicam que o item entre eles é opcional. Quando usado sem complementos, o comando DISPLAY exibe somente o registro atual do banco de dados.

O SCOPE (TAMANHO) é a abrangência do comando em questão. Se nada for especificado, o scope assumirá as especificações do registro atual. A sintaxe permitida para ele é ALL (para todo o banco de dados) ou então NEXT n (próximos n registros), onde n é um número. Se for usada a opção FOR, o scope assumirá sempre ALL, a menos que o usuário o restrinja com o uso de NEXT n.

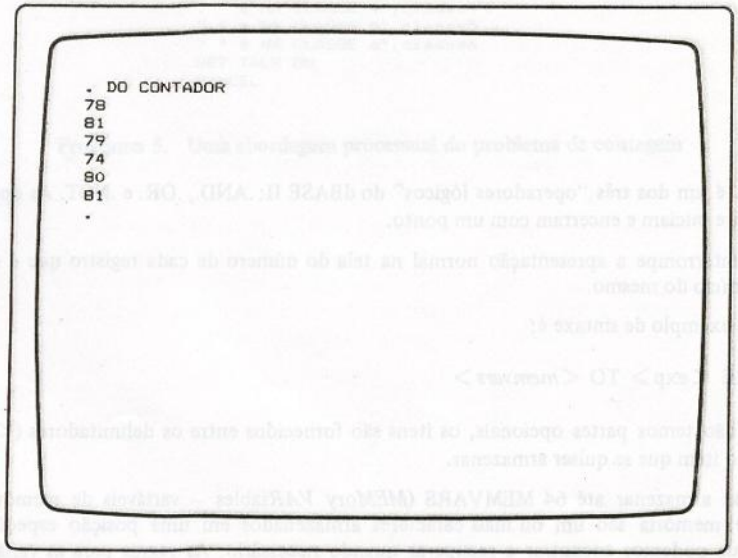
< exp list > possibilita especificar o que deve ser exibido. A cada comando que possibilita especificar, < exp list > assume tudo ou nada. Você pode especificar NOMES DE CAMPO, variáveis de memória (memvars), operações em campos e memvars e textos contidos por delimitadores. Um exemplo de uma < exp list > seria este:

NOME, SALA, CLASSE, NOTA*100,
- 'dados sobre o aluno'

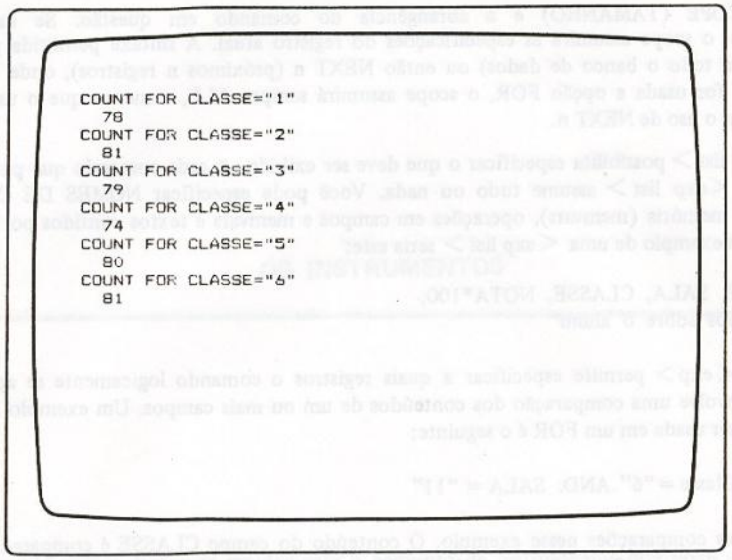
FOR < exp > permite especificar a quais registros o comando logicamente se aplica. A expressão envolve uma comparação dos conteúdos de um ou mais campos. Um exemplo de uma expressão a ser usada em um FOR é o seguinte:

FOR Classe = "6" .AND. SALA = "11"

Há duas comparações nesse exemplo. O conteúdo do campo CLASSE é comparado a "6" e o conteúdo do campo SALA é comparado a "11"



Tela 2. Execução do programa CONTADOR



Tela 3. Resposta modificada do programa de contagem

.AND. é um dos três "operadores lógicos" do dBASE II: .AND., .OR. e .NOT. As operações lógicas sempre iniciam e encerram com um ponto.

OFF interrompe a apresentação normal na tela do número de cada registro que é exibido sempre no início do mesmo.

Outro exemplo de sintaxe é:

STORE < exp > TO < *memvars* >

Aqui não temos partes opcionais, os itens são fornecidos entre os delimitadores (< >). A expressão é o item que se quiser armazenar.

Pode-se armazenar até 64 MEMVARS (*MEM*ory *VAR*iables – variáveis de memória). As variáveis de memória são um ou mais caracteres armazenados em uma posição específica da memória, que podemos encontrar e recuperar quando necessário. As regras para as variáveis de memória são as mesmas para os nomes de campo, elas podem ter até 254 bytes e podem ser de caracteres lógicos ou numéricos. Um exemplo dessa sintaxe é STORE 25 TO *número*.

O Uso das MEMVARS

O Programa 2 mostrou-nos como apresentar os comandos na tela com SET ECHO ON. Para obter o efeito contrário, e inibir a apresentação normal de respostas na tela pelo dBASE II, use SET TALK OFF. Assim, se quisermos uma apresentação mais elegante na tela ao contarmos nossos alunos, podemos usar variáveis de memória para armazenar temporariamente a resposta a cada comando de contagem. Fazendo isso, podemos apresentar na tela o conteúdo das variáveis juntamente com o texto que as descreve.

Usamos letras minúsculas para indicar as variáveis de memória de modo a distingui-las dos nomes de campo e de arquivo nos programas. No texto, as variáveis de memória aparecerão em *itálico*.

Quando TALK está desligado, os dados são apresentados somente com o uso de um dos comandos display. OFF que segue o comando DISPLAY impede que o dBASE II apresente o número do registro atual. Nesse programa, armazenamos os resultados de cada uma das operações de contagem em uma variável de memória, que então apresenta o texto entre os delimitadores. O Programa 3 daria origem à Tela 4.

Toda vez que contamos os alunos matriculados em uma classe tivemos de "ler" todo nosso banco de dados, o que somou seis leituras – uma para cada classe. Se o banco de dados for grande, sua leitura poderá exigir algum tempo, dependendo também do sistema operacional que está sendo usado.

Suponhamos que nosso banco de dados tenha 160.000 bytes e que nosso sistema o leia a uma velocidade de 1.600 caracteres por segundo. A contagem dos alunos matriculados em uma classe levaria 100 segundos. A contagem dos alunos matriculados nas seis classes levaria dez minutos.

Considerar dez minutos como excessivamente demorado ou não, depende em grande parte da freqüência com que se deseja contar os alunos. Se for somente uma vez ou duas, aceitaremos os dez minutos. Mas se quisermos uma contagem mais freqüente das salas de aula, precisaremos de um resultado mais rápido.

Um modo de conseguir isso é não ler o banco de dados freqüentemente. Suponhamos que ele tenha sido indexado por classe. Nesse caso, podemos usar o Programa 4.

Esse programa produz o mesmo resultado que o Programa 3: Tela 4. Todos os registros para cada classe estão agrupados. O comando FIND 1 posiciona o banco de dados no primeiro registro pertencente à classe 1. Como todos os registros que pertencem a uma classe estão reunidos, podemos usar a expressão WHILE em vez da expressão FOR em um comando (WHILE pode ser usada com qualquer comando do dBASE II com o qual se usa FOR exceto com LOCATE). WHILE inicia com o registro atual e processa somente enquanto o conteúdo do banco de dados atende essa condição. Com esse tipo de contagem, processamos o registro de cada aluno uma vez em vez de seis, ou cerca de seis vezes mais rápido que os programas anteriores.

```
SET TALK OFF
USE ESCOLA
COUNT FOR CLASSE="1" TO classe1
COUNT FOR CLASSE="2" TO classe2
COUNT FOR CLASSE="3" TO classe3
COUNT FOR CLASSE="4" TO classe4
COUNT FOR CLASSE="5" TO classe5
COUNT FOR CLASSE="6" TO classe6
DISPLAY OFF "# NA CLASSE 1",classe1
DISPLAY OFF "# NA CLASSE 2",classe2
DISPLAY OFF "# NA CLASSE 3",classe3
DISPLAY OFF "# NA CLASSE 4",classe4
DISPLAY OFF "# NA CLASSE 5",classe5
DISPLAY OFF "# NA CLASSE 6",classe6
SET TALK OFF
CANCEL
```

Programa 3. Exemplo de um programa simples de contagem

Quando usamos um arquivo indexado, como no Programa 4, cada operação é mais lenta que no caso de um arquivo não indexado porque acessamos os registros em uma seqüência "lógica". Essa seqüência é diferente da disposição física dos registros no disco, que necessitam mais movimento da cabeça no disco.

```
SET TALK OFF
USE ESCOLA INDEX CLASSE
FIND 1
COUNT FOR CLASSE="1" TO classe1
COUNT FOR CLASSE="2" TO classe2
COUNT FOR CLASSE="3" TO classe3
COUNT FOR CLASSE="4" TO classe4
COUNT FOR CLASSE="5" TO classe5
COUNT FOR CLASSE="6" TO classe6
DISPLAY OFF "# NA CLASSE 1",classe1
DISPLAY OFF "# NA CLASSE 2",classe2
DISPLAY OFF "# NA CLASSE 3",classe3
DISPLAY OFF "# NA CLASSE 4",classe4
DISPLAY OFF "# NA CLASSE 5",classe5
DISPLAY OFF "# NA CLASSE 6",classe6
SET TALK OFF
CANCEL
```

Programa 4. Contagem a partir de um arquivo de banco de dados indexado


```
DO CONTADOR
# NA CLASSE 1     78
# NA CLASSE 2     81
# NA CLASSE 3     79
# NA CLASSE 4     74
# NA CLASSE 5     80
# NA CLASSE 6     81
```

Tela 4. Exemplo do comando COUNT, do dBASE II

Há dois tipos básicos de acesso ao banco de dados no dBASE II: seqüencial e direto. No *acesso seqüencial*, folheamos o banco de dados, um registro por vez. A maioria dos comandos de alto nível no dBASE II (LIST, COUNT, SUM etc.) opera de modo seqüencial. O *acesso direto* possibilita ir diretamente a qualquer registro – desde que saibamos a qual registro queremos ir.

Nossos primeiros programas neste capítulo foram não-processuais; simplesmente informamos ao computador o que queríamos e ele mostrou como executá-lo. À medida que avançamos, mais nos familiarizamos com as operações realizadas, tornando nossos programas muito mais processuais.

Nosso exemplo seguinte de contagem de crianças será completamente processual. Os instrumentos básicos são DO WHILE/ENDDO e IFF/ENDIF. Embora a maior parte da programação no dBASE II use uma combinação de elementos processuais e não-processuais, o Programa é inteiramente processual. Para realmente nos adaptarmos à situação que queremos, quase sempre temos de ser processuais.

Quando usamos as estruturas DO WHILE/ENDDO e IF/ENDIF, os comandos que estiverem dentro de cada estrutura serão executados sempre que as expressões que seguirem DO WHILE e IF forem verdadeiras. Observe que o programa não requer qualquer ordem especial dos registros no banco de dados. O resultado final do Programa 5 é idêntico ao dos Programas 3 e 4 (Tela 4). O Programa 5 é também substancialmente mais rápido que o Programa 3.

O Programa 5 examina todos os registros no banco de dados, um por vez. Dependendo do conteúdo do campo CLASSE, o conteúdo de uma das seis variáveis de memória é aumentado em

uma unidade. Avançamos um registro (seqüencialmente) e repetimos o processo até chegarmos ao final do banco de dados. Após examinar todos os registros, apresentamos na tela os resultados.

Neste programa precisamos inicializar nossas variáveis *classe 1* até *classe 6*. Não é possível acrescentar 1 a *classe 1* antes que *classe 1* exista. Para existir, a variável tem que ter um valor.

A expressão usada com DO WHILE é .NOT.EOF, onde EOF é uma função do dBASE que significa final do arquivo (END of File – final do arquivo).

A expressão usada com IF compara o conteúdo do campo CLASSE à classe em questão. No contexto deste programa COUNT FOR CLASSE = "1" equivale aproximadamente a:

```
DO WHILE .NOT.EOF
   IF CLASSE = "1"
      STORE classe 1 + TO classe 1
   ENDIF
   SKIP
ENDDO
DISPLAY OFF classe 1
```

Nesse programa, também decidimos usar o comando ? em vez de DISPLAY OFF. O comando ? é usado para exibir na tela campos, variáveis de memória, expressões e texto; pode ser usado até mesmo quando não se está utilizando um banco de dados. Isso não acontece com o comando DISPLAY.

Escrever programas é uma questão de agrupar comandos para resolver um determinado problema. Após aprendermos os comandos, podemos facilmente agrupá-los para resolver problemas simples. Veremos mais adiante, neste livro, que geralmente damos uma solução aos problemas complexos desdobrando-os em vários problemas simples. As dificuldades encontradas na programação de soluções para problemas complexos geralmente têm pouco a ver com a complexidade da programação, mas sim, com nossa concepção do problema.

Muitas vezes haverá soluções de programação possíveis para um problema. A concepção escolhida é muitas vezes uma questão de preferência pessoal. Outros fatores que determinam a escolha de uma determinada concepção podem ser: a freqüência com que usaremos o programa, para quem ele é feito, e a necessidade de velocidade no programa.

Os programadores principiantes não precisam se preocupar em escrever programas "elegantes". Não há programa "melhor" que aquele que resolve um problema. Um dos melhores métodos de se aprender a programar é através de tentativa e erro.


```
SET TALK OFF
USE ESCOLA
STORE 0 TO classe1
STORE 0 TO classe2
STORE 0 TO classe3
STORE 0 TO classe4
STORE 0 TO classe5
STORE 0 TO classe6
DO WHILE .NOT. EOF
   IF CLASSE="1"
      STORE classe1 + 1 TO classe1
   ENDIF
   IF CLASSE="2"
      STORE classe2 + 1 TO classe2
   ENDIF
   IF CLASSE="3"
      STORE classe3 + 1 TO classe3
   ENDIF
   IF CLASSE="4"
      STORE classe4 + 1 TO classe4
   ENDIF
   IF CLASSE="5"
      STORE classe5 + 1 TO classe5
   ENDIF
   IF CLASSE="6"
      STORE classe6 + 1 TO classe6
   ENDIF
   SKIP
ENDDO
? " # NA CLASSE 1",classe1
? " # NA CLASSE 2",classe2
? " # NA CLASSE 3",classe3
? " # NA CLASSE 4",classe4
? " # NA CLASSE 5",classe5
? " # NA CLASSE 6",classe6
SET TALK ON
CANCEL
```

Programa 5. Uma abordagem processual do problema de contagem

Capítulo Três

OS INSTRUMENTOS

Neste capítulo vamos descrever rapidamente o conjunto de instrumentos que pode ser usado na programação com o dBASE II. Esse conjunto é extremamente útil para a maioria das aplicações financeiras devido à capacidade da linguagem do dBASE II de manipular caracteres de dados. Essa capacidade é o ponto forte do dBASE II como linguagem; seu ponto fraco é a incapacidade de lidar com matemática de alto nível. Felizmente, a maioria dos problemas financeiros pode ser solucionada por meio das quatro funções matemáticas básicas – adição, subtração, multiplicação e divisão.

A apresentação de cada um dos instrumentos será resumida. Não vamos tentar substituir o manual do dBASE II, que deverá ser consultado, no caso de se desejar mais informações com relação a qualquer um dos instrumentos.

O mais básico dos instrumentos são os arquivos de banco de dados e a linguagem de consulta. No Capítulo Um, apresentamos de modo sucinto os arquivos de banco de dados e parte da linguagem de consulta. A linguagem de consulta ou comando permite localizar e recuperar dados e conectar o sistema a programas aplicativos. É a linguagem que usaremos na programação. Além dos arquivos do banco de dados, o dBASE II usa vários tipos de arquivos em disco:

DATABASE	ARQUIVO (.DBF)
INDEX	ARQUIVO (.NDX)
COMMAND	ARQUIVO (.CMD)
PROGRAM	ARQUIVO (.PRG)
REPORTFORM	ARQUIVO (.FRM)
FORMAT	ARQUIVO (.FMT)
TEXT	ARQUIVO (.TXT)
MEMORY	ARQUIVO (.MEM)

OS ARQUIVOS

Arquivos de Bancos de Dados (.DBF)

Os bancos de dados consistem em um ou mais arquivos de banco de dados, que armazenam dados em registros e campos. Cada registro automaticamente recebe um número de identificação.

Os arquivos de bancos de dados podem ter até 65.535 registros ou 8.388.480 bytes. O limite de tamanho do arquivo depende do CP/M 80 e pode-se com certeza esperar que seja ampliado no futuro. Cada registro pode conter até 254 bytes.

Arquivos de Índice (.NDX)

Os arquivos de índice permitem-nos usar os dados em um banco de dados numa ordem diferente da ordem física em que os registros aí estão. Os arquivos de índice fazem a conversão entre uma chave (dados de interesse para nós, como um nome) e o número de registro no banco de dados.

A chave está limitada a 100 caracteres. Ela pode ser parte de um campo ou pode abranger vários campos. Para indexar uma lista de presença por sala de aula teríamos de usar: INDEX ON ANO + SALA + ALUNO TO saladeaula, onde ano, sala de aula e estudantes são campos do banco de dados.

Pode-se usar simultaneamente até sete arquivos de índice em um arquivo de banco de dados. Todos os arquivos de índice serão atualizados quando forem feitas mudanças no banco de dados.

Arquivos de Comando (.CMD) e de Programa (.PRG)

São ambos programas do dBASE. A diferença está no identificador de arquivo. Nas máquinas do tipo PC, o dBASE II identifica os programas com .PRG. Nos computadores que usam CP/M 80 ele os identifica com .CMD.

A criação e o uso desses arquivos são os objetivos essenciais deste livro.

Arquivos de Relatórios (.FRM)

Esses arquivos são criados com o comando REPORT. Contêm as informações necessárias para preparar relatórios. Os arquivos de relatórios podem ser mostrados por meio de um editor de texto ou pelo comando MODIFY COMMAND do dBASE II.

Se for usado o MODIFY COMMAND, deve-se tomar o cuidado de incluir o identificador de arquivo .FRM com o nome de arquivo. Se isso não for feito, o dBASE II considerará que o nome de arquivo se refere a um arquivo de comando.

MODIFY COMMAND < nome de arquivo > .FRM

Arquivos de Formatação (.FMT)

São arquivos que criam disposições especiais na tela para entrada de dados. Os arquivos de formatação podem ser criados com o MODIFY COMMAND. Se for usado MODIFY COMMAND será necessário incluir o identificador de arquivo .FMT com o nome de arquivo, para evitar confusão com um arquivo de comando.

MODIFY COMMAND < nome de arquivo > .FMT

Arquivos de Texto (.TXT)

São arquivos ASCII puros e contêm somente caracteres ASCII imprimíveis. Podem ser "lidos para" um banco de dados do dBASE II como forma especial do comando APPEND e criados a partir de um banco de dados por meio de uma forma especial do comando COPY.

Também podem ser usados para "registrar" o processamento do dBASE II, por meio do teclado no modo de consulta ou por meio dos programas do dBASE II.

Arquivos de Memória (.MEM)

São usados para gravar as variáveis de memória do dBASE II em um arquivo em disco. Os arquivos são criados pelo comando SAVE e então lidos novamente para a memória pelo comando RESTORE.

A LINGUAGEM

A linguagem de consulta é composta de comandos, funções e operadores. Os comandos são verbos, como COUNT (contar) e DISPLAY (exibir) que podem ser adaptados aos usos que se tiver em mente, como explicado no Capítulo Dois.

As funções são instrumentos que podem ser usados juntamente com os comandos. Por exemplo, a função EOF permite testar uma condição de fim de arquivo de um programa.

Os operadores possibilitam efetuar operações aritméticas e lógicas. O sinal + é um exemplo de operador aritmético e .AND. é exemplo de operador lógico.

Os comandos podem ser agrupados em uma série de categorias funcionais. Um comando pode pertencer a mais de uma dessas categorias:

CRIAÇÃO DE BANCO DE DADOS
USO DE BANCOS DE DADOS
MODIFICAÇÃO DE BANCOS DE DADOS
CLASSIFICAÇÃO E INDEXAÇÃO
OBTENÇÃO DE INFORMAÇÕES
APRESENTAÇÃO DE INFORMAÇÕES NA TELA

IMPRESSÃO
POSICIONAMENTO
COMANDOS DE ENTRADA DE DADOS
AUXÍLIO DE PROGRAMAÇÃO
VARIÁVEIS DE MEMÓRIA
OPERAÇÕES GERAIS DE ARQUIVO
COMANDOS DE FUNÇÕES ESPECIAIS
CONTROLES DE PARÂMETROS DO dBASE II

Criação de Banco de Dados

Os comandos abaixo podem ser usados para criar novos arquivos de bancos de dados.

CREATE COPY JOIN SORT TOTAL

Somente CREATE pode originar um banco de dados completamente novo. Os outros quatro comandos criam arquivos de bancos de dados baseados em bancos de dados já existentes.

Uso dos Bancos de Dados

Há somente dois comandos que possibilitam usar diretamente um arquivo de banco de dados:

USE e SELECT

O comando USE EXEMPLO, por exemplo, encerra o arquivo de banco de dados que está sendo usado e abre o arquivo EXEMPLO.

Podemos usar dois arquivos de banco de dados ao mesmo tempo. Esses arquivos são denominados primários (primary) e secundários (secondary).

```
SELECT PRIMARY
USE EXEMPLO1
SELECT SECONDARY
USE EXEMPLO2
```

Essa seqüência abre os arquivos EXEMPLO1 e EXEMPLO2 e os identifica como os arquivos primário e secundário. Ao usá-los, podemos passar de um para outro com SELECT PRIMARY e SELECT SECONDARY.

Modificação dos Bancos de Dados

Há dois modos de se modificar um banco de dados: podemos modificar a forma do arquivo ou o conteúdo de registros já existentes.

COMANDOS PARA MODIFICAR A FORMA

MODIFY STRUCTURE	DELETE
APPEND	RECALL
INSERT	PACK

MODIFY STRUCTURE – possibilita a redefinição de uma estrutura de arquivo já existente. Modifica os nomes, tipos e tamanhos de campo. Esse comando apaga todos os registros.

APPEND – acrescenta registros ao final do banco de dados.
INSERT – insere um registro dentro do banco de dados.
DELETE – marca um registro para exclusão.
RECALL – elimina a marca de exclusão de registro.
PACK – apaga todos os registros marcados para exclusão e renumera todos os registros que restarem.

COMANDOS PARA MODIFICAR O CONTEÚDO

BROWSE – edição em tela cheia de 19 registros por vez. Especialmente útil para atualização de colunas.
CHANGE – permite edição seletiva de registros por campo.
EDIT – edição em tela formatada de um único registro por vez.
READ – edição em tela cheia; usada com o comando GET para "editar" campos ou variáveis de memória.
REPLACE – substituição direta do conteúdo de um campo.

Classificação e Indexação

Os comandos usados para classificar e indexar arquivos de bancos de dados são:

SORT INDEX REINDEX

SORT ON <nomedecampo> cria uma cópia do banco de dados onde os registros do novo banco de dados ficam em ordem alfabética ou numérica, segundo o campo designado.

INDEX ON <expressão> cria um arquivo que relaciona os números de registros do dBASE II à expressão. O banco de dados usado com um arquivo de índice aparece organizado pela expressão. Os arquivos de índice (*index*) são usados juntamente com os arquivos de banco de dados. A indexação é mais rápida que a classificação.

REINDEX refaz a indexação dos arquivos de índice em uso.

Extraindo e Apresentando Informações

Os comandos abaixo são usados para extrair e apresentar dados na tela.

DISPLAY	SET FORMAT TO
LIST	READ
?	EDIT
??	BROWSE
@ linha, col SAY	SUM
USING	COUNT
@ linha, col GET	REPORT
PICTURE	SET HEADING TO

DISPLAY – apresenta todos os registros de um banco de dados ou parte deles.
LIST – usado para apresentar todos os registros em um banco de dados ou parte deles.
? – apresenta campos, variáveis de memória, expressões e texto. Usado sozinho apresentará uma linha em branco.
?? – é o mesmo que o ponto de interrogação simples, exceto que o sinal duplo não possibilita o retorno do carro da impressora ou o fornecimento de nova linha. É usado para apresentar um item na posição atual do cursor ou da cabeça de impressão.
@ linha,col SAY – exibe os itens que se iniciam na linha e coluna designadas. Pode-se exibir mais de um item por meio de SAY, concatenando-se os itens. Também pode ser usado com uma sentença USING. USING fornece uma máscara para ajudar a adaptar a tela.
@ linha,col GET – exibe um campo único ou variável de memória na linha e coluna designadas.
SET FORMAT – usado com os comandos EDIT, APPEND, INSERT e READ para criar uma tela por meio de um ARQUIVO DE FORMATAÇÃO.
READ – usado para chamar a tela de um ARQUIVO DE FORMATAÇÃO e para edição em tela cheia com o comando GET.
EDIT – apresentação em tela cheia de um único registro.
BROWSE – edição em tela cheia de 19 registros. Exibe a quantidade de cada registro que couber em uma linha.
SUM – soma conteúdos de até cinco expressões. Pode ser usado com sentenças NEXT; FOR e WHILE.
COUNT – conta os registros do banco de dados; pode ser usado com sentenças NEXT, FOR e WHILE.
REPORT – prepara relatórios padronizados sobre o conteúdo de um banco de dados. Pode ser usado com as sentenças FOR, WHILE e NEXT.
SET HEADING TO – usado com o comando REPORT para apresentar informações especiais na primeira linha do relatório.

Impressão

Há comandos especiais para controlar a entrada dos dados em uma impressora. São os seguintes:

SET PRINT ON/OFF
SET FORMAT TO PRINT/SCREEN
SET MARGIN TO
REPORT TO PRINT
SET HEADING TO
EJECT

Comandos de Posicionamento

Os comandos são usados para posicionar o banco de dados:

FIND
LOCATE
CONTINUE
GOTO ou GO
SKIP

FIND – coloca o banco de dados na primeira posição de um campo-chave especificado. Usa um arquivo de ÍNDICES para executar o posicionamento.

LOCATE – coloca o banco de dados na primeira posição do parâmetro especificado. Executa o posicionamento por meio da busca seqüencial do arquivo.

CONTINUE – posiciona na ocorrência seguinte do parâmetro especificado.

GOTO – posiciona o banco de dados no número de registro especificado. É um posicionamento de acesso direto.

SKIP – desloca para frente ou para trás o número de registros especificados pelo comando SKIP. É um acesso relativo.

Comandos de Entrada de Dados

Os comandos geralmente usados para entrada de dados são:

APPEND	CHANGE	SET FORMAT TO
INSERT	EDIT	INPUT
BROWSE	READ	ACCEPT

APPEND – acrescenta registros no final do banco de dados.

INSERT – acrescenta registros no meio do banco de dados.

BROWSE – possibilita a edição em tela cheia de até 19 registros por vez.

CHANGE – possibilita modificar o conteúdo de campos selecionados.

EDIT – permite a edição em tela cheia de um registro de dados.

READ – usado com os comandos SET FORMAT TO e @ x,y GET para entrada de dados em tela cheia nos campos e nas variáveis de memória.

SET FORMAT TO – chama telas especiais para serem usadas com os comandos APPEND, EDIT e READ.

INPUT – permite ao usuário inserir dados numéricos e lógicos em uma variável de memória.

ACCEPT – permite ao usuário inserir caracteres de dados em uma variável de memória.

Auxílio de Programação

Há alguns comandos na linguagem com a principal função de instrumentos de programação. Esses comandos especiais possibilitam usar os outros comandos de modo mais eficiente.

DO	LOOP
DO CASE/ENDCASE	WAIT
DO WHILE ../ENDDO	NOT ou *
RETURN	CANCEL
IF/ELSE/ENDIF	MODIFY COMMAND

DO – usado para "chamar" um programa do dBASE II.

DO CASE/ENDCASE – possibilita selecionar em meio a um conjunto de escolhas.

DO WHILE/ENDDO – permite repetir uma série de comandos, tantas vezes quantas necessárias.

RETURN – encerra o programa em uso. Conduz ao programa seguinte de nível mais alto.

IF/ELSE/ENDIF – permite a escolha de opções.

LOOP – conduz novamente ao comando DO WHILE. Usado para "desviar" de uma rota de processamento. É o comando do dBASE II mais próximo do comando GOTO do FORTRAN ou do BASIC.

NOTE ou * – permite acrescentar linhas de comentário ao programa.

CANCEL – encerra programas que estão sendo usados. Faz o usuário retornar ao teclado.

MODIFY COMMAND – chama um editor de texto simples a fim de escrever ou modificar arquivos do programa.

Variáveis de Memória

Alguns dos comandos do dBASE II podem armazenar dados como variáveis de memória do dBASE II. Outros são designados com o objetivo de lidar somente com variáveis de memória.

STORE SUM INPUT SAVE WAIT COUNT

ACCEPT RELEASE RESTORE

STORE – armazena um item na memória.

COUNT...TO – armazena na memória o resultado do comando de contagem.

SUM...TO – armazena nas variáveis de memória os resultados da operação de somar.

INPUT – armazena em uma variável de memória os dados numéricos ou lógicos fornecidos por meio do teclado.

ACCEPT – armazena em uma variável de memória os dados em caracteres fornecidos por meio do teclado.

RELEASE – "Apaga" as variáveis de memória.

SAVE – grava variáveis de memória em um arquivo em disco.

RESTORE – recupera para a memória variáveis de memória gravadas previamente em um arquivo em disco.

WAIT – usado para "fazer uma pausa" no programa. Pode ser usado para inserir um único caractere na memória.

Operações Gerais de Arquivo

Alguns dos comandos são recursos que possibilitam extrair informações de arquivos ou trabalhar com eles em vez de registros.

DISPLAY FILES ON
DISPLAY STATUS
DISPLAY STRUCTURE
SET ALTERNATE ON/OFF
SET ALTERNATE TO
DELETE FILE
RENAME
RESET

DISPLAY FILES ON – apresenta o diretório do disco.

DISPLAY STATUS – exibe os arquivos de banco de dados que estão sendo usados, os arquivos de índice usados com os bancos de dados, em que chaves de índices os arquivos estão sendo indexados e os parâmetros de processamento do dBASE II.

DISPLAY STRUCTURE – exibe a estrutura do banco de dados que está sendo utilizada.

SET ALTERNATE ON/OFF – possibilita enviar para um arquivo em disco os dados apresentados na tela ou enviados à impressora.

SET ALTERNATE TO [< arquivo >] – permite criar um arquivo .TXT para gravar em disco uma saída tela/impressora.

DELETE FILE – permite excluir o arquivo designado pelo nome.

RENAME – possibilita dar novo nome a um arquivo.

RESET – possibilita a mudança de discos – CP/M 80.

Comandos de Função Especiais

QUIT
REMARK
TEXT/ENDTEXT
PEEK
POKE
MODIFY COMMAND
WAIT
LOAD
CALL

QUIT – permite sair do dBASE II.

REMARK – apresenta uma linha de texto sem delimitadores.

TEXT/ENDTEXT – apresenta um bloco de texto sem delimitadores.

PEEK – exibe o conteúdo de uma posição de memória indicada pelo nome.

POKE – escreve na posição de memória designada pelo nome.

MODIFY COMMAND – cria ou modifica quaisquer arquivos ASCII escrevendo diretamente neles por meio do teclado.

WAIT – produz uma "pausa" no programa.

LOAD – carrega um programa assembly indicado pelo nome em áreas da memória não utilizadas pelo dBASE II.

CALL – chama um programa assembly anteriormente carregado na memória.

Controles de Parâmetros do dBASE II

Há uma série de parâmetros de processamento no dBASE II que podem ser controlados pelo usuário. Os parâmetros podem assumir duas formas: SET....TO ou SET...ON/OFF. Os parâmetros SET...TO não são chamados a menos que o usuário tome uma ação específica. Os parâmetros SET...ON/OFF são preestabelecidos no dBASE II. O valor preestabelecido ou "default" é a parte *em itálico* na descrição a seguir

SET CALL TO
SET CARRY ON/*OFF*
SET COLON *ON*/OFF
SET COLOR TO
SET CONFIRM ON/*OFF*
SET CONSOLE *ON*/OFF
SET DATE TO
SET DEBUG ON/*OFF*
SET DEFAULT TO
SET DELETE ON/OFF
SET ECHO ON/*OFF*
SET EJECT *ON*/OFF
SET ESCAPE ON/OFF
SET Fn TO
SET FORMAT TO
SET HEADING TO
SET INDEX TO
SET INTENSITY *ON*/OFF
SET LINKAGE ON/*OFF*
SET MARGIN TO
SET PRINT ON/*OFF*
SET RAW ON/*OFF*
SET SCREEN *ON*/OFF
SET STEP ON/*OFF*
SET TALK *ON*/OFF

SET CALL TO – permite escolher entre vários programas assembly carregados.

SET CARRY ON/*OFF* – escreve o conteúdo do último registro fornecido no novo, durante a execução do comando APPEND. É útil quando há poucas diferenças de um registro para outro.

SET COLON *ON*/OFF – dBASE II normalmente exibe o sinal de dois pontos nas extremidades de um campo ou variável em entradas de tela cheia.

SET COLOR TO – estabelece os atributos de tela (em computadores de 16 bits).

SET CONFIRM ON/*OFF* – o procedimento normal é o de passar automaticamente para o item seguinte em tela cheia. O comando CONFIRM exige que se tecle < RETURN > para avançar um item.

SET CONSOLE *ON*/OFF – interrompe, no terminal, a apresentação dos dados (inclusive das entradas pelo teclado).

SET DATE TO – possibilita estabelecimento interno das datas do sistema.

SET DEBUG ON/*OFF* – envia as linhas de comando e de controle de SET STEP ON para uma impressora. Usado para eliminar erros nos programas com operações em tela cheia.

SET DEFAULT TO – permite estabelecer ou padronizar (default) a unidade de disco que é diferente da unidade de discos normalmente acoplada.

SET DELETE ON/*OFF* – evita que os registros excluídos sejam usados pelo dBASE II.

SET ECHO ON/*OFF* – exibe as linhas de comando dos programas.

SET EJECT *ON*/OFF – possibilita que o dBASE II emita uma página ao iniciar um novo relatório.

SET ESCAPE *ON*/OFF – você pode normalmente sair de um programa ou de um comando pelo simples pressionar da tecla ESCAPE. Este parâmetro pode evitar que isso ocorra.

SET Fn TO – permite modificar as atribuições das teclas de função (somente no IBM PC).

SET FORMAT TO – permite escolher entre enviar os comandos para os arquivos de formatação, para o CRT ou para a impressora, pelo uso dos comandos @ x,y SAY e GET.

SET EADING TO – possibilita a entrada de uma linha de título quando usado com o comando REPORT.

SET INDEX TO – possibilita a seleção de arquivos de índice.

SET INTENSITY *ON*/OFF – as operações normais em tela cheia estão em intensidade total ou simples ou em vídeo normal e reverso. Coloca todos os dados no padrão de vídeo durante operação em tela cheia.

SET LINKAGE ON/*OFF* – une dois bancos de dados (com base em números de registro) com a finalidade de apresentação na tela e de relatórios.

SET MARGIN TO – estabelece a margem esquerda da impressão.

SET PRINT ON/*OFF* – liga e desliga a impressora.

SET RAW ON/*OFF* – as informações armazenadas são compactas. A apresentação normal dos campos introduz um espaço entre eles para facilitar a visualização.

SET SCREEN *ON*/OFF – liga e desliga operações na tela.

SET STEP ON/*OFF* – possibilita avançar uma linha de comando por vez em um programa.

SET TALK *ON*/OFF – liga e desliga a apresentação da resposta normal do dBASE II aos comandos.

Funções Aritméticas

O dBASE II permite executar as quatro funções aritméticas básicas: adição, subtração, multiplicação e divisão. Os operadores aritméticos padrão que o dBASE II pode identificar são:

+	adição
–	subtração
*	multiplicação
/	divisão
( )	parênteses para agrupamento
<	menor que
>	maior que
=	igual
#	diferente
< >	diferente
< =	menor do que ou igual
> =	maior do que ou igual
INT(X)	inteiro de x

Operações Lógicas

O dBASE II também permite usar os operadores lógicos padrão:

( )	parênteses para agrupamento
.NOT.	não, em notação booleana
.AND.	e, em notação booleana
.OR.	ou, em notação booleana

Para exemplificar o uso de um operador booleano, considere o comando:

DISPLAY FOR SALA = "3"
- .AND.CLASSE = "5"

Esse comando apresentaria todos os registros do banco de dados em que o conteúdo do campo SALA fosse 3 e o conteúdo do campo CLASSE fosse 5.

Operações com "Strings"

O dBASE II é especialmente adequado para operações com *strings*. As funções e os operadores usados são:

+	concatenação (justaposição) de *strings* "alfa" + "beto" = "alfabeto"

–	concatenação de *strings* com os espaços em branco inseridos à direita "alfa " – "beto" = "alfabeto "
$	operador de *substrings* "alfabeto" = memvar $(memvar,3,2) = "fa"
@	operador de busca de *strings* @("fa",memvar) = 3
LEN	tamanho das *strings* LEN(memvar) = 8
TRIM	elimina os espaços em branco após a *string* "alfabeto " = memvar TRIM(memvar) = "alfabeto"
!	função de maiúsculas !(memvar) = "ALFABETO"
VAL	converte a *string* em um número; pára na primeira letra "12B13" = numvar VAL(numvar) = 12
STR	converte um número em uma *string* 123.6 = numvar STR(numvar,8,2) = " 123.60"

Outras Funções

Há várias outras funções que servem como instrumentos para situações especiais. A lista abaixo mostra alguns exemplos de comandos que usam essas funções.

#	número de registro STORE # TO posição IF # = 27
*	registro excluído COUNT FOR * DISPLAY FOR *

Na presente edição do dBASE II, os registros excluídos não podem ser copiados ou usados em um relatório preparado pelo comando REPORT.

EOF – fim de arquivo
DO WHILE .NOT. EOF
IF EOF

TYPE – digitação de dados
U – não definido – não existe
C – Caractere
N – Numérico
L – Lógico

IF .NOT. TYPE('memvar') = 'U'

FILE – teste para a existência de arquivo
IF FILE ('MENU.CMD')

CHR – caractere. Possibilita armazenar e transmitir os códigos de controle ASCII que não podem ser impressos. A função CHR é usada especialmente para criar efeitos especiais no CRT ou na impressora. Há vários exemplos no Capítulo 14. CHR (10) fornece uma linha e CHR (13) produz um retorno do carro. CHR usa o valor decimal associado a um caractere.

```
STORE 'ALFA' + CHR(10) + CHR(13) TO X
? CHR(27) * CHR(49) + 'HI'
```

RANK – é uma função útil nos casos em que se queira usar uma função CHR mas não se tem a tabela de conversão para código ASCII. A função RANK informará que o número decimal corresponde a um dado caractere ASCII imprimível.

```
STORE RANK('E') TO CONTROL
? RANK('E')
? CHR(RANK('E'))
```

& – função de macrosubstituição que permite o uso dos conteúdos de uma variável sob circunstâncias limitadas.

```
STORE 'ESCOLA,DBF' TO memvar
```

(USE *memvar* não é permitido – não há um arquivo chamado memvar)

USE & memvar é idêntico a USE ESCOLA.

Essa função é usada onde o dBASE II normalmente não encontraria uma variável de memória e normalmente interpretaria de modo literal o que lá estivesse digitado. (Obs.: RunTime™ não permite um "&" como o primeiro caractere de uma linha.)

Essa foi uma apresentação resumida dos instrumentos usados com o dBASE II. Recapitulando, nossos instrumentos são:

Os Arquivos
A Linguagem

A linguagem consiste em comandos, funções e operadores. As funções e os operadores são usados com os comandos para especificar o que desejamos que o dBASE II execute. Todos eles podem ser combinados de inúmeras formas possíveis. Para uma visão geral das funções, operadores e comandos do dBASE II, consulte os Apêndices A, B e C.

Parte II

MONTAGEM DE UM SISTEMA DE CONTABILIDADE

Capítulo Quatro

O EXEMPLO DE UM SISTEMA DE CONTABILIDADE

Muitos de nós aprendemos a programar porque temos uma tarefa específica para o computador executar. Muitas vezes não há tempo suficiente para se tentar a abordagem acadêmica tradicional nem para estudar teoria da programação. Temos uma tarefa diante de nós – que precisa ser imediatamente executada. Assim, aprendemos a programar examinando programas que outros desenvolveram e como os instrumentos de programação foram usados. É um modo natural de aprender.

Para ajudá-lo a programar com o dBASE II, vamos desenvolver um conjunto de aplicativos possíveis em contabilidade. Isso nos permitirá explicar a linguagem a partir de um contexto prático. Usaremos a maioria dos instrumentos de programação do dBASE II pelo menos uma vez; naturalmente, alguns deles serão usados várias vezes. Os padrões básicos da estrutura dos programas em dBASE II ficarão, assim, evidenciados. Tente considerar os exemplos em dois níveis: como solução para um problema claramente estruturado para contabilidade e como solução para um problema em geral.

Nos próximos capítulos vamos imaginar que somos distribuidores de livros. Nossa empresa compra livros de editoras e os revende a livrarias. Para auxiliar a dirigir mais eficientemente a empresa, vamos adquirir um microcomputador e o dBASE II. Como o hardware de que vamos precisar – isto é, o sistema de disco necessário – dependerá do tamanho dos bancos de dados com seus índices correspondentes, vamos primeiramente planejar nosso software e depois adquirir o hardware necessário.

Grande parte da atividade diária de nossa empresa se relaciona com a manutenção dos registros. Para que o sistema de software nos auxilie nessa tarefa, vamos planejá-lo de modo a garantir apoio nas seguintes áreas:

- Manutenção dos registros de clientes
- Listas de etiquetas de endereçamento
- Gerenciamento de inventários
- Entradas de pedidos de vendas
- Preenchimento e faturamento de pedidos
- Contas a receber

Há dois modos de desenvolver um programa que execute o que desejamos. O primeiro consiste em escrever um único programa, longo e complexo, que faça tudo. Em termos gerais, esse procedimento não é aconselhável. O segundo método é um sistema de módulos que inclui vários programas pequenos e simples que remetem a cada uma das funções de contabilidade (ver Figura 1). Cada módulo é um programa individual que executa uma parte de toda a tarefa. Por meio dessa abordagem por módulos, podemos estruturar nosso sistema, um programa por vez, com a vantagem adicional de podermos modificar um dos programas, ou incluir um outro com relativa facilidade.

Para unificar todos os programas individuais em um único pacote coerente, utilizaremos um *sistema de menu*. Aqui, usamos um programa para apresentar uma lista de outros programas que podem ser escolhidos pelo "menu". A Figura 2 mostra um diagrama de menu.

Se tivermos uma grande variedade de escolhas a fazer, poderemos optar por ter um supermenu ou um "menu de menus". É esta a abordagem que utilizaremos em nosso sistema de software para a distribuidora de livros.

A Figura 3 mostra o nosso sistema de menu, cujo diagrama é muito semelhante a um organograma. Todos os submenus pertencem ao Menu Principal – um sistema hierárquico. É o mais simples dos vários esquemas de menu possíveis.

A finalidade de cada menu é a de apresentar uma lista de escolhas e fornecer um método para selecionar uma delas. Quase todas as escolhas que o Programa 1 – nosso Menu Principal – apresenta também consistem em menus. Cada um desses menus de segundo nível possui, por sua vez, uma lista de tarefas que podem ser realizadas por meio dele.

Por exemplo, se escolhermos o Menu 2, o Menu do Arquivo de Clientes, vamos receber uma lista das funções que podem ser executadas por meio dele. Ao escolher um item, na verdade estamos "chamando" um programa que executa a tarefa – como, por exemplo, acrescentar novos registros a um banco de dados de clientes.

O sistema hierárquico de menus não permite ir diretamente de um programa no Menu de Arquivo de Clientes para um outro no Menu de Pedidos de Vendas. Quando terminamos de utilizar um programa, somos automaticamente enviados de volta ao menu no qual ele foi escolhido. Para escolher um programa em outro menu, teremos, antes, de retornar ao Menu Principal, selecionar o menu que contenha o programa que desejamos e então escolher o programa desejado no submenu.

A vantagem em se usar um sistema de menus é evidente – ele oferece um método de nos "lembrarmos" qual dos programas, entre todos que estão no disco, é o específico que queremos usar. Ele também "documenta" indiretamente o sistema de software. Podemos usar um sistema de menus para executar tarefas realmente complexas quase sem qualquer treinamento. A principal desvantagem é a de que só podemos fazer o que está no menu.

Os Programas

Se você desenvolver um programa (às vezes chamado arquivo comando) em dBASE II, usando um computador de 16 bits, como o PC da IBM, o dBASE II automaticamente incluirá .PRG após o nome de arquivo de seu programa. Se estiver usando um computador de 8 bits, rodando em CP/M, o dBASE II incluirá .CMD, também após o nome de arquivo do programa. Aqui, incluiremos .CMD/.PRG após os nomes de nossos programas.

Se estivermos no sistema operacional do computador poderemos carregar o dBASE II e "rodar" nosso programa menu, chamado MENU, com um comando. Digite DBASE MENU após a indicação do sistema operacional.

```
A > DBASE MENU
```

Para "chamar" o programa MENU.CMD/.PRG já no dBASE II, basta digitar DO MENU após uma indicação de ponto:

```
. DO MENU
```

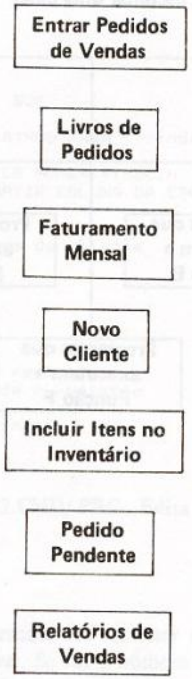


Figura 1. Cada quadro representa uma função

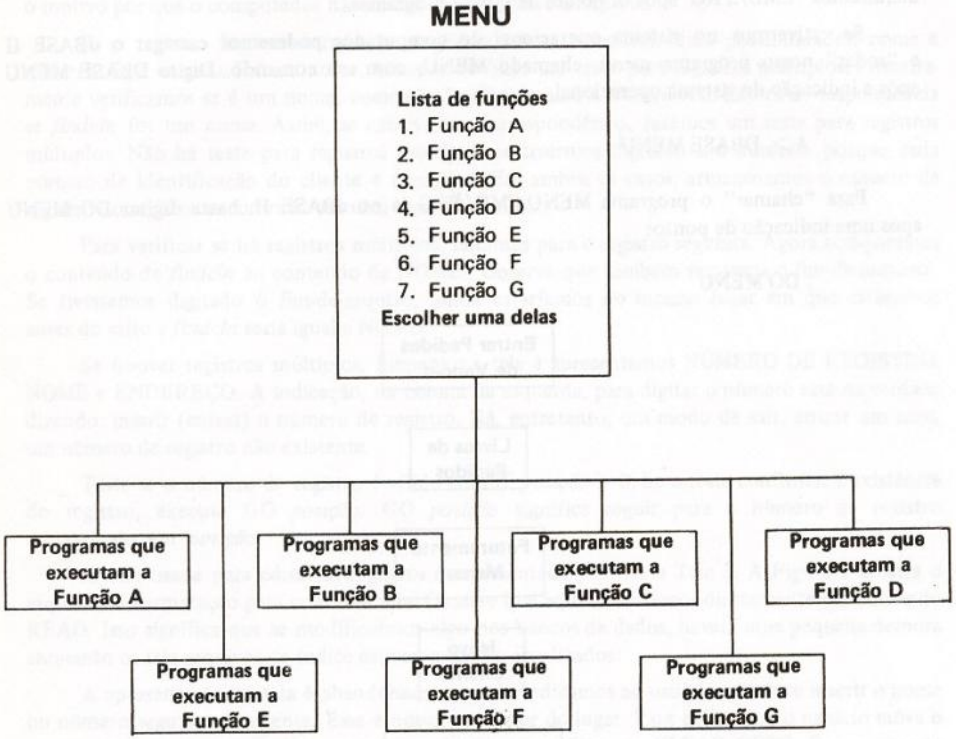


Figura 2. Diagrama do sistema de menus

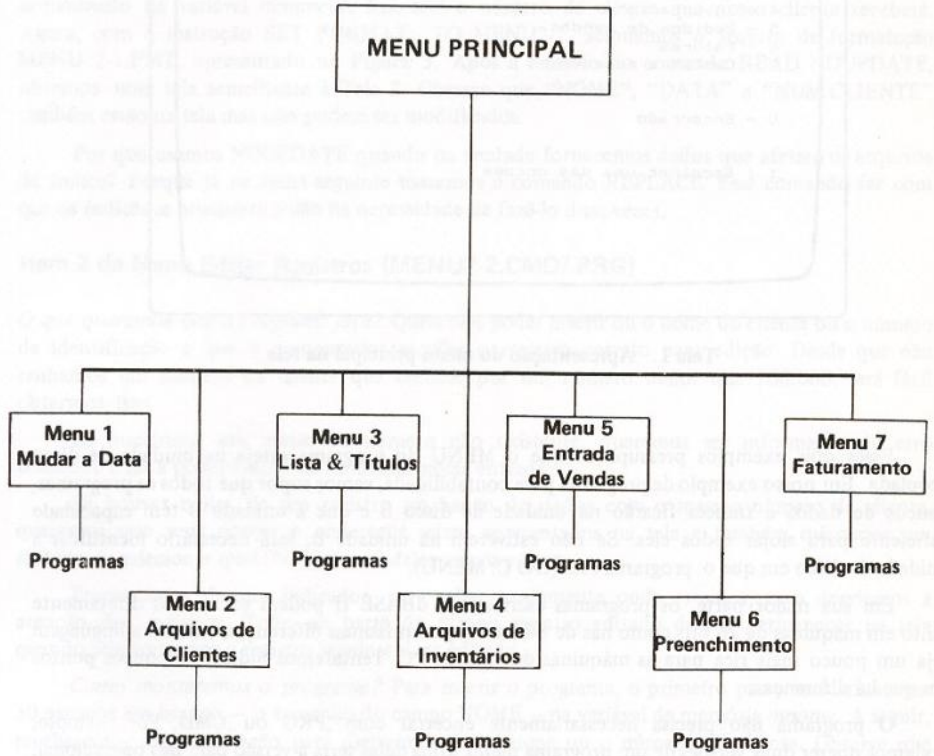
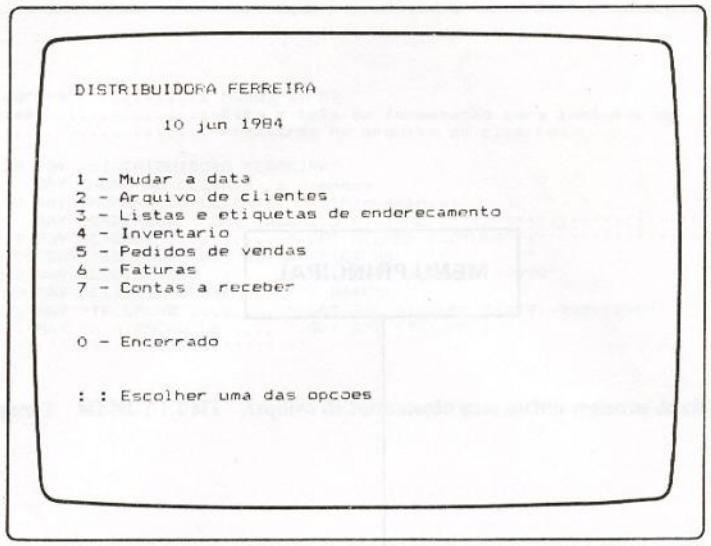


Figura 3. Sistema hierárquico de menus



Tela 1. Apresentação do menu principal na tela

Esses dois exemplos pressupõem que o MENU do programa esteja na unidade de disco acoplada. Em nosso exemplo de programa para contabilidade, vamos supor que todos os programas, bancos de dados e índices ficarão na unidade de disco B e que a unidade B tem capacidade suficiente para alojar todos eles. Se não estiverem na unidade B, será necessário identificar a unidade de disco em que o programa está (DO C: MENU).

Em sua maior parte, os programas escritos em dBASE II podem ser usados diretamente tanto em máquinas de 16 bits como nas de 8 bits. Há pouquíssimas diferenças, embora a linguagem seja um pouco mais rica para as máquinas do tipo do PC. Tentaremos indicar os poucos pontos em que há diferenças.

O programa não precisa necessariamente encerrar com .PRG ou .CMD. Por exemplo, podemos querer duas versões de um programa menu. Uma delas seria a versão para uso operacional, a outra, a nova versão que está sendo desenvolvida. Podemos chamar a nova versão NOVO.MEN. Para executar a nova versão, usamos:

```
. DO NOVO.MEN
```

Nomearemos sistematicamente os programas de nosso software para contabilidade. Os programas a serem escolhidos com o MENU serão designados por MENU, seguido de seu número na relação. Por exemplo, Item 1, Modificar a Data, torna-se MENU1.CMD/.PRG. Os programas que forem selecionados em um submenu, como MENU3, serão designados, de modo semelhante: MENU3-1, MENU3-2 etc. Isso fornece um método rápido para dar uma boa visão do programa.

Desenvolveremos as partes de nosso sistema de contabilidade pela ordem em que aparecem no Menu Principal. Isso nos possibilitará executar os itens mais fáceis em primeiro lugar. Além disso, precisaremos desenvolver o arquivo de clientes e o de inventário antes de podermos desenvolver o sistema de entradas de pedidos de vendas.

Quando desenvolvemos um programa, devemos iniciar com uma noção clara sobre o que desejamos obter dele. Somente então poderemos adaptar o software para fazer exatamente aquilo que precisamos. O tempo gasto no desenvolvimento é justificado pela maior eficiência (para as pessoas) que o programa irá proporcionar.

O Programa

Agora vamos iniciar realmente a programação do Menu Principal e dos submenus. Quando chamamos nosso Menu Principal, queremos que a tela tenha a mesma apresentação que a Tela 1.

Avaliamos o que o programa deverá fazer e temos alguma certeza de que o Menu Principal está montado para abranger todas as tarefas escolhidas. Também escolhemos o tipo de disposição na tela que o Menu Principal deverá ter. Assim, começamos a desenvolver um programa que possibilite essa apresentação, bem como os métodos de escolher os itens assim dispostos. Apresentamos esse programa como o Programa 1 – o programa do Menu Principal.

```
SET TALK OFF
```

Ao inserir o programa nossa primeira providência será acionar o comando SET TALK OFF. Isso cancelará o modo conversacional do dBASE II. Com TALK OFF a tela ficará limpa, a menos que diretamente façamos algo para produzir uma apresentação nela. Naturalmente, as mensagens de erro do dBASE II e do sistema operacional continuarão a ser apresentadas se e quando ocorrerem erros.

```
SET DEFAULT TO B
```

Esse comando permite colocar todos os nossos programas e bancos de dados em uma unidade de disco que não seja a unidade acoplada, sem necessidade de identificar a unidade a cada vez que um arquivo for chamado. Esse comando faz com que a unidade designada se apresente como "unidade acoplada" para operações no interior do dBASE II.

O comando SET controla uma grande quantidade de parâmetros de processamento em dBASE II, além de TALK e DEFAULT. A situação desses parâmetros e as informações sobre o estágio dos arquivos de bancos de dados e de índices selecionados podem ser exibidos na tela com o comando DISPLAY STATUS.

```
RESTORE FROM DATA
```

Esse comando "restaura" as variáveis de memória do dBASE II que foram anteriormente gravadas em disco. DATA é o nome do arquivo em disco. Esse arquivo aparecerá no diretório de disco como DATA.MEM. Os programas do exemplo de sistema de contabilidade utilizarão duas variáveis datas: *mdata* e *cdata*. Se *mdata* for 11/10/84, *cdata* será 11 OUT 84.

```
* Programa ..............: MENU.CMD (ou MENU.PRG)
* Obs ...................: Este programa e o Menu Principal

SET TALK OFF
SET DEFAULT TO B
RESTORE FROM Data ADDITIVE
SET DATE TO madata
DO WHILE T
   ERASE
   @ 5,30 SAY "OS MELHORES LIVROS DA DISTRIBUIDORA FERREIRA"
   @ 7,32 SAY cdata
   @ 10,25 SAY "1 - Mudar a data"
   @ 11,25 SAY "2 - Arquivo de clientes"
   @ 12,25 SAY "3 - Listas e etiquetas de enderecamento"
   @ 13,25 SAY "4 - Inventario"
   @ 14,25 SAY "5 - Pedidos de vendas"
   @ 15,25 SAY "6 - Faturas"
   @ 16,25 SAY "7 - Contas a receber"
   @ 19,25 SAY "0 - Encerrado"
   @ 21,29 SAY "Escolher uma das opcoes"
   STORE " " TO escolha
   @ 21,24 GET escolha
   READ NOUPDATE
   IF escolha = "0"
      QUIT
   ENDIF
   IF escolha $"1234567"
      STORE "MENU"+escolha TO selecao
      DO &selecao
   ENDIF
ENDDO
```

Programa 1. MENU.CMD/.PRG O programa do menu principal

Antes de podermos utilizar o programa menu, temos de criar essas variáveis de memória e gravá-las (SAVE) no arquivo DATA.MEM. Isso pode ser feito da seguinte forma:

```
STORE "11/10/84" TO mdata
STORE "11 OUT 84" TO cdata
SAVE ALL LIKE ?data TO B: DATA
```

Daqui por diante, nosso sistema se encarregará dos processos de modificação e gravação de datas por meio do Menu Item 1. Essa seleção de menu chama um programa (Programa 5, no Capítulo 12) cuja função está em modificar, validar e gravar essas duas variáveis de datas.

O programa do Menu Principal foi desenvolvido para funcionar tanto nas máquinas de 8 como de 16 bits. Com um computador de 16 bits, como o PC, da IBM, poderíamos utilizar-nos do fato de que o dBASE II pode "ler" o sistema de relógio do PC. O mesmo acontece com os computadores de CP/M 80.

```
SET DATE TO &mdata
```

Esse comando substitui o sistema de datas do dBASE II [DATE ( )] pelo conteúdo de variável *mdata*. O símbolo & identifica *mdata* como uma variável de memória e permite seu uso em uma linha de comando.

O sistema de datas é usado para atualizar o cabeçalho dos registros do banco de dados. É também usado nos relatórios do dBASE II. Nós o usaremos freqüentemente em nosso exemplo de sistema de contabilidade (a questão das datas em geral será apresentada de modo mais completo no Capítulo Doze).

```
DO WHILE T
```

Os comandos entre DO WHILE T e o comando ENDDO são repetidos enquanto a expressão que segue DO WHILE permanece verdadeira. A letra T representa True (Verdadeiro). O comando que usamos cria um loop e literalmente significa EXECUTE (DO) INFINITAS VEZES.

A cada passagem pelo loop, o programa limpa a tela, apresenta o menu e espera pela escolha de um item. Se o item escolhido não for válido, o programa nos remeterá ao início do loop, onde reinicia o procedimento.

```
ERASE
```

Esse comando limpa a tela.

```
@ 5,30 SAY "DISTRIBUIDORA FERREIRA"
```

O texto, as expressões, variáveis de memórias e os campos podem ser apresentados na tela pelo comando SAY.

A @ lin,col SAY é usada para apresentação na tela da expressão após a palavra SAY. A informação é apresentada na linha relacionada, começando na coluna indicada. No dBASE II a tela é numerada de cima para baixo. A primeira linha é a linha 0. As colunas são numeradas da esquerda para a direita. A primeira coluna é a coluna 0. Também não é aconselhável usar a Coluna 0.

Pode-se usar um par de aspas simples ou um par de aspas duplas como delimitadores, porém o delimitador deve ser o mesmo em ambas as extremidades.

Nosso exemplo emprega aspas duplas. Alguns computadores possuem tanto o sinal de aspas simples (apóstrofo ') como o sinal de acentuação (`) no teclado. O acento *não pode* ser usado como delimitador!

```
@ 7,33 SAY cdata
```

Apresenta o conteúdo da variável de memória *cdata* na Linha 7, a partir da Coluna 33.

```
STORE '      ' TO escolha
```

Armazena um espaço em branco na variável de memória *escolha*. Esse comando cria a variável se ela não existir. É preciso que a variável exista antes de podermos usá-la. Depois de criada a variável, pode-se tomar uma ação direta no sentido de eliminá-la.

```
@ 21,24 GET escolha
```

Apresenta a variável *escolha* na tela, na Linha 21, Coluna 24. Somente as variáveis de memória e os campos do banco de dados podem ser apresentados por meio de GET.

```
READ NOUPDATE
```

Posiciona o cursor no início do primeiro item apresentado, por meio do comando GET, neste caso, a variável *escolha*. Isso possibilita fornecer dados diretamente para a variável com o uso do teclado. Somente os caracteres imprimíveis podem efetivamente ser fornecidos à variável por meio do teclado. O dBASE II pode agir sobre os caracteres de controle, mas os filtra e evita que se tornem efetivamente parte da variável.

O comando READ fará com que o programa se interrompa nesse ponto, até que o usuário insira alguma informação por meio do teclado.

A palavra NOUPDATE evita que os arquivos de índice que estão sendo usados sejam atualizados. Na primeira passagem pelo menu, não há índices sendo usados. Mas isso pode não acontecer nas passagens seguintes. Se a variável que está sendo "lida" por meio de READ mudar, o dBASE II examinará cada arquivo de índice em uso para se certificar de que nenhuma chave de índice tenha sido modificada. Esse procedimento não leva muito tempo – mas pode ser notado. É um bom hábito de programação usar a palavra NOUPDATE sempre que não se quiser modificar uma chave de índice.

O comando READ armazena nossa escolha na variável de memória *escolha.*

```
IF escolha = "0"
  QUIT
ENDIF
```

O comando IF é às vezes considerado como um comutador. Ele executa os comandos entre IF e ENDIF sempre que a expressão que vem em seguida ao termo IF for verdadeira (True). A expressão em nosso exemplo é *escolha* = "0". Se a variável *escolha* contiver um 0, vamos querer abandonar (QUIT), um dos comandos mais importantes do dBASE II. Quando usamos o comando QUIT, o dBASE II assume toda a operação de encerramento: grava e limpa todos seus buffers internos, encerra todos os arquivos abertos, e em seguida sai para o sistema operacional.

Observe que o zero está entre os delimitadores. Isso porque nossa variável *escolha* é uma *string* de caracteres; a entrada de um número não modifica o tipo da variável. Os números podem ser tanto caracteres como números e é necessário informar ao computador como os estamos considerando.

```
IF escolha $ "1234567"
  STORE "MENU" + escolha TO seleção
  DO &seleção
ENDIF
```

Esse comutador demonstra vários recursos do dBASE II e elimina 21 linhas do código convencional de programação do dBASE II. Em nosso sistema de menus tínhamos sete escolhas

possíveis, os itens de 1 a 7, designados como MENU1, MENU2 etc. Esses recursos funcionam porque foi essa a estrutura utilizada para designar nossos programas. Não conseguiríamos esse resultado se tivéssemos chamado os programas de JOE, FRED etc.

Normalmente precisaríamos de uma estrutura IF/ENDIF para cada uma das possibilidades de menu. Seria necessário toda uma seqüência de comandos IF e cada uma delas teria a seguinte forma:

```
IF escolha = '1'
  DO MENU1
ENDIF
.
.
.
IF escolha = '7'
  DO MENU7
ENDIF
```

O programa examinaria cada IF em seqüência até satisfazer uma determinada condição. A seguir executaria os comandos dentro da estrutura IF/ENDIF. Após terminar, o programa continuaria a examinar os comandos IF até atingir ENDDO.

O primeiro recurso que utilizamos está no operador de *substrings* – $. A linha de comando agora significa: "se a escolha estiver contida na *string* de caracteres marcada pelos delimitadores". Esse recurso não apenas permite substituir uma série de comutadores, mas também elimina uma série talvez longa de operadores lógicos OR (OU). Teríamos então escrito:

```
IF escolha $ '1234567'
```

da seguinte forma:

```
IF escolha = '1' .OR. escolha = '2' .OR. escolha = '3' ...etc.
```

As duas construções chegam exatamente ao mesmo resultado. Qual delas você prefere?

Nosso próximo truque será o de concatenar ou unir a *string* de caracteres MENU com o conteúdo da variável *escolha* para obter nossa SELEÇÃO. Ao utilizar o sinal + entre duas *strings* de caracteres, estamos unificando-as. As crianças muitas vezes dizem 2 + 2 igual a 22. Estão próximas: '2' + '2' igual '22'. Talvez por isso elas se dêem tão bem com o computador!

Se tivesse escolhido o Item 1 – Modificar a Data – a *seleção* de variáveis conteria a *string* de caracteres MENU1. Instruímos o computador para executar o programa com:

```
DO &seleção
```

O sinal & é uma indicação para o dBASE II usar a variável *seleção*. DO *seleção* só teria sentido se houver um programa chamado SELEÇÃO. Se *seleção* contiver MENU1,

DO &seleção = DO MENU1

Esse é um bom exemplo do uso de operações com *strings* de caracteres para diminuir a quantidade de código vertical. O dBASE II é um programa interpretado e deve "ler" todo o código vertical. Aumentamos a velocidade do programa com a diminuição do código vertical. Apesar da execução desejada ser um pouco mais lenta, a velocidade geral do programa é maior.

Neste exemplo específico, a velocidade relativa não faz muita diferença; mas nem sempre é assim. Entretanto, não vamos nos demorar na questão da velocidade. Uma das críticas feitas aos programadores profissionais é de darem ênfase excessiva à velocidade. Devemos tomar cuidado, ao escrever nosso programa, para que seja adequado a nossas necessidades, sempre que isso for possível.

A ordem no programa dos dois comutadores IF/ENDIF é importante. O comutador IF escolha = '0' deve ser colocado em primeiro lugar porque o '0' também será utilizado como meio de retornar ao Menu Principal. O programa retornará no ENDIF que vem em seguida a DO *&seleção;* e se IF escolha = 0 vier em seguida, o programa executará o comando QUIT (abandonar e sair para o sistema operacional). Não é isso entretanto o que desejamos, Assim, você pode perceber por que precisamos de um método estruturado para sair dos submenus para o Menu Principal (a passagem de um menu para outro é geralmente realizada pelo comando DO CASE exatamente por esse motivo).

Nosso programa introdutório realiza muita coisa em um pequeno programa. Ele fornece o esquema geral de nosso sistema de software, apresenta na tela aquilo que desejamos, fornece-nos métodos de selecionar em meio às opções da tela e rejeita qualquer seleção que não esteja no menu. Pode receber acréscimos sempre que necessário e poderá ser executado em qualquer computador que opera com o dBASE II.

Capítulo Cinco

O ARQUIVO DE CLIENTES

O arquivo de clientes é um elemento básico em muitos dos sistemas de contabilidade. Nele armazenamos informações essenciais tais como nome, endereço, número telefônico e número de identificação. Nosso arquivo de banco de dados é denominado CLIENTES.

Pelo fato de o arquivo de clientes ser utilizado por outros programas, como pedidos de vendas e contas a receber, é importante vincular eficientemente os registros de clientes de um sistema de contabilidade. O nome do cliente não é necessariamente único (pense por exemplo no nome "Acme Hardware"). Fornecemos esse identificador, atribuindo a cada cliente um número exclusivo. Optamos por inserir esse número de cliente (NUMCLN) como um campo de caracteres, pois não precisaremos efetuar cálculos aritméticos com ele.

Nesse capítulo, vamos usar o arquivo de clientes (MENU 2 de nosso Menu Principal) para exemplificar os programas que acrescentam, modificam e excluem registros de um banco de dados. O programa será MENU 2-1.CMD/.PRG, MENU 2-2.CMD/.PRG e MENU 2-3.CMD/.PRG, respectivamente. Poderíamos executar todas essas atividades sem necessidade de um programa, mas usaremos o programa porque ele possibilita fazer mais do que simplesmente executar inclusões (APPEND), edições (EDIT) ou exclusões (DELETE), como veremos.

Com CLIENTES.DBF serão usados e mantidos três arquivos de índice (ver Figura 1). O índice de números de identificação de clientes, NUMCLN.NDX, tem como função principal vincular os registros de CLIENTES.DBF a registros em outros bancos de dados. O índice de nomes de clientes possibilita o acesso aos registros por um método conveniente para nós. O índice do CEP ordena os registros pela ordem alfabética dos códigos CEP. Observe que em nosso sistema, o computador atribui os números de registro e a data de entrada; não há condições de mudar isso.

Para ter um arquivo de índice, os bancos de dados não precisam ter registros já inseridos em ordem e, de fato, é mais rápido montar o índice *antes* que os registros sejam fornecidos.

Como pretendemos fazer mais que uma única coisa com o arquivo de clientes, vamos construir um menu que nos permita selecionar a partir de nossas escolhas. A tela resultante deste programa de menus é apresentada como Tela 1. O programa menu, MENU 2.CMD/.PRG, é apresentado como Programa 1.

```
   DESCRICAO          NOME DO CAMPO     TIPO DE CAMPO    TAMANHO DE CAMPO

NOME DO CLIENTE       NOME                   C                30
A ATENCAO DE          ATN                    C                30
ENDERECO              ENDERECO               C                25
CIDADE                CIDADE                 C                20
ESTADO                ESTADO                 C                 2
CODIGO CEP            CEP                    C                 5
NUMERO TELEFONICO     TEL                    C                14
CLIENTE DESDE         DATA                   C                 8
NUMERO DO CLIENTE     NUMCLN                 C                 6

                                  TAMANHO DO REGISTRO:... 141 BYTES
ARQUIVO A SER INDEXADO EM:   NUMCLN        PARA  NUMCLN
                             NOME          PARA  CLIENTE
                             CEP + NOME    PARA  CEP
```

Figura 1. Plano do banco de dados de clientes

Descrição do Programa Menu

A primeira coisa a fazer em qualquer programa é reservar algumas linhas para a apresentação dos objetivos do programa e quando foi desenvolvido ou modificado. O asterisco no início da linha significa o mesmo que o comando NOTE. O dBASE ignora as linhas que se iniciam por * ou então por NOTE. É um bom hábito de programação fazer comentários (observações) em todo o programa para explicar o que está sendo feito e por quê – especialmente se for usado algum artifício. Em nossos exemplos normalmente não faremos comentários pois estaremos sempre descrevendo o programa.

O comando SET TALK OFF cancela o modo convencional normal do dBASE II – só será apresentado o que o usuário comandar. A seguir, abrimos o arquivo de clientes com USE CLIENTES e posicionamos o banco de dados no último registro fornecido por meio de GO BOTTOM. A razão para isso é que esse procedimento permite determinar o último número de identificação de cliente. Vamos precisar desse número para poder incluir novos clientes no banco de dados.

Neste ponto, devemos examinar a possibilidade de não haver registros no banco de dados. Se não houver, o banco de dados estará posicionado no Registro 0. Para testar se há ou não registros no banco de dados usamos:

```
IF # > 0
```


(O símbolo # é usado no dBASE II para representar o número de registro. Também é utilizado para significar "diferente de".)

Se tivermos registros (lembre-se, estamos no último registro), armazenamos o conteúdo do campo NUMCLN na variável de memória *mnumcln* (*não* é aconselhável deixar uma variável de memória em um campo com o mesmo nome). Também criamos uma *string* de caracteres, *ultentr*, composta pelo número de identificação, pelo nome e pelo endereço do cliente (não se esqueça de deixar espaços entre delimitadores para separar esses campos). Usaremos essas duas variáveis, *mnumcln* e *ultentr*, para acrescentar registros no programa.

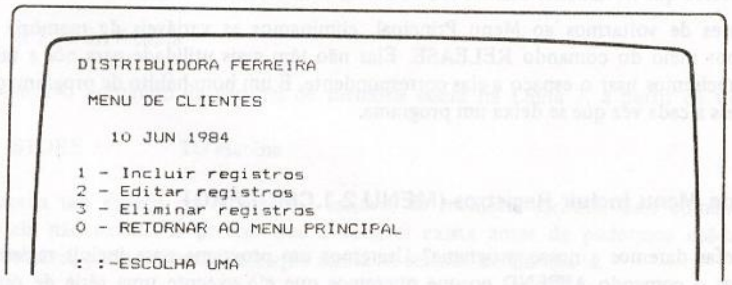
Não havendo registros no banco de dados, armazenamos o valor 100000 em *mnumcln*. Isto significa que nosso primeiro cliente terá o número 100001. Observe que também criamos uma variável de memória, *ultentr*, com um espaço em branco. Isso porque queremos que a variável seja apresentada em nosso programa para incluir registros. O dBASE II responderá a uma variável não-existente com uma mensagem de ERRO DE SINTAXE.

O dBASE II permite abrir 16 arquivos. Um loop DO contará com um arquivo aberto. Quando inserirmos esse loop DO teremos usado cinco de nossos 16 arquivos. Abriremos três arquivos de índice e um programa adicional em um total de nove. Esses nove arquivos são:

1. MENU .CMD/.PRG (Capítulo 4)
2. DO WHILE (em MENU.CMD/.PRG)
3. MENU2.CMD/.PRG (programa menu do arquivo de clientes)
4. DO WHILE (em MENU2.CMD/.PRG)
5. CLIENTES.DBF (em MENU2.CMD/.PRG)
6. MENU 2-1.CMD/PRG (inclui registros)
7. NUMCLN.NDX
8. CLIENTES.NDX
9. CEP.NDX

Ainda temos muitos arquivos não usados, mas sua utilização é algo que se deve ter em mente.

Nesse menu optamos por usar DO CASE para selecionar nossa escolha de programa. O comando DO CASE é semelhante ao IF, exceto que somente uma das alternativas poderá ser executada a cada passagem do loop.



Tela 1. Tela do menu de clientes

```
* Programa ..... : MENU2.CMD (ou MENU2.PRG)
* Observacoes ..... : Este e o programa do Menu do Arquivo de Clientes

USE Clientes
GO BOTTOM
IF # > 0
   STORE VAL(NUMCLN) TO mnumcln
   STORE NUMCLN + "  " + NOME + " " + ENDERECO TO Ultentr
ELSE
   STORE "  " TO Ultentr
   STORE 100000 TO mnumcln
ENDIF
DO WHILE T
   STORE " " TO escolha
   ERASE
   @ 2,30 SAY "DISTRIBUIDORA FERREIRA"
   @ 4,32 SAY "MENU DE CLIENTES"
   @ 6,34 SAY cdata
   @ 8,25 SAY "1 - Incluir registros de clientes"
   @ 9,25 SAY "2 - Editar registros de clientes"
   @ 10,12 SAY "3 - Excluir registros de clientes"
   @ 14,15 SAY "0 - VOLTAR AO MENU PRINCIPAL"
   @ 17,25 SAY "  - ESCOLHER UMA"

   @ 17,24 GET escolha
   READ NOUPDATE
   DO CASE
      CASE escolha="1"
           DO Menu2-2
      CASE escolha="2"
           DO Menu2-2
      CASE escolha="3"
           DO Menu2-3
      CASE escolha="0"
           RELEASE mnumcln,Ultentr
           RETURN
   ENDCASE
ENDDO
```

Programa 1. MENU2.CMD/.PRG Programa menu do arquivo de clientes

Observe que novamente usamos NOUPDATE com o comando READ.

Antes de voltarmos ao Menu Principal, eliminamos as variáveis de memória *mnuncln* e *ultentr* por meio do comando RELEASE. Elas não têm mais utilidade para nós e talvez futuramente precisemos usar o espaço a elas correspondente. É um bom hábito de programação eliminar as variáveis a cada vez que se deixa um programa.

Item 1 de Menu Incluir Registros (MENU 2-1.CMD/.PRG)

Qual tarefas daremos a nosso programa? Usaremos um programa para incluir registros, em vez de utilizar o comando APPEND porque queremos que ele execute uma série de outras tarefas além da simples inclusão de registros.

O programa deverá verificar se nosso "novo" registro já está no banco de dados. De modo geral, não é aconselhável ter vários registros para um mesmo cliente. Como não é impossível que dois clientes tenham o mesmo nome, não será conveniente que o computador cancele a inserção duplicada, mas apenas nos apresente suficientes quantidades de dados da(s) inserção(ões) duplicada(s) para decidirmos sobre a inserção ou não do novo registro.

O programa também deverá fazer a atribuição, fornecer o número do cliente, além de inserir a data no campo DATA.

Como é fácil perder de vista o ponto do programa em que nos encontramos ao incluir registros, também será útil que o programa mantenha a localização do último cliente incluído e apresente essa informação na tela.

Como montamos o programa? Nosso primeiro passo consiste em selecionar os arquivos índices a serem usados com o banco de dados CLIENTES. No programa, vamos utilizar todos os índices: NUMCL, NOME e CEP. É com o primeiro índice da lista que vamos efetivamente trabalhar. Os outros dois serão usados para manter os arquivos atualizados.

No exemplo, usaremos uma versão ligeiramente diferente do comando DO WHILE. Em primeiro lugar, armazenaremos um dado lógico True em uma variável de memória denominada inclreg (de "incluir registros"). Enquanto *inclreg* permanecer verdadeira, o loop continuará a se repetir.

Tão logo estejamos no interior do loop, a tela será apagada e serão armazenados 30 espaços em branco (o tamanho exato do campo NOME) na variável de memória *mnome*. Apresentamos a variável de memória *ultentr* na última linha da tela e damos uma indicação ao usário para que digite o nome do novo cliente.

Neste ponto, há duas possibilidades: inserirmos ou não um nome. Se não inserirmos, será necessário encerrar o programa e retornar ao menu. Assim, fazemos um teste com o espaço em branco. Se o campo estiver em branco armazenamos False em *inclreg*. O comando LOOP faz o usuário retornar ao DO WHILE. Como *inclreg* é agora Falso, o programa automaticamente salta para a instrução ENDDO. Usando o comando RELEASE, eliminamos todas as variáveis criadas no programa e retornamos ao menu.

Alguns consideram o uso de LOOPs como uma técnica não muito boa de programação em uma linguagem estruturada como o dBASE II. Nós utilizamos LOOP nesses exemplos porque ele é facilmente identificável e ajuda a compreender o que está ocorrendo. Em MENU2-1.CMD/.PRG, precisaríamos incorporar uma série de comandos IF/ELSE para obter o mesmo resultado.

Se for fornecido um nome, eliminaremos os espaços em branco que restarem e armazenaremos o nome na nova variável *trimnome*. A função! converte a expressão para maiúsculas. Sabemos que os caracteres "A" e "a" são equivalentes, mas o computador não. Como o computador não é eficiente em avaliação de contexto, todas as comparações deverão estar em maiúsculas. STORE! ("abc" TO *alfa* armazenaria ABC em *alfa*).

A função TRIM elimina os espaços em branco entre *strings*.

```
* Programa ..............: MENU2-1.CMD (ou MENU2-1.PRG)
* Obs. ..................: Este e o programa para incluir registros ao
* ........................ Arquivo de Clientes.

SET INDEX TO Clientes,Numcln,CEP
STORE T TO inclreg
DO WHILE inclreg
   ERASE
   STORE "                                  " TO mnome
   @ 23,1 SAY "ULTIMA ENTRADA " + Ultentr
   @ 21,1 SAY "ENTRAR O NOME DO NOVO CLIENTE" GET mnome
   READ NOUPDATE

   IF mnome = "
      STORE F TO inclreg
      LOOP
   ENDIF
   STORE !(TRIM(mnome)) TO trimnome
   FIND &trimnome
   STORE F TO oktoadd

   IF # > 0
      ERASE

      @ 1,1 SAY "REGISTROS QUE CORRESPONDEM AO NOME DO CANDIDATO"
      DISPLAY OFF NOME,ENDERECO WHILE NOME = trimnome
      INPUT "INCLUIR O NOME ASSIM MESMO (Y/N) ?" TO oktoadd
   ENDIF

   IF # = 0 .OR. oktoadd
      APPEND BLANK
      STORE mnumcln + 1 TO mnumcln
      SET FORMAT TO Menu2-1
      READ NOUPDATE
      REPLACE NOME WITH trimnome, DATA WITH DATE(),;
      NUMCLN WITH STR(mnumcln,6)
      SET FORMAT TO SCREEN
      STORE NUMCLN+"  "+NOME+" "+ENDERECO TO Ultentr
   ENDIF
ENDDO
RELEASE inclreg,mnome,oktoadd,trimnome
RETURN
```

Programa 2. MENU2-1.CMD/.CMD/.PRG/ Inclui registros no arquivo de clientes

Nós a utilizamos para aumentar ao máximo nossa chance de encontrar registros duplicados. Suponhamos, por exemplo, que tivéssemos um registro chama LOPES S.A. e tivéssemos inserido LOPES na variável *mnome*. Se fizéssemos uma busca usando *mnome* não conseguiríamos identificar LOPES S.A. como uma possível duplicação. Por quê? Porque estaríamos fazendo uma comparação de todos os 30 caracteres. A abreviação S.A. não é o mesmo que quatro espaços em branco. Com TRIM faríamos a comparação somente dos cinco primeiros caracteres (e o tamanho do LOPES).

A instrução FIND *&trimnome* significa encontrar o registro idêntico ao conteúdo da variável de memória *trimnome*.

Neste ponto, temos novamente duas possibilidades. Ou encontramos uma possível duplicação de registro ou não. Se encontramos, o número de registro será maior que zero. Caso não, o número de registro será zero.

Se acharmos tratar-se de uma duplicação, limparemos a tela e forneceremos o NOME e o ENDEREÇO de duplicações possíveis. O comando INPUT possibilita perguntar ao usuário se deseja ou não inserir o registro. Utilizamos INPUT porque sua posição será a primeira linha da tela, abaixo do último item apresentado. Não sabemos com antecedência onde estará esse item. O comando INPUT pode ser usado para inserir tanto valores lógicos como números.

Se não encontrarmos uma duplicação ou se o usuário decidir inserir o item assim mesmo, teremos de acrescentar ao banco de dados um registro em branco e somar 1 ao número armazenado na variável *mnumcln*. Esse será o número de cliente que nosso cliente receberá. Agora, com a instrução SET FORMAT TO MENU2-1, acionamos o arquivo de formatação MENU 2-1.FMT, apresentado na Figura 3. Após a emissão do comando READ NOUPDATE, obtemos uma tela semelhante à Tela 2. Observe que "NOME", "DATA" e "NUM.CLIENTE" também estão na tela mas não podem ser modificados.

Por que usamos NOUPDATE quando na verdade fornecemos dados que afetam os arquivos de índice? Porque já na linha seguinte usaremos o comando REPLACE. Esse comando faz com que os índices se atualizem e não há necessidade de fazê-lo duas vezes.

Item 2 de Menu Editar Registros (MENU2-2.CMD/.PRG)

O que queremos que o programa faça? Queremos poder inserir ou o nome do cliente ou o número de identificação e que o computador escolha o registro correto para edição. Desde que não tenhamos um número de cliente que comece por um número maior que 100.000, será fácil obtermos isso.

Se inserirmos um nome ou número não existente, queremos ser informados do erro cometido e ter a possibilidade de modificar nossa entrada.

Se houver mais de um registro no banco de dados com o mesmo número de cliente, queremos que seus nomes e endereços sejam apresentados na tela e também queremos um método de selecionar qual (ou nenhum) deles vamos editar.

Precisaremos de um indicador, para saber exatamente onde estamos, caso desviemos a atenção dos registros. Todo ou parte do último registro editado deverá permanecer na tela quando selecionarmos o registro seguinte a ser editado.

Como montaremos o programa? Para inserir o programa, o primeiro passo será armazenar 30 espaços em branco – o tamanho do campo NOME – na variável de memória *mnome*. A seguir, receberemos uma indicação para digitarmos ou o nome ou o número do cliente. Observe que novamente utilizamos a opção NOUPDATE .

Aqui também temos duas possibilidades a considerar. Inserimos ou não algum dado? Se não o fizermos, deveremos voltar ao menu.

Se inserirmos um nome ou um número, desejaremos que o computador o descubra.

Nosso esquema para seleção entre as duas possibilidades se apóia no fato de não haver nome de cliente iniciando por um número maior que 100.000. Utilizamos a função VAL para testar *mnome*. Se VAL (mnome) for maior que 100.000, então a inserção será um número de cliente. Caso contrário, será um nome de cliente. A função VAL se encerra com o primeiro caractere que não for um número ou um espaço em branco, por exemplo, VAL (10B16) é 10.

```
* Programa ...........: MENU2-1.FMT
* Notas ..............: Esta e tela de formatacao para inclusao de
* ....................  registros no arquivo de clientes.

@ 3,28 SAY "DISTRIBUIDORA FERREIRA"
@ 6,1  SAY "NOME DO CLIENTE  : "+mnome
@ 6,50 SAY "NUMERO DO CLIENTE: "+STR(numcln,6)
@ 8,1  SAY "ENDERECO " GET ENDERECO PICTURE '!!!!!!!!!!!!!!!!!!!!!!!!!'
@ 10,1 SAY "CIDADE ......... " GET CIDADE PICTURE '!!!!!!!!!!!!!!!!!!!'
@ 10,50 SAY "ESTADO " GET ESTADO PICTURE '!!'
@ 12,1 SAY "CODIGO CEP ..... " GET CEP PICTURE '99999'
@ 8,50 SAY "CLIENTE DESDE   : " + DATE()
@ 14,1 SAY "TELEFONE ....... " GET TEL PICTURE "(999)-999-9999"
@ 16,1 SAY "A ATENCAO DE .... " GET ATN PICTURE :
'!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!'
```

Figura 2. MENU2-1.FMT Arquivo de formatação para incluir registros de clientes

```
                    DISTRIBUIDORA FERREIRA

NOME DO CLIENTE  :VOLUMES TECNICOS LTDA.      NUMERO DO CLIENTE: 120213

ENDERECO :                                    CLIENTE DESDE: 22/01/51

CIDADE ......... :                            ESTADO :

CODIGO CEP ..... :

TELEFONE .........: (   )-    -     :

A ATENCAO DE .....:
```

Tela 2. Tela para inclusão de registros de clientes


```
* Programa ...........:MENU2-2.CMD (ou MENU2-2.PRG)
* Obs. ...............:Este programa edita os registros do Arquivo de
* ....................Clientes.

STORE T TO editreg
ERASE
DO WHILE editreg
STORE "                                  " TO mnome
@ 21,1 SAY "ENTRAR O NOME OU O NUMERO DO CLIENTE" GET mnome
READ NOUPDATE
IF MNOME=''
   STORE F TO editreg
   LOOP
ENDIF
IF VAL(mnome) > 100000
   SET INDEX TO Numcln,Cliente,CEP
   STORE $(mnome,1,6) TO findcln
ELSE
   SET INDEX TO Cliente,Numcln,Cep
   STORE !(TRIM(mnome)) TO findcln
ENDIF
FIND &findcln
IF # = 0

   @ 22,1 SAY findcln+' NAO ESTA NO BANCO DE DADOS'
   LOOP
ENDIF
@ 22,0
IF # > 0
   STORE # TO posicao
   IF .NOT. NUMCLN = findcln
      SKIP
      IF NOME= findcln .AND. .NOT. EOF
         ERASE
         @ 1,1 SAY 'MULTIPLOS REGISTROS PARA '+findcln
         SKIP -1
         DISPLAY NOME,ENDERECO WHILE NOME= findcln
         INPUT "ENTRAR NUMERO A PARTIR COLUNA DA ESQUERDA - 0 PARA ABORTAR":
         TO posicao
      ENDIF caso o nome = findcln
   ENDIF caso selecionado pelo nome do cliente
   IF posicao > 0
      GO posicao
      SET FORMAT TO Menu2-2
      READ
      CLEAR GETS
      SET FORMAT TO SCREEN
      ENDIF para numero valido de registro
   ENDIF para caso de ser encontrado o registro
ENDDO
RELEASE editreg,mnome,posicao,findcln
RETURN
```

Programa 3. MENU2-2.CMD/.PRG Edita os registros de clientes

O resultado do teste de VAL permite-nos escolher a ordem dos arquivos de índice. Ele também indica como manipular a variável. Se for o número de identificação de um cliente, vamos armazenar os seis primeiros caracteres de mnome em uma nova variável, *findcln*. Caso contrário, vamos armazenar o conteúdo de *mnome* – com todos os espaços em branco que se seguirem eliminados e em caracteres maiúsculos – em *findcln*.

A função ! converte para maiúsculas. A função TRIM elimina, em uma variável, os espaços em branco excedentes. Combinamos as duas funções em uma operação única, como fizemos com o programa para inclusão de registros.

Encontraremos o registro desejado, ou não. Caso não, digitaremos uma mensagem na Linha 22 que nos mostrará o que estávamos procurando. Então fazemos um loop retornando ao início e tentamos uma outra vez. A mensagem tem o objetivo de ajudar o usuário a diagnosticar o motivo por que o computador não encontrou o registro desejado.

Se o registro for encontrado, o programa deverá considerar duas possibilidades: nome e número. Se a entrada for um nome, teremos de fazer um teste para registros múltiplos. Primeiramente verificamos se é um nome, comparando *findcln* ao NUMCLN. Não haverá correspondência se *findcln* for um nome. Assim, se não houver correspondência, fazemos um teste para registros múltiplos. Não há teste para registros múltiplos se tivermos digitado um número, porque cada número de identificação do cliente é exclusivo. Em ambos os casos, armazenamos o número de registro do registro encontrado em uma variável *posição*.

Para verificar se há registros múltiplos, saltamos para o registro seguinte. Agora comparamos o conteúdo de *findcln* ao conteúdo de NOME. Observe que também testamos o fim-de-arquivo. Se tivéssemos digitado o fim-de-arquivo, ainda estaríamos no mesmo lugar em que estávamos antes do salto e *findcln* seria igual a NOME.

Se houver registros múltiplos, limpamos a tela e apresentamos NÚMERO DE REGISTRO, NOME e ENDEREÇO. A indicação, na coluna da esquerda, para digitar o número está na verdade dizendo: inserir (entrar) o número de registro. Há, entretanto, um modo de sair: entrar um zero, um número de registro não existente.

Teste se o número de registro é válido com IF *posição* > 0. Se o teste confirmar a existência do registro, execute GO *posição*. GO *posição* significa seguir para o número de registro armazenado em *posição*.

A tela usada para editar os registros é representada por nossa Tela 3. A Figura 3 mostra o arquivo de formatação para criar essa tela. Observe que aqui executamos diretamente um comando READ. Isso significa que se modificarmos algo nos bancos de dados, haverá uma pequena demora enquanto os três arquivos de índice estiverem sendo atualizados.

A apresentação na tela é abandonada, quando indicamos ao usuário que deve inserir o nome ou número seguinte de cliente. Esse é nosso marcador de lugar. Para evitar que o usuário mova o cursor até uma das áreas de apresentação de campo, utilizamos CLEAR GETS. Esse comando "cancela" as operações para campos do comando READ e a entrada de variáveis que precedem o comando CLEAR GETS.

Item 3 de Menu Eliminar Registros (MENU2-3.CMD/.PRG)

A eliminação de registros é uma questão importante. Uma vez apagado, está definitivamente perdido.

Por esse motivo, a eliminação dos registros é feita em dois estágios. O comando DELETE marca o registro para exclusão e o comando PACK o elimina efetivamente. Quando utilizamos

o comando PACK, todos os registros após o excluído são renumerados (a atribuição de números de registro do dBASE II é refeita) e os arquivos de índice são reindexados. Essa operação exigirá tempo em um banco de dados extenso. De modo geral é aconselhável excluir vários registros por vez, se possível numa hora do dia em que haja pouca atividade.

```
                    DISTRIBUIDORA FERREIRA

NOME DO CLIENTE  : VOLUMES TECNICOS LTDA.        NUMERO DO CLIENTE: 120213
ENDERECO : Rua das Acacias, 100 :                CLIENTE DESDE: 22/01/81
CIDADE ..... : CIDADE INDUSTRIAL :               ESTADO: SP
CODIGO CEP ..... :91232:
TELEFONE ..... :(213)-521-6578:
A ATENCAO DE .... : Paulo R. Medeiros :
```

Tela 3. Tela para edição dos registros de clientes

```
* Programa .........: MENU2-2.FMT
* Notas ............: Este programa e o arquivo de formatacao para
* ................... modificar registros de clientes.
@ 3,28 SAY "DISTRIBUIDORA FERREIRA"
@ 6,1 SAY "NOME DO CLIENTE" GET NOME PICTURE '!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!'
@ 6,50 SAY "NUMERO DO CLIENTE: "+NUMCLN
@ 8,1 SAY "ENDERECO " GET ENDERECO PICTURE '!!!!!!!!!!!!!!!!!!!!!!!!!!!'
@ 10,1 SAY "CIDADE ........." GET CIDADE PICTURE "!!!!!!!!!!!!!!!!!!!!!!!"
@ 10,50 SAY "ESTADO " GET CEP PICTURE "!!"
@ 12,1 SAY "CODIGO CEP ....." GET CEP PICTURE "99999"
@ 8,50 SAY "CLIENTE DESDE "+DATA
@ 14,1 SAY "TELEFONE ........" GET TELEFONE PICTURE "(999)-999-9999"
@ 16,1 SAY "A ATENCAO DE ...." GET ATN PICTURE '!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!'
```

Figura 3. MENU2-2.FMT Arquivo de formatação para modificar registros de clientes

```
* Programa .........: MENU2-3.CMD (ou Menu2-3.PRG)
* Obs. .............: Este programa exclui registros do arquivo de clientes.

SET INDEX TO Cliente,Numcln,Cep
STORE T TO delereg
STORE F TO dreg,pergunta2
ERASE
DO WHILE delereg
   STORE "                                  " TO mnome
   @ 21,1 SAY "FORNECER O NOME PARA EXCLUSAO" GET mnome
   READ NOUPDATE
   IF mnome = '
      STORE F TO delereg
      LOOP
   ENDIF
   STORE '(mnome) TO mnome
   FIND &mnome
   IF #. = 0
      ERASE
      @ 22,1 SAY TRIM(mnome)+"NAO SE ENCONTRA NO BANCO DE DADOS"
      LOOP
   ENDIF
   @ 22,0

   IF # > 0
      STORE # TO posicao
      SKIP
      IF NOME=mnome .AND. .NOT. EOF
         ERASE
         @ 1,1 SAY "REGISTROS MULTIPLOS PARA "+mnome
         SKIP -1
         DISPLAY NOME,ENDERECO WHILE NOME=mnome
         SKIP -1
         STORE # TO parar
         STORE T TO teste
         DO WHILE teste
            INPUT "ENTRAR NUMERO A PARTIR COLUNA DA ESQUERDA - 0 PARA ABORTAR";
            TO posicao
            IF posicao = 0 .OR. (posicao >= inicio .AND. posicao = parar)
               STORE F TO teste
            ENDIF
         ENDDO
      ENDIF
      IF posicao > 0
         GO posicao
         SET FORMAT TO SCREEN
         INPUT "APAGAR ESTE REGISTRO (Y/N)?" TO pergunta
            IF pergunta
               DELETE
               STORE T TO dreg
            ELSE
               RECALL
            ENDIF
         ENDIF
      ENDIF
ENDDO
IF dreg
   ERASE
   DISPLAY OFF NOME,ENDERECO FOR *
   INPUT "ENCERRAR A EXCLUSAO DESSES REGISTROS (Y/N)?" TO pergunta2
   IF pergunta2
      PACK
   ENDIF
ENDIF
RELEASE editreg,mnome,posicao,dreg,pergunta
RETURN
```

Programa 4. MENU2-3.FMT Arquivo de formatação para excluir registros de clientes


```
* Programa .........: MENU2-3.FMT
* Obs. .............: Este programa e o arquivo de formatacao para excluir
* ................... registros.

@ 3,28 SAY "DISTRIBUIDORA FERREIRA"
@ 6,1 SAY "NOME DO CLIENTE  : " + NOME
@ 6,50 SAY "NUMERO DO CLIENTE : "+NUMCLN
@ 8,1 SAY "ENDERECO          " + ENDERECO
@ 10,1 SAY "CIDADE ..........: "+CIDADE
@ 10,50 SAY "ESTADO : "+ESTADO
@ 12,1 SAY "CODIGO CEP ......: "+CEP
@ 8,50 SAY "CLIENTE DESDE : " + DATA
@ 14,1 SAY "TELEFONE ........: "+TEL
@ 16,1 SAY "A ATENCAO DE ....: "+ATN
```

Figura 4. MENU2-3.CMD/.PRG Exclui os registros de clientes

Ao chamar um registro para exclusão, queremos vê-lo na tela e também que nos seja perguntado se realmente desejamos excluí-lo. Também queremos recuperar (RECALL) um registro. O comando RECALL elimina o sinal de exclusão.

Uma vez estando o banco de dados posicionado no registro que desejamos excluir, apresentamo-lo por meio do arquivo de formatação da Figura 4. Perguntamos ao operador se esse registro deve ser excluído. A resposta é armazenada na variável *pergunta*. Se *pergunta* for verdadeira, marcamos o registro para exclusão e armazenamos um T na variável *dreg*.

Após identificar todos os registros para exclusão, deixamos o DO entrar em loop. Se *dreg* for verdadeira, apresentamos o nome e o endereço em cada registro marcado para exclusão. Então, perguntamos mais uma vez se o usuário realmente quer apagar os registros. Se ele o quiser, eliminaremos todos esses registros por meio do comando PACK. Caso contrário, retiraremos os sinais de exclusão por meio do comando RECALL ALL.

Capítulo Seis

LISTAS E ETIQUETAS

Um dos usos mais populares de um sistema de banco de dados é a impressão de listas e etiquetas de endereçamento. Geralmente, as listas de endereçamento são arquivos de clientes, montadas com a finalidade de relatórios mensais, publicidade, cartas circulares etc. Como o Correio tem preferência por clientes que organizam a correspondência pelo código CEP, o banco de dados deve pelo menos ser indexado pelo CEP. Um método mais elaborado será o de indexar a lista de endereçamento pela ordem alfabética do CEP, como mostramos no Capítulo 5; isto é, os códigos CEP são ordenados e os nomes para cada um deles ficam na ordem alfabética.

Usaremos o Menu 3, o terceiro item de nosso Menu Principal, para gerar uma série de programas para imprimir listas ou etiquetas a partir de nosso arquivo de clientes.

No Programa 1, usamos o sinal $ para identificar um número de Linha 2 como a nova base. Queremos que a segunda linha apresentada esteja duas linhas abaixo da primeira (na linha 4). Os sinais $ + 2 ficam então dispostos em espacejamento duplo; $ + 1, fica em espacejamento simples. Se fôssemos introduzir uma linha na tela, não precisaríamos renumerar as linhas. Isso também se aplica às posições de coluna.

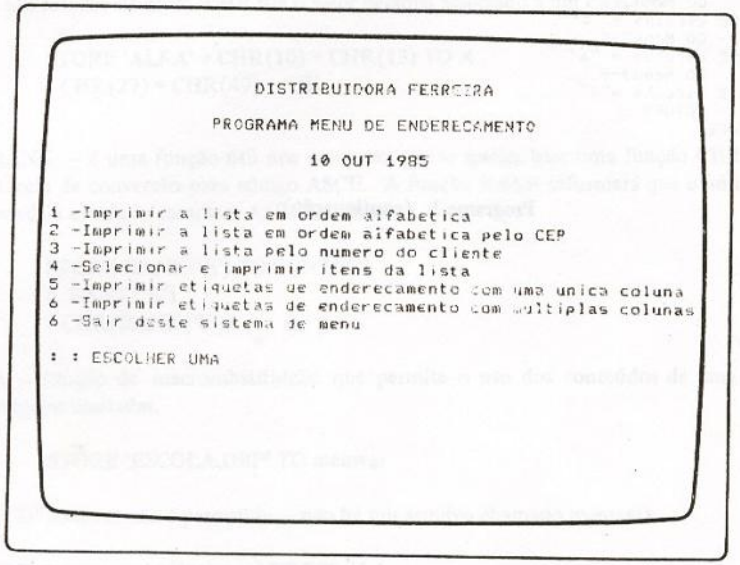
Observe que as escolhas 1 e 3 selecionam o programa MENU3 – 1, que imprime a lista de clientes pela ordem alfabética ou pelo número de identificação do cliente. Isso não está indicado na tela.

Item 2 de Menu Imprimir Listas de Endereçamento pela Ordem Alfabética ou pela Ordem de Número de Cliente (MENU3 – 1.CMD/.PRG)

Desejamos uma listagem em ordem alfabética de todos os clientes em nosso arquivo, como a da Figura 1. O programa para preparar o relatório é apresentado como Programa 2. O item 3

de menu, que imprime a lista pelo número dos clientes, é feito exatamente como o Item 1 de menu, assim ambos utilizam o Programa 2.

Alguns clientes em nossa lista não têm entradas para o campo ATN.; desse modo, algumas das entradas têm três linhas e outras, quatro. Não desejamos uma linha em branco entre o nome e o endereço. Se examinar o Programa 2, você notará que há uma linha (IF .NOT. ATN. = ' ') que possibilita uma entrada, se necessário.



Tela 1. Tela do menu de endereçamento

```
* Programa ........: MENU3-CMD (ou MENU3-PRG)
* Obs .............: Este  e o programa menu de enderecamento

USE Clientes
DO WHILE T
   STORE " " TO escolha
   ERASE
   @   2,30 SAY "DISTRIBUIDORA FERREIRA"
   @ $+2,29 SAY "PROGRAMA MENU DE ENDERECAMENTO"
   @ $+2,32 SAY cdata
   @ $+2,16 SAY "1 -Imprimir a lista em ordem alfabetica"
   @ $+1,16 SAY "2 -Imprimir a lista em ordem alfabetica pelo CEP"
   @ $+1,16 SAY "3 -Imprimir a lista pelo numero de cliente"
   @ $+1,16 SAY "4 -Selecionar ee imprimir itens da lista"
   @ $+1,16 SAY "5 -Imprimir etiquetas de endereçamento com uma unica coluna"
```

Programa 1. MENU3.CMD/.PRG Programa menu de endereçamento

```
   @ $+1,16 SAY "6 -Imprimir etiquetas de endereçamento com multiplas colunas"
   @ $+1,16 SAY "0 -Sair deste sistema de menu"
   @ $+2,20 SAY "ESCOLHA UMA"
   @ $,15 GET escolha
   DO CASE
      CASE escolha = "1"
           DO Menu3-1
      CASE escolha = "2"
           DO Menu3-2
      CASE escolha = "3"
           DO Menu3-3
      CASE escolha = "4"
           DO Menu3-4
      CASE escolha = "5"
           DO Menu3-5
      CASE escolha = "6"
           DO Menu3-6
      CASE escolha ="0"
           RETURN
   ENDCASE
ENDDO
```

Programa 1. (continuação)

```
        COPIA IMPRESSA DA LISTA DE ENDERECAMENTO DESDE 15/10/84

102003          LIVRARIA SIENG                          (213) 555-5555
                Av. Politeira, 123
                Sao Paulo - SP          91222

100001          VOLUMES TECNICOS                        (213) 521-6578
                Paulo R. Medeiros
                Praca dos Canarios,2123
                CIDADE INDUSTRIAL, SP       91232
```

Figura 1. Disposição da lista impressa


```
* Programa .........:MENU3-1.CMD (ou MENU3-1.PRG)
* Obs ..............:Este programa imprime lista de clientes em ordem
* ...................alfabetica ou pelo numero de identificacao dos clientes.

ERASE
IF escolha ="1"
   SET INDEX TO Clientes
ELSE
   SET INDEX TO Numcln
ENDIF
GO TOP
SET FORMAT TO PRINT
SET MARGIN TO 15
STORE 0 TO pagina
STORE "COPIA IMPRESSA DA LISTA DE ENDERECAMENTO DESDE "+cdata TO titpag
DO WHILE .NOT. EOF
   @ 6,25 SAY titpag
   STORE 8 TO linha
   DO WHILE linha<55 .AND. .NOT. EOF
      @ linha,1 SAY NUMCLN
      @ linha,15 SAY NOME
      @ linha,50 SAY TEL
      IF .NOT. ATN = ''
         STORE linha+1 TO linha
         @ linha,15 SAY ATN
      ENDIF
      @ linha+1,15 SAY ENDERECO
      @ linha+2,15 SAY TRIM(CIDADE) + ", " + ESTADO
      @ linha+2,42 SAY CEP
      STORE linha+4 TO linha
      SKIP
   ENDDO
   STORE pagina+1 TO pagina
   @ 62,40 SAY STR(pagina,3)
   EJECT
ENDDO
SET FORMAT TO SCREEN
SET MARGIN TO 0
RELEASE pagina,titpag,linha
```

Programa 2. MENU3-1.CMD/.PRG Imprimir a lista alfabeticamente ou pelo número do cliente

O comando @x,y SAY pode ser usado para impressão, usando-se SET FORMAT TO PRINT. Temos que tomar algum cuidado ao usar essa técnica. Os itens devem ser impressos seqüencialmente em cada linha, as quais devem ser impressas em seqüência em cada página. Se você estiver imprimindo um campo de 30 caracteres que comece na Coluna 1, não inicie o item seguinte dessa linha antes da Coluna 31. Se der um comando para imprimir em uma posição na página "anterior" à posição atual da cabeça de impressão, a impressora parecerá ter enlouquecido. Ela irá à posição solicitada, mas na página seguinte. Embora o comando possa fazer páginas de formatação com muita facilidade, ele não pode controlar a impressora. Se cometer um erro dentro de um loop DO, você desperdiçará muito papel em pouco tempo.

Observe que depois de selecionado o índice, utilizamos o comando GO TOP, que posiciona o banco de dados no primeiro registro indicado pelo arquivo de índice (o primeiro registro lógico).

SET FORMAT TO PRINT fará com que a apresentação dos comandos @ x,y SAY seja direcionada para a impressora. Eles não aparecerão na tela.

Neste exemplo, temos um loop DO alojado em outro loop DO. O loop externo, DO WHILE .NOT. EOF, é executado até atingirmos a marca de fim-de-arquivo.

Ao inserir o loop externo, imprimimos o título da página e estabelecemos o número da linha de início do loop interior. Executamos este loop até o contador de linha atingir 55 ou até chegarmos à marca de fim-de-arquivo. A seguir, saímos desse loop, acrescentamos uma unidade ao número da página e o imprimimos, emitindo a página. Se não tivermos atingido o final-de-arquivo, voltamos ao início do loop exterior.

Um erro comum de programação está em esquecer o .AND. NOT. EOF do loop interno. Se o fizermos, o que acontecerá se chegarmos ao fim-de-arquivo antes do contador de linha atingir 55? O último registro será impresso seguidamente até atingir 55. Se suspendermos ambos os loops, a impressão continuará até ficarmos sem papel.

O comando SKIP avança no banco de dados um registro por vez a cada execução do loop DO. O número de linha aumenta em quatro unidades, a cada passagem pelo loop.

Nós imprimimos a linha ATN. com a instrução IF .NOT. ATN " ", se não a quisermos em branco. Precisamos aumentar o contador de linha em uma unidade. As instruções de impressão após isso fazem referência por estágios ao número armazenado no contador de linha.

A última atividade antes de cada página ser emitida será imprimir o número de página. Neste exemplo imaginamos que não haverá mais de 999 páginas. Observe que usamos a função STR para imprimir o número de página. O comando SAY deveria ser usado somente com um campo de caracteres ou com uma variável.

No dBASE II, o comando RETURN é tecnicamente opcional, embora o uso de RETURN seja um procedimento aconselhável. O dBASE II identificará a marca EOF (todos os arquivos têm marcas EOF) como sendo um comando RETURN. Se efetivamente usarmos RETURN, poderemos digitar após o comando a quantidade de texto e de comentário que quisermos, sem afetar a execução do programa.

Esse programa é curto, simples e fácil de ler. Seu único inconveniente é imprimir toda a lista a cada vez que é utilizado.

Item 2 de Menu Imprimir a Lista de Endereçamento em Ordem Alfabética e pelo CEP (MENU3.2.CMD/.PRG)

O programa 3 é muito semelhante ao programa anterior.

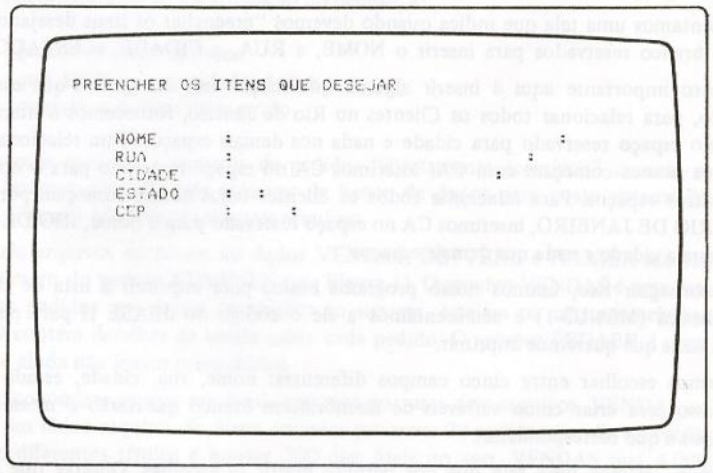
Nosso arquivo de índice CEP foi criado por meio do comando INDEX, INDEX ON CEP + NOME TO CEP, e depois SET INDEX TO CEP.

Novamente iniciamos pela escolha do índice desejado, posicionando o banco de dados na parte superior.


```
* Programa .........:MENU3-2.CMD (ou MENU3-2.PRG)
* Obs ..............: Este programa imprime a lista de endereçamento
* ................... pela ordem alfabética de CEP.

SET INDEX TO Cep
GO TOP
SET FORMAT TO PRINT
SET MARGIN TO 15
STORE 0 TO pagina
DO WHILE .NOT. EOF
   @ 6,25 SAY "LISTA DE ENDEREÇAMENTO PELO CEP"+CEP+"  DESDE "+DATE()
   STORE CEP TO mcep
   STORE 8 TO linha
   DO WHILE CEP = mcep .AND. linha<55 .AND. .NOT. EOF
      @ linha,1 SAY NUMCLN
      @ linha,15 SAY NOME
      IF .NOT. ATN = "       "
         STORE linha+1 TO linha
         @ linha,15 SAY ATN
      ENDIF
      @ linha,50 SAY TEL
      @ linha+1,15 SAY ENDERECO
      @ linha+2,15 SAY TRIM(CIDADE)+", "+ESTADO
      @ linha+2,42 SAY CEP
      STORE linha+4 TO linha
      SKIP
   ENDDO
   STORE pagina+1 TO pagina
   @ 62,40 SAY STR(pagina,3)
   EJECT
ENDDO
SET MARGIN TO 0
SET FORMAT TO SCREEN
RELEASE pagina,linha,mcep
RETURN
```

Programa 3. MENU3-2.CMD/.PRG Imprimir a lista por ordem alfabética e pelo CEP



Tela 2. Tela para imprimir registros escolhidos

Optamos por incorporar o CEP ao título da página, assim realizamos a formatação do título no interior do loop DO.

O armazenamento do CEP na variável de memória CEP permite-nos garantir que cada novo código CEP automaticamente comece em uma nova página. O loop interior se encerrará se encontrarmos a marca de fim-de-arquivo, ou se a linha de contagem passar de 55 ou, ainda, se o CEP no campo CEP não for o mesmo que o da variável de memória cep.

Observe que o teste para a *linha* > 55 ocorre somente no início de cada passagem pelo loop anterior. Ela poderá ultrapassar 55 durante o loop. Esperamos que isso aconteça. Essa estrutura específica foi escolhida para evitar a possibilidade de algum registro ficar dividido entre duas páginas.

O item importante que este programa examina é o uso da variável mcep. Não precisamos conhecer os códigos CEP que estão no banco de dados. Podemos usar os próprios dados como chave para controlar o programa.

Item 4 de Menu Imprimir Registros Escolhidos (MENU3-4.CMD/.PRG)

Nos exemplos de menu até agora vistos, fizemos a impressão de toda a lista de clientes, quer precisando disso quer não. Se nossa lista for pequena, sua impressão não causará problemas. Se ela tiver 10.000 clientes, isso se tornará inviável.

No exemplo que damos a seguir, imprimiremos partes escolhidas da lista pela ordem alfabética. Queremos um programa que nos permita especificar exatamente o que desejamos imprimir. O Programa 4 nos mostra como montar um tal sistema.

Apresentamos uma tela que indica quando devemos "preencher os itens desejados". Temos espaços em branco reservados para inserir o NOME, a RUA, a CIDADE, o ESTADO e o CEP.

O ponto importante aqui é inserir alguma informação nos cabeçalhos que escolhermos. Por exemplo, para relacionar todos os Clientes no Rio de Janeiro, fornecemos o nome RIO DE JANEIRO no espaço reservado para cidade e nada nos demais espaços. Para relacionar todos os clientes cujos nomes começam com CA, inserimos CA no espaço reservado para o nome e nada mais nos outros espaços. Para relacionar todos os clientes cujos nomes começam por CA e que moram no RIO DE JANEIRO, inserimos CA no espaço reservado para o nome, RIO DE JANEIRO no espaço para a cidade e nada nos demais espaços.

Para conseguir isso, usamos nosso programa básico para imprimir a lista de clientes por ordem alfabética (MENU3-1) e acrescentamos a ele o código do dBASE II para possibilitar a escolha dos itens que queremos imprimir.

Queremos escolher entre cinco campos diferentes: nome, rua, cidade, estado e CEP. O primeiro passo será criar cinco variáveis de memória em branco que terão o mesmo tamanho que os campos a que correspondem.

A seguir, criamos uma tela que nos permite inserir as escolhas. Observe que novamente usamos a instrução READ NOUPDATE.

A seguir, eliminamos os espaços a mais e passamos nossas entradas para maiúsculas. É importante observar que quando usamos o comando TRIM com uma variável que só contém espaços em branco, ficamos com uma variável que tem um espaço em branco. Usamos esse espaço para verificar se inserimos algum dado. Se não, vamos para o fim do programa e retornarmos ao MENU3.

Usaremos o comando LOCATE FOR para fazer a nossa seleção. LOCATE FOR <expressão> começará no início do banco de dados e fará uma busca seqüencial no banco de dados até encontrar um registro que corresponda à expressão, ou que chegue ao fim-de-arquivo. LOCATE NEXT n FOR <expressão> iniciará a busca a partir do ponto em que você estiver no banco de dados. Em nosso exemplo, vamos estabelecer n = 65535, que é o número máximo de registros do dBASE II. Fazemos assim, porque se buscarmos o registro e não o encontrarmos, iremos ao final do banco de dados, chegando à marca de fim-de-arquivo. Se efetivamente encontrarmos o registro, não iremos querer retornar ao início e realizar nova busca.

```
* Programa ..........:MENU3-4.CMD (ou MENU3-4.PRG)

SET INDEX TO Clientes
GO TOP
ERASE
STORE "                                   " TO mnome
STORE "                                   " TO mrua
STORE "                                   " TO mcidade
STORE "  " TO mestado
STORE "     " TO mcep
ERASE
@ 1,10 SAY "PREENCHA OS ITENS QUE DESEJAR"
@ 3,10 SAY "NOME" GET mnome
@ 4,10 SAY "RUA " GET mrua
@ 5,10 SAY "CIDADE" GET mcidade
@ 6,10 SAY "ESTADO" GET mestado
@ 7,10 SAY "CEP " GET mcep
READ
STORE !(TRIM(mnome)) TO mnome
STORE !(TRIM(mrua)) TO mrua
STORE !(TRIM(mcidade)) TO mcidade
STORE !(TRIM(mestado)) TO mestado
STORE !(TRIM(mcep)) TO mcep
IF mnome+mrua+mcidade+mestado+mcep = "     "
   RETURN
ELSE
   STORE "LOCATE NEXT 65535 FOR " TO comando
   IF .NOT. mnome=" "
      STORE comando+"NOME=mnome .AND. " TO comando
      FIND &mnome
   ENDIF
   IF .NOT. mrua=" "
      STORE  comando+"mrua=ENDERECO .AND. " TO comando
   ENDIF
   IF .NOT. mcidade=" "
      STORE  comando+ "CIDADE=mcidade .AND. " TO comando
   ENDIF
   IF .NOT. mestado=" "
      STORE comando+"ESTADO=mestado .AND. " TO comando
   ENDIF
   IF .NOT. mcep=" "
      STORE comando +"CEP=mcep .AND. " TO comando
```

Programa 4. MENU3-4.CMD/.PRG Impressão de registros escolhidos

```
  ENDIF
  STORE $(comando,1,LEN(comando)-5) TO comando
  &comando
  SET FORMAT TO PRINT
  STORE 0 TO pagina
  SET MARGIN TO 15
  DO WHILE .NOT. EOF
     @ 6,15 SAY DATE() + "LISTA DE ARQUIVO DE CLIENTES ESPECIAIS"
     STORE 10 TO linha
     DO WHILE .NOT. linha<55 .AND. .NOT. EOF
        @ linha,1 SAY NUMCLN
        @ linha,15 SAY NOME
        @ linha,50 SAY TEL
        IF .NOT. ATN
           STORE linha+1 TO linha
           @ linha,15 SAY ATN
        ENDIF
        @ linha+1,15 SAY ENDERECO
        @ linha+2,15 SAY TRIM(CIDADE)+", "+ESTADO
        @ linha+2,42 SAY CEP
        STORE linha+4 TO linha
        SKIP
        &comando
     ENDDO
     STORE pagina+1 TO pagina
     @ 62,40 SAY STR(pagina,3)
     EJECT
  ENDDO
ENDIF
SET FORMAT TO SCREEN
RELEASE pagina,linha,comando,mnome,mrua,mcidade,mestado,mcep
RETURN
```

Programa 4. (continuação)

Agora armazenamos o núcleo de nosso comando na variável COMANDO. A seguir, reunimos a <expressão>, usando a série de comandos IF/ENDIF. Observe que cada elemento adicional incorporado à expressão se encerra com ".AND.". Observe, também, que, se tivéssemos inserido algo em seleção para NOME, teríamos executado um comando FIND nesse item. Isso nos posicionou no primeiro registro que tem a possibilidade de atender a nossos critérios de busca.

Ao encerrar com os comandos IF, removemos o ".AND." final. A sintaxe do dBASE II não admite o encerramento de um comando por meio de ".AND.". Ele imprimirá uma mensagem de erro e abortará o programa. Para eliminar os cinco últimos caracteres, utilizamos a função LEN no interior da operação de *substring*:

```
STORE $(comando,1,LEN(comando)–5)TO
■ comando
```

Isso é lido como: armazenar os primeiros (n-5) caracteres da variável COMANDO novamente em COMANDO.

Agora iniciamos a nossa busca fora do loop, usando &COMANDO. Isso significa literalmente "executar o conteúdo de COMANDO". Isso foi colocado fora do loop para posicionarmos no primeiro registro possível. Se não houvesse registros, teríamos teclado o sinalizador de fim-de-arquivo. Um inconveniente é que o esquema só será totalmente confiável se tivermos menos de 65535 registros.

O pacote dBASE II RunTime™ não admitirá que você use uma linha de comando que se inicie por um "&", como o &COMANDO, neste programa exemplo, embora isso não ocorra com o dBASE II padrão. Essa é uma das poucas diferenças na programação do dBASE II e do RunTime. Para usar a versão RunTime, teríamos de eliminar a palavra LOCATE de "comando". A linha de comando em questão ficaria LOCATE&COMANDO.

Agora inserimos nossa rotina de impressão da lista básica. Observe que saltamos (SKIP) logo antes de retornar à rotina de busca. Por quê? Se estivermos, por exemplo, no registro 10, e usarmos o comando LOCATE NEXT 10 FOR (exp) e a expressão for atendida pelo registro 10, ainda estaremos no registro 10. Para continuar nossa busca, precisamos avançar um registro a cada vez que executarmos um comando LOCATE.

Por que o fizemos dessa forma? Poderíamos ter usado um IF sem expressão e colocado um IF <expressão> logo após o DO WHILE do loop interior. O ENDIF ficaria imediatamente antes do SKIP. Este é o modo mais comum de executar nossa tarefa. Utilizamos o esquema LOCATE porque é muito mais rápido.

O comando LOCATE é essencialmente uma rotina de busca da linguagem assembly. Para utilizar a abordagem IF/ENDIF, teríamos uma rotina de busca do dBASE II, que exigiria passar pelo loop a cada registro no banco de dados. O loop leva de 30 a 50 milissegundos, em cada registro, mais o tempo necessário para efetivamente ler os registros em disco. Se tivermos 10.000 registros, essa tarefa levará de 300.000 a 500.000 milissegundos. Com LOCATE reduzimos a aproximadamente o tempo necessário para ler o disco, uma economia de 300 a 500 segundos.

Se, para você, for importante a velocidade geral da execução, ao programar em dBASE II, use o comando de nível mais alto que puder.

Item 5 de Menu Imprimir Etiquetas com uma Única Coluna (MENU3-5.CMD/.PRG)

Quase todas as empresas imprimem etiquetas. É uma tarefa que a maioria das pessoas abomina, mas é natural para o computador.

As etiquetas podem ser de vários tamanhos, ocupando de uma a cinco colunas por folha. As folhas ou etiquetas têm geralmente 12 polegadas de comprimento em vez do comprimento padronizado de página, de 11 polegadas.

Vamos utilizar dois tipos de etiquetas em nossa empresa. A primeira, de coluna única, é a mais fácil de se lidar.

As etiquetas que vamos usar têm 1,5 por 4 polegadas. Como nossa impressora imprime dez caracteres por polegada e seis linhas por polegada, temos 40 caracteres disponíveis no sentido da largura. O maior dos campos que estamos imprimindo tem 30 caracteres, de modo que ficamos com uma margem de cinco caracteres (1/2 polegada) de cada lado da etiqueta. Seis linhas por polegadas possibilitam imprimir até nove linhas em cada etiqueta. Vamos usar quatro linhas no máximo.

A linha na qual as etiquetas são impressas é determinada pelo arquivo de índice escolhido.

Escolhemos a margem esquerda com 1/2 polegada por meio do comando SET MARGIN TO 5.

```
* Programa .........:MENU3-5.CMD (ou MENU3-5.PRG)
* Obs ..............:Este programa imprime Etiquetas de uma unica coluna.

SET INDEX TO Cep
GO TOP
SET PRINT ON
SET MARGIN TO 5
DO WHILE .NOT. EOF
   ?
   ?
   ? NOME
   IF .NOT. ATN = "     "
      ? ATN
   ENDIF
   ? ENDERECO
   ? TRIM(CIDADE)+", "+ESTADO+"  "+CEP
   IF ATN = "     "
      ?
   ENDIF
   ?
   ?
   ?
   SKIP
ENDDO
SET PRINT OFF
RETURN
```

Programa 5. MENU3-5.CMD/.PRG Programa de impressão de etiquetas simples

Observe que utilizamos SET PRINT ON de preferência a SET FORMAT TO PRINT. Este último pode ser usado somente com o comando @x,y SAY.

Utilizamos o comando ? para imprimir os itens que desejamos. O sinal de interrogação imprimirá os itens relacionados após ele; ? produz um retorno de carro e fornece uma linha. Usado sozinho, fará com que o papel avance em uma linha.

Cada etiqueta emprega nove linhas. Precisamos utilizar nove sinais de interrogação. Vamos imprimir três ou quatro linhas em uma etiqueta, dependendo do conteúdo do campo ATN. Imprimimos o ATN no lugar correto se o campo não estiver em branco. Se estiver, imprimimos um sinal extra de ? após a linha que contém a cidade, o estado e o código CEP.

Deverão ser utilizadas etiquetas de coluna única somente quando o número de etiquetas a ser impresso for relativamente pequeno, porque colocamos maior tensão e desgaste na impressora com etiquetas de uma única coluna. O tempo usado para imprimir um determinado número de etiquetas é também muito maior do que o utilizado para imprimir etiquetas de colunas múltiplas, pois a impressora utiliza maior número de linhas. E o fornecimento de linhas gasta tempo.

Item 6 de Menu Imprimir Etiquetas de Colunas Múltiplas (MENU3-6.CMD/.PRG)

As etiquetas também são encontradas em folhas com várias colunas de etiquetas por folha. Vamos utilizar uma que possui 12 carreiras que de cinco etiquetas por folha.

Cada etiqueta tem 1 polegada por 25/8 polegadas. O espaço utilizável em cada etiqueta é de um pouco menos de 2,5 polegadas. Esse tipo de etiquetas é mais adequado para impressoras que imprimem com tipo de 12 caracteres por polegada (elite).

Ao comprar as etiquetas, deverá ser levado em conta o tamanho dos itens que se quer imprimir. Em nosso exemplo, usaremos uma impressora de dez caracteres por polegada. Isso significa que alguns campos não poderão ter todos os seus dados impressos em uma etiqueta. Entretanto, podemos imprimir os 25 primeiros caracteres de cada campo com uma impressora de 10-cpi. Esse exemplo ilustra a parte de impressão dos conteúdos de um campo. O Programa 6 ilustra o motivo por que deverão ser canceladas variáveis de memória à medida que se abandona um programa. As cinco variáveis de memória neste programa exigem 650 bytes de memória, o que representa mais de um terço do espaço disponível no dBASE II para variáveis de memória.

Nossa abordagem da solução do problema de impressão consiste em estabelecer o tempo de toda uma carreira de etiquetas na memória. Uma vez feito o "estabelecimento do tipo" imprimiremos essa carreira.

```
* Programa .........:MENU3-6.CMD (ou MENU3-6.PRG)
* Obs ..............:Este programa imprime Etiquetas de colunas multiplas.

USE Clientes
SET TALK OFF
SET INDEX TO Cep
GO TOP
SET PRINT ON
SET MARGIN TO 0
STORE "                                                    " TO branco
DO WHILE .NOT. EOF
   STORE 0 TO registros
   STORE " " TO linha1,linha2,linha3,linha4,linha5
   DO WHILE registros < 5 .AND. .NOT. EOF
      STORE registros + 1 TO registros
      STORE TRIM(CIDADE)+", "+ESTADO+branco TO varlinha
      STORE linha1+$(NOME,1,25)+" " TO linha1
      IF ATN = "     "
         STORE linha2+ENDERECO+" " TO linha2
         STORE linha3+$(varlinha,1,25)+" " TO linha3
         STORE linha4+CEP+$(branco,1,20)+" " TO linha4
         STORE linha5+$(branco,1,25)+" " TO linha5
      ELSE
         STORE linha2+$(ATN,1,25)+" " TO linha2
         STORE linha3+ENDERECO+" " TO linha3
         STORE linha4+$(varlinha,1,25)+" " TO linha4
         STORE linha5+CEP+$(branco,1,20)+" " TO linha5
      ENDIF
      SKIP
   ENDDO
   ?
   ? linha1
   ? linha2
   ? linha3
   ? linha4
   ? linha5
ENDDO
SET PRINT OFF
RELEASE linha1,linha2,linha3,linha4,linha5,branco,registros
RETURN
```

Programa 6. MENU3-6.CMD/.PRG Etiquetas de endereçamento de colunas múltiplas

Uma carreira de etiquetas corresponde a cinco registros no banco de dados. Isso significa que precisamos trabalhar no banco de dados, cinco registros por vez. Usaremos dois loops DO. O loop exterior é utilizado para estabelecer o loop interior e para realizar a impressão efetiva. O loop interior "estabelece os tipos" de até cinco registros por vez. Uma marca de fim-de-arquivo encerrará tanto o loop interior como o exterior.

Antes de começar a "estabelecer os tipos", armazenaremos um espaço em branco em cinco variáveis de memória, linha 1 até linha 5. Isso inicializa as variáveis a cada vez.

Nossa operação de "estabelecimento dos tipos" consiste na montagem dessas cinco linhas de modo que cada variável imprimirá uma linha em cada uma das cinco etiquetas. Após termos passado pelo loop interior cinco vezes, a variável linha 1 conterá os 25 primeiros caracteres de cada um dos cinco nomes de clientes. Há, também, um espaço em branco que separa cada um dos nomes.

Em nosso exemplo, temos registros em que o conteúdo do campo ATN, está em branco. Mas não vamos aceitar uma linha em branco em uma etiqueta – queremos deslocar os campos restantes até aquela etiqueta. Isso significa que à medida que processamos cada registro temos duas possibilidades a considerar: vamos "estabelecer o tipo" do campo ATN. ou não.

A linha 1 pode conter os campos ATN. e ENDEREÇO. Não importa o que acrescentemos a cada variável de memória, o acréscimo terá sempre 25 caracteres mais um espaço.

TRIM(CIDADE) varia em tamanho. Para nos assegurarmos de que a linha que contém esse item seja aumentada adequadamente, criamos a variável VARLINHA.

Então, usamos os 25 primeiros caracteres de VARLINHA quando quisermos "estabelecer o tipo" desse item.

Uma vez terminada nossa digitação, imprimimos uma linha em branco e as cinco variáveis de memória. Então voltamos e repetimos tudo novamente.

Observe que em uma passagem pelo loop interior, montamos cada uma das cinco variáveis por um número fixo de caracteres. Os itens atribuídos a variáveis podem ser diferente em cada registro.

Observe que ao montar o programa, usamos o comando SET MARGIN TO 0. Isto é apenas uma garantia – queremos que a impressão se inicie na margem esquerda da página.

Capítulo Sete

O INVENTÁRIO

As operações que queremos executar com o banco de dados de inventário são apresentadas na Tela 1. Essa tela é o produto do Programa 1, o programa menu para nosso sistema de inventários.

Nesta seleção de itens de nosso menu, discutiremos três deles:

- Inclusão de itens no inventário
- Pedidos de livros
- Atualização do inventário

Os programas para editar e excluir registros de inventário podem ser encontrados no Apêndice E.

Nosso banco de dados inventário é composto de três arquivos separados:

O arquivo inventário básico:	INV
O arquivo das editoras:	EDITORAS
Livros no arquivo de pedidos:	PEDIDOS

A Figura 1 mostra os planos para cada um desses três arquivos. A Figura 2 mostra a relação entre esses arquivos. Neste capítulo, trabalharemos com mais de dois arquivos por vez, o que exige fechar um dos arquivos abertos a cada vez que precisarmos abrir um novo arquivo.

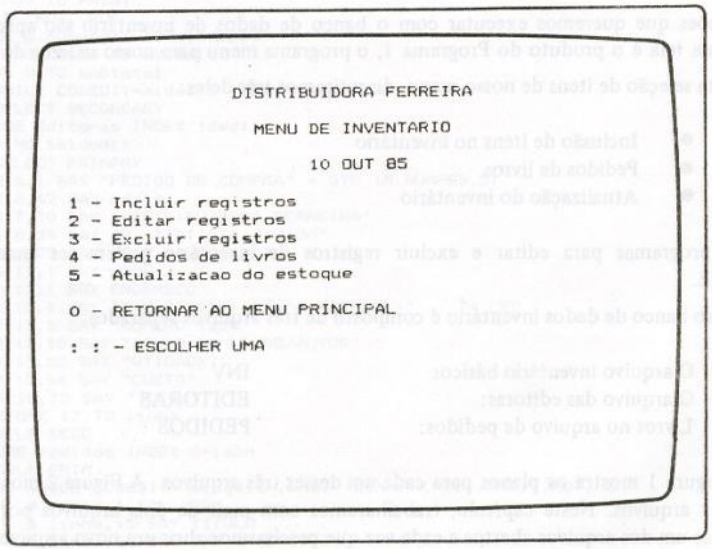
Observe que INV está ligado a PEDIDOS e a EDITORAS. A ligação entre INV-PEDIDOS é o número ISBN; a ligação entre INV ou PEDIDOS e EDITORAS é o código de identificação da editora, que esta incluído no número ISBN.

O Número ISBN

A estrutura do número ISBN é complexa, mas merece ser examinada, já que exemplifica como se deve buscar e verificar outros tipos de números. Ela também mostra como se poderia criar números de peças, números de clientes etc., que podem ser verificados pelos computadores.

A sigla ISBN significa *International Standard Book Numbering*. O número em si representado (e usado) como um número de 13 caracteres. O número possui quatro partes, geralmente separadas por hífens. Há efetivamente dez dígitos nos quatros grupos. São exemplos dos números ISBN:

3-88053-002-5
950-7000-10-X



Tela 1. Tela para o menu de inventário


```
* Programa ........: MENU4.CMD (ou MENU4.PRG)
* Obs. ............: Este e o programa Menu de arquivo de clientes

USE Inv
GO BOTTOM
IF .NOT. # = 0
   STORE ISBN+"  "+TITULO TO Ultentr
ELSE
   STORE "     " TO Ultentr
ENDIF
DO WHILE T
   STORE " " TO escolha
   ERASE
   @ 2,30 SAY "DISTRIBUIDORA FERREIRA"
   @ 4,32 SAY "MENU DE INVENTARIO"
@ 6,34 SAY cdata
@ 8,25 SAY "1 - Incluir registros"
@ 9,25 SAY "2 - Editar registros"
@ 10,25 SAY "3 - Excluir registros"
@ 11,25 SAY "4 - Pedidos de livros"
@ 12,25 SAY "5 - Atualizar o estoque"
@ 14,25 SAY "0 - RETORNAR AO MENU PRINCIPAL"
@ 17,25 SAY "  - ESCOLHA UMA"

 17,24 GET escolha
READ NOUPDATE
IF escolha = "0"
   RELEASE Ultentr,escolher
   RETURN
ENDIF
IF escolha$"12345"
   STORE "MENU4-"+escolha TO escolher
   DO &escolher
ENDIF
ENDDO
RETURN
```

Programa 1. MENU4.CMD/.PRG Programa menu de inventário

```
ARQUIVO DE BANCO DE DADOS:    INV

                                NOME DE CAMPO   TIPO    TAM    DEC
Numero ISBN (numero de peca)    ISBN            C       013
Nome do autor                   AUTOR           C       030
Titulo do livro                 TITULO          C       040
Tema do livro                   TEMA            C       015
Quantidade disponivel           QTIDDISP        N       003
Nivel minimo de estoque         ESTOQMIN        N       002
Quantidade vendida ate este ano VENDIDOS        N       004
Quantidade vendida o ano passado VENDPASS       N       004
Data da ultima venda            ULTVENDA        C       008
Data da ultima remessa recebida ULTREC          C       008
Preco de venda                  PRECOVEN        N       006    002
Custo de compra                 CUSTO           N       006    002
Editora (MFGR) Codigo de identificacao CODEDIT  C       006
Quantidade media para pedidos   QTIDPED         N       003

ARQUIVO DE INDICE : numero ISBN     ISBN. NDX
                    Autor           AUTOR NDX
                    Titulo          TITULO. NDX
                    Codigo da Editora CODEDIT. NDX
                    Tema            TEMA. NDX
```

Figura 1. Planejamento do banco de dados para arquivos de inventário

```
ARQUIVO DE BANCO DE DADOS: EDITORA

                                NOME DE CAMPO    TIPO     TAM     DEC
Nome da editora                 NOME             C        030
A atencao de (nome do contato)  ATN              C        030
Endereco da editora             ENDERECO         C        025
Cidade da editora               CIDADE           C        020
Estado da editora               ESTADO           C        002
CEP                             CEP              C        005
Numero telefonico               TEL              C        014
Codigo de identificacao da editora  IDEDIT       C        007

ARQUIVOS DE INDICE: IDENT. EDITORA   IDEDIT. NDX
                    NOME DA EDITORA  NOME. NDX

ARQUIVO DE BANCO DE DADOS: PEDIDO

                                NOME DE CAMPO    TIPO     TAM     DEC
Numero ISBN                     ISBN             C        013
Numero de nosso pedido de compra NUMPED          N        006
Quantidade solicitada           QTIDPED          N        003
Data do pedido                  DATAPED          C        008
Quantidade recebida ate o momento QTIDREC        N        003

ARQUIVO DE INDICE: NUMERO ISBN   0-ISBN.NDX
```

Figura 1 (Continuação)

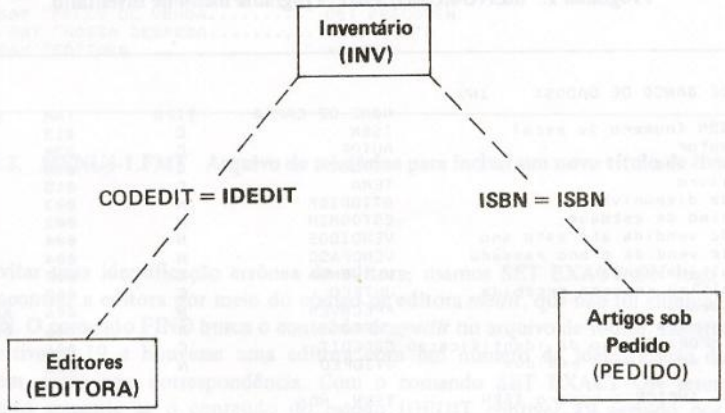


Figura 2. Relação entre os arquivos no banco de dados de inventário

O tamanho de cada um dos três primeiros pode variar, embora haja sempre um total de nove dígitos. O quarto grupo é sempre um único dígito.

O primeiro grupo é o identificador de grupo, que é utilizado para identificar países, regiões geográficas, idiomas etc.

O segundo dígito identifica o editor. Seu tamanho varia de acordo com o número de títulos para um editor. Grandes editoras possuem identificadores pequenos e vice-versa.

O terceiro grupo é o número de identificação do livro para a editora.

O quarto, que é sempre um único dígito, é o dígito de controle.

O número ISBN pode ser verificado por meio de um cálculo numérico que envolve cada um dos seus oito dígitos:

ISBN	3	8	8	0	5	3	0	0	2	5
Multiplicador Digital	10	9	8	7	6	5	4	3	2	1
Produto	30	+ 72	+ 64	+ 0	+ 30	+ 15	+ 0	+ 0	+ 4	+ 5 = 220

Se o produto da operação acima for exatamente divisível por 11, o número será um número ISBN válido. Esse é um exemplo de um número designado para uso com um computador – o número pode ser testado quanto a sua validade.

O Programa 2 é usado para testar a validade de um número ISBN. O primeiro problema a resolver consiste em eliminar os hífens para podermos testar e isolar essa parte do número que identifica a editora. Isso nos dá a oportunidade de demonstrar melhor o uso das funções @ e LEN.

A única posição de dígito de que temos certeza é a última – o dígito de controle. Assim, nosso primeiro passo consiste em isolar o dígito de controle do resto do número até o último hífen.

A seguir, extraímos o identificador de grupo (o primeiro grupo), localizando o primeiro hífen, com o uso da função @. O restante do número é armazenado na variável de memória *rem*, que contém somente o identificador da editora e o do livro (o segundo e o terceiro grupos). Observe que a variável de memória *x* usada nas operações de *substring* é um número:

```
x = @('-',REM)
```

Esse número *x* é a posição do primeiro hífen em *rem*. Não podemos utilizar um operador de *substring* em que a posição de início ou o tamanho seja zero. Se o fizermos, obteremos um erro de sintaxe. É por esse motivo que usamos as duas instruções IF que testam se *x* é maior que zero.

Quando isolamos o identificador de grupo (o primeiro conjunto de números no código ISBN), sabíamos que o tamanho de *rem* era 11, pois quando subdividimos o número ISBN pela primeira vez o fizemos em duas partes, sabíamos que o último dígito, o 13.º, era o dígito de controle e que o 12.º dígito era um hífen. Quando "extraímos" o identificador do grupo, isolamos o restante desse caractere de 11 dígitos em uma variável *rem*. Não conhecemos o tamanho da nova *rem*. Entretanto, podemos obtê-lo usando a função LEN (*rem*) que é um

número. Ela pode ser usada como um número em cálculos e em outras funções do dBASE II. Vamos utilizá-la como um número na operação de *substring* que extrai os dígitos identificadores de livros e armazena-os na variável de memória *tipolivro*.

O loop DO executa a função de multiplicar cada dígito pelo multiplicador apropriado e armazena o resultado acumulado na variável *verifica*. Observe que a primeira linha no programa armazenado armazenava 1 em *verifica*. Isso é uma proteção contra a possibilidade de não haver hífens no número ISBN, pois sem eles não conseguiríamos desmembrar o número para o teste de validade. Se fosse este o caso, o teste de validade mostraria que *verifica* = 1 e o teste falharia — o que esperamos que aconteça.

```
* Programa.....TESTIBSN. CMD ( OU TESTISBN> PRG)
* Obs..........ESTE PROGRAMA TESTA NUMERO ISBN
STORE 1 TO verifica
IF $(misbn,13,1)$"0123456789X"
   STORE $(misbn,13,1) TO digitcntrl
   STORE $(misbn,1,11) TO rem
   STORE @("-",rem) TO X
   IF X > 0
      STORE $(rem,1,X-1)TO grupo
      STORE $(rem,X+1,LEN(rem)-X) TO rem
      STORE @("-",rem) TO X
      IF X > 0
         STORE $(rem,1,X-1) TO medit
         STORE $(rem,X+1,LEN(rem)-X) TO tipolivro
         STORE grupo+medit+tipolivro TO testibsn
         IF digitcntrl= "X"
            STORE 10 TO verifica
         ELSE
            STORE VAL(digitcntrl) TO verifica
         ENDIF
         STORE 1 TO X
         DO WHILE X <= 9
            STORE VAL($(testibsn,X,1))*(11-X)+verifica to verifica
            STORE X+1 TO X
         ENDDO
      ENDIF FOR TESTE3
   ENDIF FOR TESTE2
ENDIF FOR TESTE1
IF verifica-INT(cntrl/11)*11=0
   STORE T TO testvalido
ELSE
   STORE F TO testvalido
ENDIF
RELEASE tipolivro,digitcntrl,rem,grupo,verifica,testisbn
RETURN
```

Programa 2. Programa para testar números ISBN válidos

Item 1 de Menu Incluir Títulos de Livros no Inventário (MENU4-1.CMD/.PRG)

O que desejamos que nosso programa faça? Quando acrescentamos um novo livro em nosso inventário queremos que o computador faça muito mais que simplesmente verificar se o título já

está no banco de dados. Queremos entrar novos livros pelos seus números ISBN. Queremos garantir que cada número ISBN fornecido seja um número válido. Queremos verificar se ele já não está no arquivo de inventário. Queremos garantir que a editora do livro esteja no arquivo de editores e, caso não, queremos incluí-la naquele arquivo.

O programa 3 executa tudo isso. É também nosso primeiro exemplo do uso de dois arquivos de banco de dados ao mesmo tempo. Os dois arquivos que estamos usando são os de inventário INV e o de editora EDITORAS.

O dBASE II permite-nos usar dois arquivos de bancos de dados simultaneamente, sendo um deles identificado como o arquivo primário e o outro como o arquivo secundário. Escolhemos esses arquivos por meio do comando SELECT.

Como montamos o programa? O arquivo já está aberto a partir de MENU4.CMD/.PRG, quando inserimos o programa. Como ainda não há nenhum outro arquivo aberto, INV será o arquivo primário. Se tivéssemos que abrir um outro arquivo nesse ponto, INV estaria fechado e o segundo seria o único arquivo em uso. Para abrir dois arquivos ao mesmo tempo, usamos o comando SELECT SECONDARY. Qualquer arquivo que abrirmos agora será o secundário. O arquivo primário permanece aberto e disponível para uso.

Podemos nos mover dentro do arquivo selecionado, fazer escolhas, incluir e excluir registros, sem afetar o outro arquivo. Qualquer um dos dois arquivos, primário e secundário, ou ambos, pode ser usado com arquivos de índice, como neste programa. O mesmo arquivo não pode ser usado como primário e secundário ao mesmo tempo.

Quando se abre um arquivo, o banco de dados é posicionado no primeiro registro. Quando um dado é escolhido, o banco de dados é posicionado no último registro utilizado. Isso significa que não perderemos nossa posição no banco de dados selecionado, não importa o que façamos com o outro banco de dados. No exemplo, INV é o arquivo primário e EDITORAS é o arquivo secundário.

A Figura 3 é uma representação esquematizada da estrutura lógica do Programa 3. Esse tipo de diagrama é chamado fluxograma. Os fluxogramas podem ser usados para elaborar a estrutura lógica de um programa, antes de se tentar codificá-lo. Embora sejam usados mais freqüentemente com manuais que na programação em si, eles podem nos ajudar a perceber melhor o que desejamos executar.

Observe que a maior parte deste programa consiste em testes. Primeiramente testamos para verificar se efetivamente inserimos ou não um número de livro. Se tivermos inserido, então testamos a validade do número ISBN por meio do comando DO TESTISBN. Lembre-se de que TESTISBN é o Programa 2 e que ele contém duas variáveis de memória: *medit* (o número de identificação da editora) e *test válido* (uma variável lógica que nos informa se o número ISBN era válido ou não). Usamos essas duas variáveis no Programa 3.

Se o número inserido for válido, podemos testar para verificar se esse número, e daí o livro correspondente, está no banco de dados. Se o livro não estiver no banco de dados, usamos o comando SELECT para chamar o arquivo de editoras e verificar se já há um registro para aquela editora. Se não houver, incluiremos todas as informações sobre a editora no arquivo de editoras. Então, incluímos as informações desejadas sobre o livro. Observe que em ambos os casos utilizamos READ NOUPDATE para as informações fornecidas por meio do teclado. O comando REPLACE, que inclui os números apropriados de identificação, vem após o READ.

```
* Programa.....:MENU4-1. CMD (OU MENU 4-1. PRG)
* Obs..........:ESTE PROGRAMA INCLUI REGISTROS NO ARQUIVO DE INVENTARIO

SET INDEX TO Isbn,Autor,Titulo,Codedit,Tema
SELE SECO
USE Editoras INDEX Idedit,Nomedit
SELE PRIM
STORE T TO inclreg
ERASE
STORE "              " TO misbn
@ 23, 1 SAY "ULTIMA ENTRADA"+ Ultentr
DO WHILE inclreg
   @ 21, SAY "ENTRAR O NOVO NUMERO ISBN" GET misbn PICTURE "99999999999-!"
   READ NOUPDATE
   @ 20,0
   IF misbn = "  "
      STORE F TO inclreg
      LOOP
   ENDIF
   DO Testisbn
   IF .NOT. Testvalido
      @ 20,1 SAY "NUMERO ISBN INVALIDO"
      LOOP
   ENDIF
   FIND &misbn
   IF # > 0
      @ 20,1 SAY " O LIVRO NUMERO  " +ISBN+" JA ESTA NO INVENTARIO"
      LOOP
   ENDIF
   SELECT SECO
   SET EXACT ON
   FIND &medit
   IF # = 0
      SET FORMAT TO MENU4-1A
      APPEND BLANK
      READ NOUPDATE
      REPL MEDIT WITH medit
   ENDIF
   SELE PRIM
   SET EXACT OFF
   APPEND BLANK
   SET FORMAT TO MENU4-1
   READ NOUPDATE
   REPLACE ISBN WITH misbn,CODEDIT WITH medit
   SET FORMAT TO SCREEN
   STORE "              " TO misbn
ENDDO
RELEASE inclreg,misbn,oktoadd,trimisbn,medit
RETURN
```

Programa 3. MENU4-1.CMD/.PRG Inclui novos títulos ao inventário

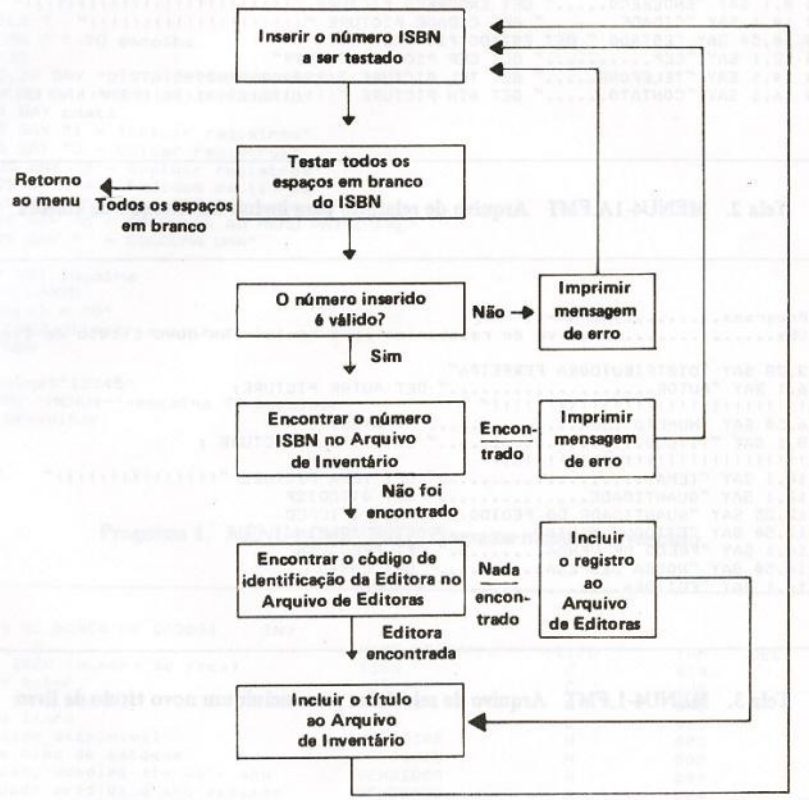


Figura 3. Diagrama lógico do programa para incluir títulos no inventário

```
* Programa.........:MENU4-1A.FMT
* Obs..............:Arquivo de relatorios para incluir informacoes sobre
* editoras

@ 3,28 SAY "DISTRIBUIDORA FERREIRA"
@ 6,1 SAY "EDITORA....." GET NOME PICTURE "!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
@ 8,1 SAY "ENDERECO....." GET ENDERECO PICTURE "!!!!!!!!!!!!!!!!!!!!!!!!!!!"
@ 10,1 SAY "CIDADE....." GET CIDADE PICTURE "!!!!!!!!!!!!!!!!!!!"
@ 10,50 SAY "ESTADO " GET ESTADO PICTURE "!!"
@ 12,1 SAY "CEP....." GET CEP PICTURE "99999"
@ 14,1 SAY "TELEFONE....." GET TEL PICTURE "(999)-999-9999"
@ 16,1 SAY "CONTATO....." GET ATN PICTURE "!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
```

Tela 2. MENU4-1A.FMT Arquivo de relatório para incluir informação da editora

```
* Programa.........:Menu4-1. FMT
* Obs..............:Arquivo de relatorios para incluir um novo titulo de livro

@ 3,28 SAY "DISTRIBUIDORA FERREIRA"
@ 6,1 SAY "AUTOR....." GET AUTOR PICTURE;
"!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
@ 6,50 SAY "NUMERO ISBN....." + misbn
@ 8,1 SAY "TITULO....." GET TITULO PICTURE ;
"!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
@ 10,1 SAY "TEMA....." GET TEMA PICTURE "!!!!!!!!!!!!!!!"
@ 12,1 SAY "QUANTIDADE....." GET QTIDDISP
@ 12,25 SAY "QUANTIDADE DO PEDIDO..:." GET QTIDPED
@ 12,50 SAY "ESTOQUE MINIMO....." GET ESTOQMIN
@ 14,1 SAY "PRECO DE VENDA....." GET PRECOVEN
@ 14,50 SAY "NOSSA DESPESA....." GET CUSTO
@ 16,1 SAY "EDITORA....." + NOME
```

Tela 3. MENU4-1.FMT Arquivo de relatórios para incluir um novo título de livro

Para evitar uma identificação errônea de editora, usamos SET EXACT ON logo antes de tentarmos encontrar a editora por meio do código de editora *medit*, que não foi eliminado depois do teste ISBN. O comando FIND busca o conteúdo de *medit* no arquivo de índice. Habitualmente, se *medit* contivesse 19 e houvesse uma editora com um número de identificação de 19234, levaríamos em conta essa correspondência. Com o comando SET EXACT ON teremos uma correspondência somente se o conteúdo do campo IDEDIT contiver 19 seguido por espaços em branco.

Neste programa, utilizamos dois arquivos de registros: MENU4-1A.FMT, para os dados do arquivo de editoras e MENU4-1.FMT, para as informações sobre livros. Esses arquivos são apresentados pelas Telas 2 e 3.

Item 4 de Menu Pedidos de Livros (MENU4-4.CMD/.PRG)

O que desejamos que nosso programa faça? Neste exemplo queremos que o computador prepare pedidos de compra para os livros que desejamos comprar, ou seja, aqueles cuja quantidade disponível mais a quantidade encomendada está abaixo do nível mínimo de estoque.

Como montamos o programa? A escolha dos livros que devem ser adquiridos é um processo em dois estágios. Primeiramente formamos uma lista dos livros que *talvez* queiramos comprar. São os livros cuja quantidade disponível (QTIDDISP) é menor que o nível mínimo de estoque (ESTOQMIN). Nossa lista de livros possíveis é criada por meio do comando COPY. Agora temos um banco de dados chamado ARQTEMP, em que os únicos registros são de livros que podemos comprar. Os registros estão classificados pela ordem de código de identificação, porque o banco de dados do qual os copiamos estava indexado pelo código de editora.

O passo seguinte será eliminar todos os registros nesse arquivo temporário em que haja número suficiente de livros encomendados para fazer atingir o nível mínimo de estoque no inventário. Para isso, precisamos utilizar o arquivo ARQTEMP juntamente com o arquivo PEDIDOS. Esses dois arquivos são "ligados" por meio do número ISBN. O arquivo ARQTEMP se torna nosso arquivo primário; PEDIDOS, o nosso arquivo secundário.

A seguir, avançamos no arquivo temporário, um registro por vez, e tentamos encontrar cada livro no arquivo PEDIDOS. Podemos executar a operação FIND com os conteúdos do campo ISBN, se primeiramente armazenarmos o conteúdo do campo ISBN em uma variável de memória. Também inicializamos uma variável *qtidade* no valor atual de QTIDDISP.

Para realizar a operação FIND, devemos escolher o arquivo secundário, PEDIDOS. Você poderá se mover no interior de um arquivo quando ele for o escolhido. Pode haver mais de um registro no arquivo PEDIDOS para o livro em questão. O loop interior DO atualizará a variável *qtidade* de acordo com o número de livros encomendados. Observe que se o livro não estiver encomendado, o loop DO interior será ignorado.

Agora retornamos ao arquivo primário. É importante observar que automaticamente retornamos ao registro que estávamos usando antes de sair para examinar o arquivo secundário. Se a variável *qtidade* estiver agora maior que ESTOQMIN, excluímos o registro. Então avançamos um registro e repetimos todo o procedimento. O comando SKIP é uma técnica de endereçamento "relativo". A sintaxe é SKIP n. Se n for um, ele será compreendido (podemos usar o comando SKIP para avançar ou para retroceder).

Essa parte de nosso programa exemplo caracteriza o uso de dois arquivos de banco de dados, que foram relacionados por "referência cruzada com o uso do número ISBN", de forma muito semelhante à que seria usada sem auxílio do computador.

Agora temos de solicitar nossa lista de pedidos. Os que não encomendaremos foram "riscados da lista" (excluídos). Podemos escolher o método de "nos livrarmos" dos registros excluídos, utilizando o comando PACK e eliminando-os fisicamente. Ou podemos usar SET DELETE ON e removê-los logicamente. Quando usamos SET DELETE ON, o dBASE II passa a considerar os registros como se não estivessem lá.

```
Menu 4-4. CMD          Encomendar livros

ERASE
SET INDEX TO Codedit
COPY TO ARQTEMP FOR QTIDDISP<ESTOQMIN
USE ARQTEMP
IF # > 0
   SELE SECO
   USE ENCOMEND INDEX 0-isbn
   SELE PRIM
   DO WHILE. NOT. EOF
      STORE F TO delereg
      STORE QTIDDISP TO QTIDADE
      STORE ISBN TO misbn
      SELE SECONDARY
      FIND &misbn
      DO WHILE misbn=S.ISBN .AND. .NOT. EOF
         STORE S.QTIDPED + qtidade TO qtidade
         SKIP
      ENDDO
      SELECT PRIMARY
      IF qtidade > ESTOQMIN
         DELETE
      ENDIF
      SKIP
   ENDDO
   SET DELETE ON
   GO TOP
   RESTORE FROM Numped ADDITIVE
   ERASE
   @ 5,1 SAY "LIGAR A IMPRESSORA!"
   SET MARGIN TO 15
   SET FORMAT TO PRINT
   DO WHILE. NOT. EOF
      STORE m.numped+1 TO m.numped
      STORE CODEDIT TO midedit
      STORE 0 TO subtotal
      DO WHILE CODEDIT=midedit .AND. .NOT. EOF
         SELECT SECONDARY
         USE Editoras INDEX Idedit
         FIND &midedit
         SELECT PRIMARY
         @ 6,1 SAY "PEDIDO DE COMPRA" + STR (M.NUMPED,5)
         @ 6,62 SAY cdata
         @ 7,30 SAY "DISTRIBUIDORA FERREIRA"
         @ 8,29 SAY "R. ITATIAIA, 10150"
         @ 9,32 SAY "SAO PAULO, SP"
         @ 11,1 SAY NOME
         @ 12,1 SAY ENDERECO
         @ 13,1 SAY TRIM(CIDADE) +","+ ESTADO +"     "+ CEP
         @ 15,1 SAY "NUMERO ISBN"
         @ 15,15 SAY TITULO DO LIVRO/AUTOR"
         @ 15,55 SAY "QTIDADE"
         @ 15,64 SAY "CUSTO"
         @ 15,75 SAY "TOTAL"
         STORE 17 TO linha
         SELE SECO
         USE Pedidos INDEX 0-isbn
         SELE PRIM
         DO WHILE CODEDIT= midedit .AND.  LINHA < 55 .AND. .NOT. EOF
            @ linha,1 SAY ISBN
            @ linha,15 SAY TITULO
            @ linha,55 SAY STR(QTIDPED,3)
            @ linha,60 SAY STR(CUSTO,8,2)
            @ linha,70 SAY STR(CUSTO* QTIDPED,10,2)
            @ linha+1,15 SAY AUTOR
```

Programa 4. MENU4-4.CMD/.PRG Programa para encomendar livros


```
            STORE linha+3 TO linha
            STORE CUSTO * QTIDPED+ subtotal TO subtotal
            SELE SECO
            APPEND BLANK
            REPLACE S.ISBN WITH P.ISBN, S.QTIDPED WITH P.QTIDPED,;
                    NUMPED WITH m.numped, DATAPED WITH DATE()
            SELE PRIM
            SKIP
         ENDDO
         IF CODEDIT#midedit .OR. EOF
            @ linha+ 1,60 SAY "TOTAL"
            @ linha+ 1,70 SAY STR(SUBTOTAL,10,2)
         ENDIF
         EJECT
      ENDDO
   ENDDO
   SET FORMAT TO SCREEN
   SELECT SECONDARY
   USE
   SELECT PRIMARY
   SAVE ALL LIKE numped TO NUMPED
ENDIF
USE INV
SET MARGIN TO 0
DELETE FILE ARQTEMP.dbf
SET DELETE OFF
RETURN
```

Programa 4 (Continuação)

Na segunda parte de nosso programa queremos realmente encomendar os livros preparando um pedido de compra para cada editora da qual estamos comprando – ver a Figura 4 – (em todo este livro estaremos "criando" nossos próprios formulários). Muitas vezes não é conveniente usar formulários impressos, a menos que o computador e sua impressora sejam reservados para uma tarefa específica. Se não o forem, cada vez que quisermos executar uma nova tarefa teremos de parar e trocar de formulário. Uma impressora de qualidade relativamente boa será adequada para produzir formulários "autogerados".

Também queremos preparar um registro para cada compra de livro. Esta parte do programa usará três arquivos de banco de dados para cada pedido de compra preparado: ARQTEMP, EDITORAS e PEDIDOS. ARQTEMP é nossa fonte de livros a serem requisitados. O arquivo EDITORAS fornece-nos o nome e o endereço das editoras das quais estamos comprando. PEDIDOS é nosso registro de pedidos para cada livro.

Neste programa, ARQTEMP será sempre o arquivo primário. Os dois outros arquivos, EDITORAS e PEDIDOS, se revezarão como arquivos temporários. Podemos mudar o arquivo considerado como secundário a cada vez que usarmos um novo arquivo.

Para preparar a segunda parte de nosso programa, precisamos reposicionar o banco de dados no início. Esse procedimento é denominado retrocesso e o comando usado é GO TOP. Isso nos leva ao primeiro registro lógico no banco de dados. Se não houver índices sendo usados, passamos para o primeiro registro físico. O passo seguinte é um simples lembrete do dBASE II ao usuário para ligar a impressora.

Cada pedido precisa de um número de pedido de compra. Para nosso exemplo, recuperamos o último pedido de compra usado com um arquivo .MEM. Mantemos o número de pedido de

compra armazenado na memória (*numped*) durante este programa, aumentando seu valor em uma unidade para cada pedido de compra emitido. Quando encerramos, gravamos o último número usado novamente no arquivo em disco do qual ele veio.

Para executar a solicitação, usamos dois loops DO alojados em um terceiro. O loop exterior deverá repetir o processo até que todos os livros em ARQTEMP estejam classificados. O segundo loop deverá enviar todos os livros para uma editora específica. O loop final deve se encarregar da possibilidade de estarmos encomendando um número de livros maior do que o que pode conter uma única página.

A cada passagem pelo loop exterior, preparamos um pedido de compra. Precisamos aumentar o número do pedido de compra, armazenar o código da editora na memória para que funcione como um controle para os dois loops alojados e inicializar uma variável para armazenar os subtotais para o pedido de compra. Observe o "m." na frente da variável *m.numped*. Temos um campo denominado NUMPED no arquivo PEDIDOS. Como as operações com campos têm prioridade com relação a variáveis de memória, quando temos um campo e uma variável de memória com o mesmo nome, o resultado da operação com a variável não será o que queríamos.

Não usamos o *m.* com o nome da variável quando usamos o comando STORE. O nome *m.var* é um nome legítimo de variável.

STORE NUMPED TO *numped* na variável de memória *numped.*

PEDIDO DE COMPRA 1006 23 JULHO 1985

DISTRIBUIDORA FERREIRA
R ITATIAIIA , 10150
SAO PAULO-SP

LIVRARIA ODISSEIA
AV CAETES, 43
JAU, SP 97742
ATN : AGUIAR

NUMERO ISBN	TITULO DO LIVRO/AUTOR	QTIDADE	CUSTO	TOTAL
0-912677-997	CURSO BASICO DE COMPUTACAO BATAMI, ROBERTO	10	19.95	199.50
0-912677-01-5	MANUAL DE REFERENCIA PARA O PC, DA IBM NUNES, ALVARO/ NADIA	10	69.95	699.50
0-931988-84-5	INTERESSES DE MERCADO ALMEIDA, LIDIA	3	19.95	59.85
0-89588-042-3	ENSAIOS CRITICOS ARAUJO, MAURO	1	1.95	1.95
			TOTAL	960.80

Figura 4. Pedido de compra

O comando STORE NUMPED to *m.numped* armazenará o conteúdo do campo NUMPED na variável de memória *m.numped.*

STORE *m.numped* TO *novonum* armazenará o conteúdo da variável de memória *numped* na variável *novonum.*

Com o ARQTEMP como arquivo primário, nosso primeiro passo será usar o arquivo de editoras, EDITORAS, de modo que podemos imprimir o cabeçalho do pedido de compra. Observe que o cabeçalho utiliza *cdata* "em Inglês" do programa do menu principal. Então, enquanto ainda em SECONDARY, utilizamos o arquivo PEDIDOS. Agora selecionamos o arquivo primário antes de iniciar o terceiro loop.

O objetivo do terceiro loop é o de processar os livros individuais que estão sendo pedidos à editora. O contador de linha controla o espaço disponível na página.

Neste loop, imprimimos informações detalhadas sobre cada livro que estamos solicitando. Também mantemos um subtotal corrente do total em dólares para cada pedido de compra. A quantidade de livros encomendados provém do campo QTIDPED, no arquivo primário. Para cada livro solicitado acrescentamos um registro ao arquivo PEDIDOS, que contém o número ISBN, o número de livros solicitados, o número do pedido de compra e a data do sistema.

Neste exemplo, QTIDPED é comum ao arquivo primário (ARQTEMP) e ao arquivo secundário (PEDIDOS). Distinguimos um do outro precedendo o NOME DE CAMPO por um "P." – um passo importante nesta aplicação.

Sem a especificação dos arquivos armazenaríamos QTIDPED novamente em si mesma, levando-nos a nada. O mesmo acontece com NUMPED; queremos substituir o conteúdo do campo NUMPED pelo conteúdo da variável de memória *numped.*

Quando saímos do terceiro loop, testamos para verificar se realmente encerramos todas as tarefas com a editora. Caso sim, imprimimos o conteúdo da variável de subtotal, emitimos a página e seguimos para o loop exterior.

A seguir, com um pouco de ordem, sai o arquivo secundário. Não há problemas em se ter um, mas *não* ter um evita confusão e garante que selecionemos o mesmo arquivo ao mesmo tempo, como primário e como secundário. Para isso, utilize o comando USE sem o nome de arquivo acompanhando, enquanto SECONDARY for escolhido. Isso encerra o arquivo de banco de dados em uso sem abrir um outro.

Antes de retornar ao menu, voltamos ao arquivo primário e utilizamos o arquivo de banco de dados INV. Cada um dos programas que são chamados do MENU4 supõe que INV está sendo usado e que está no arquivo primário.

Item 5 de Menu Atualizar Inventário (MENU4-5.CMD/.PRG)

O que queremos que nosso programa faça? Quando uma remessa de livros é recebida, queremos atualizar a quantidade disponível no arquivo de inventário, fechar (excluir) os registros no arquivo PEDIDOS e imprimir os itens atualizados. O Programa 5 executa essas tarefas.

A operação seguinte consiste em inserir os livros recebidos em um arquivo temporário. O comando COPY pode ser utilizado para criar a estrutura de arquivo que queremos usar com nosso arquivo temporário. Realmente queremos somente parte da estrutura copiada e nenhum registro. Pela especificação dos campos a serem copiados, podemos definir a estrutura que queremos utilizar em nosso arquivo temporário. A sentença FOR utiliza uma condição impossível – quantidade recebida maior que 10.000 – o que assegura que nosso novo arquivo de banco de dados não terá nenhum registro. O comando NEXT 1 acelera o processo, evitando que o dBASE II leia todo o arquivo PEDIDOS.

Como montamos o programa? Queremos iniciar o processo de entrada de dados com INV como o arquivo primário e nosso novo arquivo ATUALIZAÇÃO como o arquivo secundário. À medida que os itens são inseridos em ATUALIZAÇÃO, vamos querer que o maior número deles permaneça na tela. Uma entrada normal de tela seria parecida com a Tela 3. Geralmente, quando os dados são fornecidos com o uso do comando GET, os sinais de dois pontos definem o início e o final da área de entrada de dados. Neste exemplo, não queremos que os dois pontos apareçam, assim usamos o comando SET COLON OFF.

```
* Programa ...........:MENU4-5.CMD (ou MENU4-5.PRG)
* Obs. ..............:Este programa atualiza o Inventario

ERASE
USE Pedidos
COPY NEXT 1 FIELDS ISBN,QTIDREC TO Atualiza FOR QTIDREC >10000
USE Inv INDEX Isbn
SELE SECO
USE Atualiza
STORE T TO matualiza
STORE "              " TO misbn
@ 1,1 SAY "NUMERO ISBN          QTIDADE          TITULO"
STORE 2 TO linha
SET COLON OFF
DO WHILE matualiza
   @ LINHA,1 GET misbn PICTURE "99999999999-!"
   READ NOUPDATE
   IF .NOT. misbn = " "
      SELE PRIM
      FIND &misbn
      IF # = 0
         @ linha,20 SAY misbn + " NAO ESTA NO BANCO DE DADOS"
         LOOP
      ENDIF
      @ linha,20 SAY TITULO
      SELE SECO
      APPEND BLANK
      @ linha,15 GET QTIDREC
      READ NOUPDATE
      CLEAR GETS
      REPLACE ISBN WITH misbn
      SELE PRIM
      REPLACE QTIDDISP WITH QTIDDISP+QTIDREC,ULTREC WITH DATE() NOUPDATE
      IF linha < 23
         STORE linha+1 TO linha
      ELSE
         ?
      ENDIF
      STORE "              " TO misbn
   ELSE
```



```
         STORE F TO matualiza
      ENDIF
   ENDDO
   SET COLON ON
   SELE SECO
   USE
   SELE PRIM
   USE Pedidos INDEX 0-isbn
   UPDATE FROM Atualiza ON ISBN RANDOM ADD QTIDREC
   GO TOP
   STORE 0 TO excesso
   STORE "      " TO misbn
   DO WHILE .NOT. EOF
      IF QTIDPED > QTIDREC .AND. ISBN = misbn .AND. excesso > 0
         REPLACE QTIDREC WITH QTIDREC+excesso
      ENDIF
      IF QTIDREC > QTIDPED
         STORE QTIDREC-QTIDPED TO excesso
         STORE ISBN TO misbn
      ENDIF
      IF QTIDPED<=QTIDREC
         DELETE
      ENDIF
      SKIP
   ENDDO
   SELE PRIM
   USE Inv INDEX Isbn
   SELE SECO
   USE Atualiza
   ERASE
   ? "VERIFIQUE SE A IMPRESSORA ESTA LIGADA"
   SET FORMAT TO PRINT
   DO WHILE .NOT. EOF
      STORE 9 TO linha
      @ 5,30 SAY "ATUALIZACAO DO INVENTARIO PARA " +cdata
      @ 7,15 SAY "NUMERO ISBN"
      @ 7,35 SAY "QTIDADE"
      @ 7,45 SAY SAY "TITULO"
      DO WHILE .NOT. EOF .AND. linha<55
         @ linha,15 SAY ISBN
         @ linha,35 SAY QTIDREC
         STORE ISBN TO misbn
         SELE PRIM
         FIND &misbn
         @ linha,45 SAY TITULO
         STORE linha+1 TO linha
         SELE SECO
         SKIP
      ENDDO
      EJECT
   ENDDO
   SET FORMAT TO SCREEN
   SELE PRIM
   RELEASE matualiza,misbn,linha,excesso
   RETURN
```

Menu4-5.CMD/.PRG Atualiza o inventário

Somente o número ISBN e a quantidade de livros recebida serão fornecidos em duas colunas. À medida que cada item é fornecido, o título do livro é recuperado do banco de dados inventário e apresentado na terceira coluna. Se o livro não for encontrado no inventário, apresentamos a mensagem "o item não está no banco de dados" na terceira coluna. Antes de fornecer o DO LOOP,

apresentamos os cabeçalhos das colunas. Estes permanecem na tela até serem fornecidos 22 itens. Nesse ponto, a tela sobe uma linha a cada inserção – e cada nova inserção é fornecida na Linha 23.

O número ISBN é fornecido em uma variável de memória *misbn* em nosso arquivo de inventário. Se não for encontrado, voltamos e tentamos mais uma vez. Se estiver, apresentamos o título e selecionamos o arquivo secundário, acrescentamos um registro em branco, inserimos a quantidade recebida no campo QTIDREC e passamos o conteúdo da variável de memória *misbn* para o campo ISBN. Temos agora um registro completo do item que está sendo atualizado.

Agora voltamos ao nosso banco de dados INV. Ele estava posicionado no registro correto junto com o último FIND. Atualizamos os campos para a quantidade disponível e para a última data recebida. Observe que o conteúdo de um campo é usado no arquivo secundário para auxiliar na atualização. Lembre-se de que, o comando NOUPDATE impede a atualização de arquivos de índice. Nenhum dos itens indexados foi modificado.

O comando CLEAR GETS impede que voltemos o cursor para "corrigir" uma entrada anterior. Devido ao modo como o programa foi estruturado, não podemos voltar para realizar correções. Além disso, podemos ter somente 64 "gets" entre um comando ERASE ou um comando CLEAR GETS.

Depois de terminada a inserção dos livros no arquivo temporário, nosso arquivo de inventário também estará atualizado. Nosso passo seguinte será apagar os registros apropriados do arquivo PEDIDOS.

O comando UPDATE é usado para atualizar um arquivo com os dados armazenados em um outro. O arquivo que está sendo atualizado é PEDIDOS; o arquivo a partir do qual a atualização é feita é o arquivo temporário ATUALIZA. Esses dois arquivos são vinculados por meio do número ISBN. Neste caso, os números ISBN em ATUALIZA podem não estar ordenados. Para usar o comando UPDATE aqui, o arquivo que está sendo atualizado deve ser indexado na vinculação (como está) e a palavra RANDOM deve ser usada na instrução de comando. ADD QTIDREC acrescenta o valor no arquivo ATUALIZA ao valor do mesmo campo no arquivo PEDIDOS.

Parte da programação consiste em prever o que pode acontecer. Poderia haver no arquivo PEDIDOS mais de um registro para um registro no arquivo ATUALIZA. Esta última operação teria atualizado somente o primeiro dos registros no arquivo PEDIDOS. Também é possível que a quantidade atualizada seja maior do que a pedida pelo registro.

Agora queremos passar pelo arquivo PEDIDOS um registro por vez. Para cada registro, há três possibilidades. A quantidade recebida pode ser maior do que, menor do que ou igual à quantidade do pedido. Se a quantidade for menor do que a quantidade do pedido, ignoramos o registro, se for igual, apagamos o registro e se for maior do que a quantidade do pedido, armazenamos o excesso em uma variável de memória e apagamos o registro. Então verificamos o registro seguinte. Se o número ISBN for o mesmo e a quantidade for menor do que a quantidade do pedido e o total armazenado em EXCESSO for maior do que zero, substituímos a quantidade pelo excesso e testamos a quantidade pela quantidade do pedido.

Agora imprimimos o conteúdo de nosso arquivo temporário. Para fazê-lo, usamos o arquivo de inventário juntamente com o arquivo temporário de modo que possamos imprimir o título do livro.

Capítulo Oito

INSERÇÃO DOS PEDIDOS DE VENDAS

A inserção de pedidos de vendas é o processo de fornecer ao computador novos pedidos para mercadorias, de tal forma que esses pedidos possam ser preenchidos. Uma vez preenchidos eles se tornam faturas e são usados por Contas a Receber para preparar os faturamentos.

O desenvolvimento de um sistema de inserção de pedidos pode ser um desafio. Muitas vezes, um pedido de compra se traduz em mais de um registro de banco de dados, e o recurso de entrada de dados pelo dBASE II permite a entrada de registro *individual*. O desafio está em tornar o processo natural para o usuário.

Em nosso exemplo de contabilidade, as informações a serem fornecidas em cada registro consistem em um número de identificação do pedido, e:

quem solicitou os livros
que títulos e quantidades de livros
quando o pedido foi inserido

Queremos manter os registros de pedidos "diretamente acessíveis" durante o período de um ano. Optamos por usar três arquivos de banco de dados para nosso sistema de inserção de pedidos. A Figura 1 representa esses três arquivos.

Os três arquivos de banco de dados VENDAS, DETVEND e PEDABR são vinculados por meio do número do pedido NUMPVEN (ver Figura 1). O arquivo VENDAS é uma listagem mestre de todos os pedidos que foram fornecidos ao sistema, estejam ou não preenchidos. O arquivo DETVEND contém detalhes de venda sobre cada pedido. O arquivo PEDABR é uma listagem de pedidos que ainda não foram preenchidos.

Para economizar espaço no disco, optamos por usar dois arquivos, VENDAS e DETVEND, em vez de um único arquivo. Se nossa empresa processar 20 pedidos por dia e a média de pedidos for de dez diferentes títulos e houver 200 dias úteis no ano, VENDAS terá 4.000 registros e DETVEND terá 40.000.

Combinados em um único arquivo, cada um dos 40.000 registros ocuparia 51 bytes de espaço em disco (50 bytes de dados mais um byte improdutivo). Isso exige mais de 1.000.000 de bytes de armazenamento em disco. Nossa abordagem em dois arquivos ocupa menos de 1.300.000 bytes – economizando cerca de 700.000 bytes.

Os pedidos de vendas são fornecidos à medida que são recebidos e os seus números são atribuídos seqüencialmente.

BANCO DE DADOS DE VENDAS

Número do Pedido de Venda	Código de Identificação do Cliente	Data	Número do Pedido de Compra do Cliente
100211	016215	09/21/83	A6125A3
100212	321311	09/21/83	B-2314
100213	116161	09/22/83	D-6524
100214	553131	09/22/83	X-63-83
100215			
100216			
100217			
100218			

BANCO DE DADOS DOS DETALHES DE VENDA

Número do Pedido de Venda	Número ISBN	QTID	Preço
100211	X – – – – –	23	8.75
100211	– – – – – –	14	21.63
100211	– – – – – –	39	– – – – – –
100212	– – – – – –	64	– – – – – –
100212	– – – – – –	101	– – – – – –
100213	– – – – – –	98	– – – – – –

PED EM ABERTO

N.º do Pedido de Vendas
100214
100215

Figura 1. Representação das vendas no banco de dados de pedidos


```
ARQUIVO DE BANCO DE DADOS: VENDAS
                                  NOME DE ARQUIVO    TIPO     TAM     DEZ
NUMERO DO PEDIDO DE VENDA         NUMPVEN            C        006
CODIGO DE IDENT. DO CLIENTE       NUMCLN             C        006
NUMERO DO PEDIDO DE COMPRA        NUMPED             C        008
DATA DE ENTRADA DO PEDIDO         DATA               C        008

ARQUIVO DE BANCO DE DADOS : DETVEND

NUMERO DO PEDIDO DE VENDA         NUMPVEN            C        006
NUMERO ISBN DO LIVRO              ISBN               C        013
QUANTIDADE SOLICITADA             QTIDADE            N        003
PRECO PARA O CLIENTE              PRECO              N        006     002

ARQUIVO DE BANCO DE DADOS: PEDABR

NUMERO DO PEDIDO DE VENDA         NUMPVEN            C        006
```

Figura 2. Esquema para a entrada dos pedidos de venda

```
* Programa.........: MENU5.CMD ( OU MENU5.PRG)
* Obs..............:ESTE PROGRAMA E O MENU DE PEDIDOS DE VENDA.

DO WHILE T
   STORE "  " TO escolha
   ERASE
   @ 2,30 SAY "DISTRIBUIDORA FERREIRA"
   @ 4,31 SAY "MENU DE PEDIDOS DE VENDA"
   @ 6,34 SAY cdata
   @ 8,25 SAY "1- Entrar um Pedido de Venda"
   @ 9,25 SAY "2- Editar um Pedido de Venda"
   @ 10,25 SAY "3- Cancelar um Pedido de Venda"
   @ 11,25 SAY "4- Imprimir Pedidos de Venda"
   @ 12,25 SAY "5- Preparar um Relatorio de Venda"
   @ 13,25 SAY "6- Pedidos Pendentes"
   @ 14,25 SAY "7- Excluir Pedidos com mais de 1 ano"
   @ 15,25 SAY "0- VOLTAR AO MENU PRINCIPAL"
   @ 18,25 SAY " - ESCOLHA UMA"
   @ 18,24 GET escolha
   READ NOUPDATE
   IF escolha = "0"
      RELEASE ultentr,escolher
      RETURN
   ENDIF
   IF escolha $ "1234567"
      STORE "MENU5-"+ escolha TO escolher
      DO ESCOLHER
   ENDIF
ENDDO
```

Programa 1. MENU5.CMD/.PRG Programa de menu para entrada de pedidos de venda

Em nosso exemplo, vamos tirar proveito disso e desenvolver nosso sistema de entrada de pedidos sem usar *nenhum* arquivo de índice. Também aproveitaremos o fato de que a maioria dos dados inseridos recentemente corresponde aos dados de maior interesse, o que é válido para muitas empresas e arquivos pessoais.

Até o final deste capítulo estaremos examinando cinco programas relacionados no Programa 1 (MENU 5):

MENU5-1 – Entrar um pedido de venda
MENU5-4 – Imprimir um pedido de venda
MENU5-5 – Preparar um relatório de venda
MENU5-6 – Apresentar os pedidos pendentes
MENU5-7 – Eliminar pedidos superados

Os MENUS 5-2 e 5-3 podem ser encontrados no Apêndice E.

Item 1 de Menu Entrar (Inserir) um Pedido de Venda (MENU5-1.CMD/.PRG)

O Programa 2, que possibilita a inserção de pedidos de vendas, é apresentado em diagrama pela Figura 3. O programa solicita ao usuário que insira o código de identificação do cliente, verifica se esse código corresponde à ultima entrada feita, localiza o registro do cliente e apresenta informações quanto ao cliente. O usuário pode então inserir o número ISBN e a quantidade de cada livro solicitado. Quando todos os livros solicitados tiverem sido fornecidos, o número de pedido de venda será acrescentado ao arquivo de pedidos pendentes.

Nosso programa está inteiramente incluído no loop *pedvenda*. Ao inserirmos o loop, selecionamos o arquivo primário. Isso deve ser feito porque um pouco adiante no programa, quando o arquivo secundário for escolhido, poderemos encontrar uma situação que nos levará de volta ao início do loop.

Quando programamos, é importante estarmos atentos quanto aos resultados de comandos. O dBASE II não permite que um mesmo arquivo seja escolhido para ser tanto o primário como o secundário. Abrir o programa com USE VENDAS faria o dBASE II apresentar uma mensagem de erro.

Este programa exibe na tela um marcador de posição a cada vez que o usuário recebe uma mensagem para inserir um código de identificação de cliente. O marcador de posição, que muitas vezes é negligenciado pelos programadores, exibe o nome do cliente para o último pedido fornecido. Sem um marcador de posição, se o operador das entradas de dados desviar a atenção dos pedidos, ele não terá indicação quanto ao último pedido fornecido.

Nosso primeiro passo no programa consiste em usar o arquivo de vendas para obter as variáveis *proxentr* e *mnumcl* (utilizaremos *proxentr* mais adiante no programa, quando o registro de vendas for acrescentado aos bancos de dados VENDAS, DTEVEND e PEDABR). Executaremos o teste para registros de pedidos de vendas a cada vez que inserirmos um pedido de venda. A variável MNUMCLN é usada para localizar o registro do cliente – para o último pedido de venda.

Este programa exige que o usuário forneça o código de identificação do cliente para iniciar um pedido de venda. Poderíamos ter usado o nome do cliente, como em um dos exemplos anteriores, ou o nome ou o código de identificação, mas, nesse caso, seria necessária uma programação adicional que não interferisse no item em questão, inserção de pedido.

Nosso próximo passo será testar a variável *mnumcln*. Se for digitado < RETURN > em vez do código de um cliente, o programa voltará ao menu. O texto seguinte prevê a possibilidade de o código de cliente fornecido ser o mesmo que o código para o último pedido de venda. Aqui existe uma opção: recomeçar o processo ou inserir um novo pedido. A seguir, tentamos localizar o código do cliente no arquivo de cliente. Se não formos bem-sucedidos nessa tentativa, receberemos uma mensagem de erro na tela e começaremos de novo.

Se assinalarmos um erro e precisarmos recomeçar, as mensagens de erro permanecerão na tela durante o tempo necessário para abrir os arquivos e localizar o último registro inserido. Isso não é muito demorado, mas é bom que o usuário tenha conhecimento desse processo, o qual também elimina a necessidade de ficarmos atentos às mensagens de erro e partes vazias da tela à medida que nos deslocamos através do programa.

Optamos por utilizar o comando INPUT para apresentar as perguntas especiais ao usuário. Isso minimiza o risco de o usuário "digitar sobre" a pergunta. INPUT exige um < RETURN > antes de admitir a entrada.

Após localizar o registro do cliente, verificamos a exatidão do código, fornecendo uma possibilidade de saída. Observe como os dados do cliente são apresentados. O sinal de $ usa a linha atual de tela (Linha 1, imediatamente após o ERASE), um método excelente para lidar a possibilidade de querermos que um item seja incluído na tela. A desvantagem do $ como um comando de posição é que não sabemos onde estamos na tela. Para garantir que apagamos a pergunta INPUT, nós apagamos ambas as Linhas 5 e 6.

Uma vez que temos o cliente desejado, acrescentamos um registro em branco, por meio do comando APPEND, ao arquivo VENDAS e digitamos o número do pedido de venda (obtido da variável *proxentr*, criada no início do programa), a data do sistema e o código do cliente. Poderíamos ter optado por fazer o nome do cliente uma parte do arquivo, o que *daria* uma melhora mínima na velocidade mas duplicaria a capacidade do arquivo.

Agora permitimos ao usuário digitar o número do pedido de compra do cliente (NUMPED). Até aqui, criamos o registro mestre de venda. O passo seguinte será inserir os livros solicitados no arquivo DETVEND.

Quando inserimos os livros que estão sendo solicitados no arquivo, utilizamos o número ISBN. Nosso programa examina o título do arquivo de inventário e apresenta o título com o preço a ser cobrado ao cliente. Então inserimos a quantidade do item que está sendo solicitado. Uma tela que representa bem esta inserção de pedido é a Tela 1.

Novamente, precisamos realizar uma série de testes: determinar se encerramos o processo de entrada de dados, verificar se o número ISBN já está no inventário, e testar a validade do número ISBN.

Se teclarmos < RETURN > em vez do número ISBN, o processo estará encerrado. Se digitarmos um número, tentamos localizar esse número no banco de dados de inventário. Se não o localizarmos, um ou dois fatos terão acontecido: ou não colocamos o livro em estoque

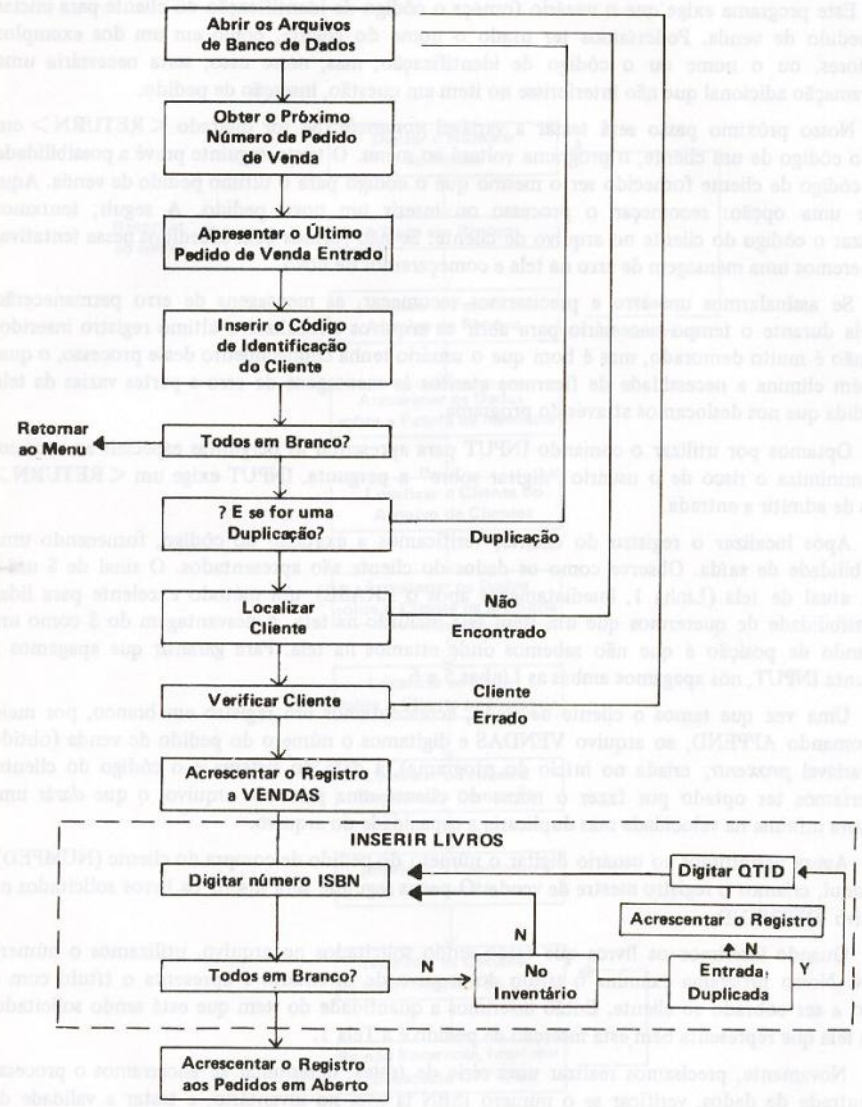


Figura 3. Diagrama do Programa 2


```
* Programa.........:MENU5-1.CMD ( OU MENU5-1.PRG)
* Obs..............:ESTE E O PROGRAMA PARA ENTRADA DE PEDIDOS DE VENDA

STORE T TO PEDVEN
DO WHILE pedven
   SELE PRIM
   USE Vendas
   GO BOTTOM
   IF # ) 0
      STORE VAL(NUMPVEN) + 1 TO proxentr
   ELSE
      STORE 100000 TO proxentr
   ENDIF
   STORE NUMCLN TO mnumcln
   SELE SECO
   USE Clientes INDEX Numcln
   FIND &mnumcln
   IF proxentr= 100000
      STORE "                " TO ultentr
   ELSE
      STORE "ULTIMA ENTRADA: "+NUMPVEN +"   " +NOME+ "   " + P.DATA TO;
      ultentr
   ENDIF
   ERASE
   @ 1,20 SAY "SISTEMA DE ENTRADA DE PEDIDOS DE VENDA"
   @ 23,1 SAY ultentr
   STORE "      " TO mnumcln
   @ 3,0 SAY "ENTRAR O NUMERO DE IDENTIFICACAO DO CLIENTE " ;
         GET mnumcln PICTURE "999999"
   READ NOUPDATE
   IF mnumcln = "      "
      STORE F TO pedven
      LOOP
   ENDIF
   IF P.NUMCLN = mnumcln
      INPUT " O NO. CLIENTE CORRESPONDENTE A ULTIMA ENTRADA -;
ENTRAR ASSIM MESMO (Y/N)?" TO pergunta
      IF .NOT. pergunta
         LOOP
      ENDIF
   ENDIF
   FIND &mnumcln
   IF # = 0
      @ 4,1 SAY "NUMERO DE CLIENTES INVALIDO - TENTE NOVAMENTE"
      LOOP
   ENDIF
   ERASE
   @ 6,1 SAY NOME
   IF .NOT. ATN = "  "
      @ 5+i,1 SAY ATN
   ENDIF
   @ 5+1,1 SAY ENDERECO
   @ 5+1,1 SAY TRIM(CIDADE) + "  ,  " + ESTADO + "   " + CEP
   ?
INPUT " E O CLIENTE CERTO (Y/N)? " TO pergunta
IF .NOT. pergunta
   LOOP
ENDIF
@ 5,0
@ 6,0
SELE PRIM
APPEND BLANK
REPL NUMPVEN WITH STR(proxentr, 6), DATA WITH DATE( ), NUMCLN WITH;
     mnumcln
STORE NUMPVEN TO mpven
```

Programa 2. MENU5-1.CMD/.PRG. Programa de inserção (entrada) dos pedidos de venda

```
@ 6,1 SAY "NUMERO DO P.VEN DO CLIENTE " GET NUMPED
READ
@ 6,1 SAY "NUMERO ISBN          QTIDADE       TITULO"
USE Detven
SELE SECO
USE Inv INDEX Isbn
SELE PRIM
STORE T TO detalhe
STORE 22 TO linha
DO WHILE detalhe
   STORE "                " TO misbn
   @ linha,1 GET misbn
   READ
   CLEAR GETS
   IF misbn = "  "
   STORE F TO detalhe
   LOOP
ENDDO
SELE SECO
FIND &misbn
IF # = 0
   DO TESTISBN
   IF .NOT. testvalido
      @ linha,30 SAY misbn+ "NAO E VALIDO"
   ELSE
      @ linha,30 SAY misbn+ " NAO ESTA NO INVENTARIO"
      STORE linha+1 TO linha
   ENDIF
   LOOP
ENDIF
@ linha,30 SAY TITULO
@ linha,65 SAY LISTPRECO
SELE PRIM
GO BOTTOM
DO WHILE NUMPVEN = mpven .AND. .NOT. ISBN = misbn .AND. .NOT. #=0
   SKIP - 1
ENDDO
IF .NOT. (NUMPVEN = mpven .AND. misbn =ISBN)
   APPEND BLANK
   REPLACE NUMPVEN WITH mpven, ISBN WITH misbn, PRECO WITH listpreco
ELSE
   @ linha,0
   @ linha,1 SAY misbn
   @ linha,30 SAY "-----ENTRADA DUPLICADA-----"
   ENDIF
   @ linha,20 GET QTIDADE
   READ
   CLEAR GETS
   IF linha > = 23
      ?
   ELSE
      STORE linha+1 TO linha
   ENDIF
   SELE PRIM
   USE Pedabr
   APPEND BLANK
   REPLACE NUMPVEN WITH mpven
ENDDO
RELEASE pedven,proxentr,maxmcln,pergunta,mpven,detalhe,linha;
      misbn,medit,testvalido
SELE SECO
USE
SELE PRIM
RETURN
```

Programa 2 (Continuação)

ou o número foi fornecido incorretamente. No espaço normalmente reservado para a impressão do título, imprimimos uma mensagem de erro que inclui o número fornecido e segue para o início desta parte do programa. Imprimimos o número como parte da mensagem de erro porque todos os espaços em branco são novamente armazenados na variável de entrada *misbn* a cada passagem pelo programa. Essa mensagem permite ao usuário determinar o problema.

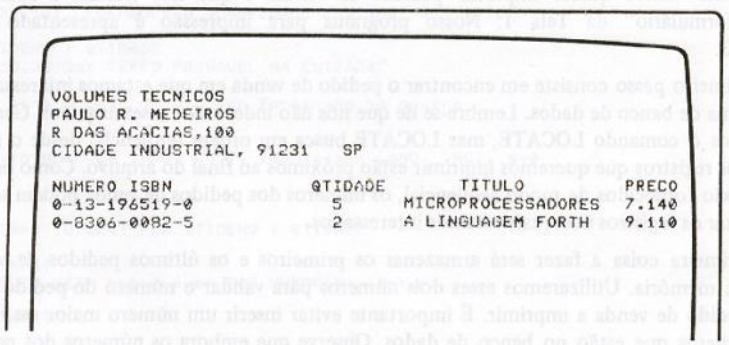
Se o número for encontrado, apresentamos o título e o preço. Antes de executarmos qualquer outra atividade, verificamos se já não inserimos o arquivo para esse pedido de venda.

Se houver uma inserção publicada, ela deverá estar em algum ponto próximo a nosso banco de dados não-indexados. Para procurá-la, fazemos a busca retrocedendo a partir do final do banco de dados.

SKIP n é um conjunto de endereçamento relativo que desloca o usuário em um total n (positivo ou negativo) a partir do registro atual. Com SKIP, você não precisa saber onde está. Ele não poderá levá-lo para além do final (ou do início) do banco de dados. O endereçamento direto com GOTO 20.000 produzirá uma mensagem de erro do dBASE II, se houver menos de 20.000 registros no banco de dados. SKIP 20.000 leva até o final do arquivo e não passa dele, sem produzir uma mensagem de erro.

Nosso loop nos fará retroceder enquanto o conteúdo de NUMPVEN corresponder ao conteúdo de *mpven* e nós não encontrarmos o número ISBN. Observe que o termo .AND. .NOT. # = 0 equivale à função EOF (fim-de-arquivo) do início do arquivo. Precisaremos dele aqui – embora nós o estaremos usando somente no primeiro pedido de venda inserido.

Se não localizarmos uma duplicação, acrescentamos um registro em branco ao banco de dados e substituímos o campo do pedido de venda pelo conteúdo de *proxentr*, o campo ISBN pela variável *misbn*, e o campo PREÇO pelo conteúdo do campo LISTPREÇO no banco de dados INV. Embora o preço já esteja no banco de dados INV, nós o acrescentamos para garantir que seja cobrada ao cliente a quantia correta, caso o preço seja alterado (em INV) antes do próximo faturamento.



Tela 1. Apresentação no vídeo de uma inserção de pedido de venda

Se encontrarmos uma entrada duplicada, apagamos a linha de entrada na tela, escrevemos o número ISBN, por meio do comando SAY, e imprimimos a mensagem ENTRADA DUPLICADA, onde estava o título.

Em ambos os casos, recebemos uma mensagem para fornecer a quantidade. Se o registro estiver sendo editado, o valor anteriormente fornecido será visível. Isso permite ao operador editar um registro anterior inserindo deliberadamente um número duplicado. Anularemos o registro inserindo 0 na quantidade.

A cada vez que o comando READ foi usado para a entrada de dados, colocamos em seguida o comando CLEAR GETS. Há dois motivos para isso: não queríamos que o usuário pudesse fazer correções (na tela) acreditando ter alterado a entrada; também estamos limitados às 64 utilizações do comando GET antes de *termos* que emitir ou um CLEAR GETS ou um ERASE.

Agora nós aumentamos a linha contadora e inserimos o livro seguinte. A abordagem que está sendo usada permite até 24 títulos por vez, na tela. Como podemos ter de inserir mais de 24 títulos para um único pedido, interrompemos o acréscimo de linhas quando chegamos à Linha 23. Neste ponto, usamos um único sinal de ? a cada vez que passamos pelo loop. O sinal de ? na Linha 23 fará com que o conteúdo da tela suba em uma linha. A partir desse ponto, cada nova entrada estará na Linha 23, pois o dBASE II conta as Linhas de 0 a 23.

Após inserir o pedido, escolhemos nosso arquivo de controle PEDABR (a lista de pedidos de vendas que não foram preenchidos) como o arquivo primário.

Quando todos os pedidos tiverem sido fornecidos, limpamos o programa, encerrando o arquivo secundário e abandonando todas as variáveis de memória.

Item 4 de Menu Imprimir Pedidos de Vendas (MENU5-4.CMD/.PRG)

Provavelmente vamos querer imprimir pedidos de vendas e que eles tenham a aparência da entrada "formulário" da Tela 1. Nosso programa para impressão é apresentado como o Programa 3.

O primeiro passo consiste em encontrar o pedido de venda em que estamos interessados para nosso sistema de banco de dados. Lembre-se de que nós não indexamos esses arquivos. Geralmente, utilizaríamos o comando LOCATE, mas LOCATE busca em ordem seqüencial desde o início do arquivo e os registros que queremos imprimir estão próximos ao final do arquivo. Como os pedidos de vendas são fornecidos de modo seqüencial, os números dos pedidos de venda podem ser usados para localizar os registros nos quais estamos interessados.

A primeira coisa a fazer será armazenar os primeiros e os últimos pedidos de venda em variáveis de memória. Utilizaremos esses dois números para validar o número do pedido de venda como o pedido de venda a imprimir. É importante evitar inserir um número maior ou menor do que os números que estão no banco de dados. Observe que embora os números dos pedidos de venda sejam armazenados como caracteres, podemos compará-los como se fossem campos numéricos.

Uma vez digitado um número que esteja no limite válido, calculamos o número de registro para aquele registro.

STORE VAL(mpven)-VAL(primeiro) + 1 TO *posição*

Utilizamos a função VAL para permitir o uso dessas variáveis de *strings* de caracteres da aritmética. Foi necessário somar um à diferença, pois nosso pedido de venda *mpven* podia ser o primeiro no banco de dados (*primeiro*).

Se nosso banco de dados for perfeito e não houver falta de registros, o comando GO LUGAR nos posicionará diretamente no registro desejado. Temos a opção de ou usar GO ou então GOTO *lugar*. Quando usamos o endereçamento direto, devemos estar seguindo para um número válido de registro.

```
* Programa.....:MENU5-4.CMD (OU MENU5-4.PRG)
* Obs..........:ESTE PROGRAMA IMPRIME PEDIDOS DE VENDA

USE Vendas
STORE NUMPVEN TO inicia
GO BOTTOM
STORE NUMPVEN TO termina
ERASE
@ 1,1 SAY "IMPRIMIR PEDIDOS DE VENDA"
STORE T TO imprimir
DO WHILE imprimir
   STORE "      " TO mpven
   @ 3,1 SAY "ENTRAR O NUMERO DO PEDIDO DE VENDA A SER IMPRESSO";
         GET mpven PICTURE "999999"
   READ
   IF mpven = "      "
      STORE F TO imprimir
    * LOOP
   ENDIF
   IF mpven ) termina
      @ 5,1 SAY "NUMERO MUITO ALTO"
      LOOP
   ENDIF
   IF mpven ( inicia
      @ 5,1 SAY "NUMERO MUITO BAIXO"
      LOOP
   ENDIF
   STORE VAL(mpven)-VAL(inicia) + 1 TO lugar
   GO LUGAR
   DO WHILE mpven ) NUMPVEN .AND. .NOT. EOF
      SKIP
   ENDDO
   DO WHILE mpven ( NUMPVEN .AND. # ) 0
      SKIP-1
   ENDDO
   IF NUMPVEN = mpven
      SET FORMAT TO PRINT
      SET PRINT ON
      SET MARGIN TO 15
      @ 6,1 SAY "PEDIDO DE VENDA" + NUMPVEN + " DATA : " + DATA
      STORE NUMCLN TO mnumcln
      STORE NUMPED TO mpcom
```

Programa 3. MENU5-4.CMD/.PRG. Impressão do pedido de venda

```
      SELE SECO
      USE clientes INDEX Numcln
      FIND &numcln
      @ 9,1 SAY NOME
      @ 9,45 SAY "P.COMPRA DO CLIENTE" + mpcom
      IF .NOT. ATN = "  "
         @ 10,1 SAY ATN
         @ 11,1 SAY ENDERECO
         @ 12,1 SAY TRIM(CIDADE)+ ", "+ESTADO+"    "+ CEP
      ELSE
         @ 10,1 SAY ENDERECO
         @ 11,1 SAY TRIM(CIDADE)+", "+ESTADO+"    "+ CEP
      ENDIF
      USE Detven
      SELE PRIM
      USE Inv INDEX Isbn
      SELE SECO
      DO Busca
      DO WHILE NUMPVEN = mpven .AND. .NOT. EOF
         @ 14,1 SAY "NUMERO ISBN          QTIDADE     PRECO      TITULO"
         STORE 16 TO linha
         DO WHILE NUMPVEN = mpven .AND. .NOT. EOF .AND. linha < 55
            @ linha,1 SAY ISBN
            STORE ISBN TO misbn
            @ linha,20 SAY QTIDADE
            SELE PRIM
            FIND &misbn
            @ linha, 30 SAY PRECO
            @ linha,40 SAY TITULO
            SELE SECO
            SKIP
            STORE linha+1 TO linha
         ENDDO
         EJECT
      ENDDO
      STORE F TO imprimir
   ENDIF
ENDDO
RELEASE linha,imprimir,selecao,inicia,termina,mpven,mpcom,lugar;
      tamsalto,enconprim,misbn
SET MARGIN TO 0
SET FORMAT TO SCREEN
SET PRINT OFF
SELE SECO
USE
SELE PRIM
RETURN
```

Programa 3 (Continuação)

Se, por algum motivo, nosso banco de dados não estiver perfeito, podemos alcançar o registro que queremos, indo para frente ou para trás, com o uso de loops DO após o comando de posicionamento. Se nosso número for maior que o conteúdo de NUMPVEN para o registro em que o posicionamos, isso significará que não avançamos o suficiente. Se for menor, é porque avançamos demais. Somente o número adequado será aplicável. Se tentarmos avançar um número no limite válido e que não esteja no banco de dados, iremos para o início ou para o final do programa sem causar qualquer problema.

Se o registro for encontrado, inserimos uma rotina de apresentação relativamente padronizada – mesmo que ela não use vários arquivos. O item usado neste loop de apresentação é o comando DO BUSCA.

Em nosso arquivo de pedidos de venda, VENDAS, temos um registro para cada pedido, de modo que há uma correlação entre o número do pedido de venda e o número de registro. Em nosso arquivo de detalhes de venda DETVEND, podemos ter vários registros para cada pedido de venda, de modo que não podemos prever exatamente onde se iniciam os dados que queremos. Entretanto, os pedidos de venda ainda estão em ordem seqüencial no arquivo. Há uma série de abordagens que podemos utilizar para localizar os registros que desejamos.

A abordagem que escolhemos pode ser aplicada de modo geral a qualquer arquivo em que os registros estejam em ordem seqüencial. Em qualquer arquivo classificado, a busca da chave da classificação utilizará essa abordagem *se* os registros estiverem em ordem seqüencial. O programa para executar nossa busca é apresentado como Programa 4.

O esquema geral consiste em primeiro verificar o início e o final do banco de dados para verificar se nenhum desses registros é o que queremos e se o registro procurado está no limite válido. A seguir, passamos para o meio. Se o conteúdo for maior, voltamos para uma posição intermediária entre esta e o início; se for menor, avançamos até uma posição intermediária entre esta e o final. Continuamos com esse tipo de movimento sempre dentro do limite cada vez mais restrito do registro que buscamos: a cada vez que saltamos, o salto é de somente a metade do anterior. Quando estivermos dentro do limite e em frente ao registro, podemos usar:

```
LOCATE NEXT N FOR ...
IF ...
```

O procedimento aqui consiste em ir se aproximando sucessivamente de modo que o uso de LOCATE não leve tempo excessivo. Essa técnica é uma variante da "Busca Binária".

Uma outra técnica seria retroceder em passos de 100 até chegar a um pedido que tenha número menor do que aquele que buscamos e então utilizar o comando LOCATE. Se sabemos que o registro que queremos está no final do banco de dados ou próximo a ele, esta será uma abordagem muito mais eficiente. Quanto mais especializadas suas tarefas, melhor uso você poderá fazer do algoritmo de busca.

Uma terceira técnica consiste em acrescentar um campo ao banco de dados VENDAS. Esse campo contém o número de registro do primeiro registro de detalhes do banco de dados DETVEND para o pedido de venda. O campo se torna um "indicador" e nosso sistema se torna semi-hierárquico. Esse é um esquema especialmente eficiente se nunca executamos exclusões aleatórias nesses bancos de dados. Ao fazer uma exclusão, eliminamos somente um bloco de registros no início do banco de dados VENDAS. Após o banco de dados estar compactado, selecionamos DETVEND como arquivo secundário, assim:

```
DELETE WHILE S.NUMPVEN < COM.NUMPVEN
PACK
SELECT PRIMARY
GO TOP
STORE indicador-1 TO ajustar
REPLACE ALL indicador WITH indicador-ajustar
```

Embora esta seja uma técnica especialmente eficiente para se usar neste conjunto de exemplos e que poderia acelerar enormemente partes do processo, ela não é totalmente confiável. Uma área potencial de problemas está na possibilidade de os "indicadores" saírem de ordem; se estiver escrevendo o programa para alguma outra pessoa, nem mesmo chegue a tentar isso. Em termos gerais, o desempenho de alta precisão sempre apresenta um risco.

```
* comando de busca

GO BOTTOM
IF .NOT.  mpven=NUMPVEN
   STORE  #  TO termina
   STORE INT (#/2) TO tamsalto
   SKIP -tamsalto
ENDIF
IF .NOT. mpven=NUMPVEN
   DO WHILE .NOT. mpven=NUMPVEN .AND. tamsalto  >3
      STORE INT(tamsalto/2) TO tamsalto
      IF mpven > NUMPVEN
         SKIP -TAMSALTO
      ENDIF
      IF MPVEN < NUMPVEN
         SKIP -TAMSALTO
      ENDIF
    ENDDO
ENDIF
DO WHILE mpven > NUMPVEN .AND. .NOT. EOF
   SKIP
ENDDO
DO WHILE mpven < NUMPVEN .AND. # >0
   SKIP -1
ENDDO
STORE T TO enconprim
DO WHILE enconprim .AND. # >0
   SKIP -1
   IF MPVEN#NUMPVEN
      SKIP
      STORE F TO enconprim
      LOOP
   ENDIF
ENDDO
```

Programa 4. Comando de busca

Item 5 de Menu Relatório de Vendas Diárias (MENU5-5.CMD/.PRG)

O Programa 5 prepara um relatório de vendas diárias ou um resumo das atividades diárias, que teria a aparência da Tela 2.

Nosso primeiro passo será determinar o número de pedidos de venda durante o dia. Nós o fazemos escolhendo VENDAS como nosso arquivo primário e armazenando o último número de registro a uma variável de memória *fim* com STORE # TO *fim*.

A seguir voltamos atrás no banco de dados um registro por vez até chegarmos ao início do banco de dados (# = 0) ou então verificamos que a data difere da data desejada. O número dos pedidos é simplesmente a diferença entre o número *fim* de registros armazenados e o número atual do registro. Observe que o número atual de registro refere-se ao registro logo antes do primeiro registro da data que desejamos. Vamos armazenar esse resultado como uma cadeia de caracteres à variável *vendas:*

STORE STR(*fim*-#,5) TO *vendas*

Agora, se avançarmos um registro, estaremos posicionados no primeiro pedido de vendas do dia. Precisamos armazenar o número do pedido em uma variável *mpven*, a variável usada por nossa rotina de busca na última seção. DETVEND é selecionado como nosso arquivo secundário. Então usamos o programa BUSCA para posicionar no primeiro registro de pedido de venda do dia no arquivo secundário DETVEND.

O próximo passo será obter a quantidade total de livros vendidos durante o dia, o valor desses livros e o número de títulos (registros no arquivo secundário) que demos entrada durante o dia. Tudo isso é feito por:

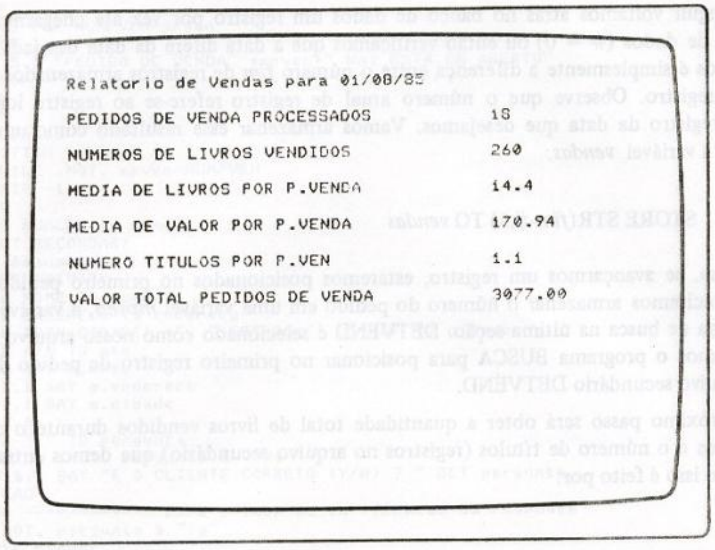
SUM NEXT 6500 QTDADE,QTIDADE*PREÇO,1 TO
- *vendido,valor,entradas*

Por que os *próximos* 65000? Queremos somar as expressões relacionadas desde o ponto em que estávamos no banco de dados até o final dele. O comando SUM encerra automaticamente no final do banco de dados. O comando NEXT faz com que o comando SUM parta do registro posicionado. Escolhemos 65000 somente para garantir que estamos usando um número maior que o número de registros entre a posição em que estamos e o final do banco de dados.

O passo final será imprimir o relatório. Já reunimos as informações necessárias. O número de livros vendidos no dia está na variável *vendido*. O número de livros por pedido é simplesmente o total de livros vendidos dividido pelo número de pedidos de venda. Observe que tivemos que multiplicar a expressão *vendido/vendas* por 1.0.

O número de casas decimais apresentado como resultado da operação aritmética do dBASE II depende do número de casas decimais usado diretamente na expressão. Esse procedimento é tecnicamente correto, embora dê um pouco de trabalho. Por exemplo, se dividirmos quatro por cinco, o dBASE II nos dirá 4/5 = 0, embora tenha de fato realizado o cálculo aritmético de modo correto. Ele apresentou o resultado de acordo com as regras das operações aritméticas.

Para que o dBASE II apresente o resultado que queremos, precisamos ou incluir o número desejado de casas decimais na expressão, como em *vendidos/vendas**1.0, ou podemos apresentar o resultado como parte de uma operação com *strings*, como em STR(*entradas/vendas*,10,1). Com a operação de *strings*, apresentamos o quociente de *entradas/vendas* como uma *string* de caracteres que possui um tamanho de 10, inclusive uma casa decimal.



```
Relatorio de Vendas para 01/08/85

PEDIDOS DE VENDA PROCESSADOS          18

NUMEROS DE LIVROS VENDIDOS            260

MEDIA DE LIVROS POR P.VENDA           14.4

MEDIA DE VALOR POR P.VENDA            170.94

NUMERO TITULOS POR P.VEN              1.1

VALOR TOTAL PEDIDOS DE VENDA          3077.00
```

Tela 2. Relatório de vendas diárias

```
* Programa .....:MENU5-5.CMD  (ou MENU5-5.PRG)
* Obs...........:Este  programa produz um  relatorio  de  Vendas Diarias

ERASE
SELE PRIM
USE Vendas
GO BOTTOM
STORE  #  TO fim
STORE DATA TO vendasdia
DO WHILE DATA = vendasdia .AND.  # >0
   SKIP -1
ENDDO
STORE fim-# TO vendas
SKIP
STORE NUMPVEN TO mpven
SELE SECO
USE Detvend
DO Busca
SUM NEXT 65000 QTIDADE,QTIDADE*PRECO,1 TO vendidos,valor,entradas
SET FORMAT TO PRINT
SET MARGIN TO 15
@ 6,1   SAY  "Relatorio de Venda para "+DATA
@ 9,1   SAY  "PEDIDOS DE VENDA PROCESSADOS"
@ 9,40  SAY  vendas
@ 11,1  SAY  "NUMERO DE LIVROS VENDIDOS"
@ 11,40 SAY  vendidos
```

Programa 5. MENU5-5.CMD/.PRG Relatório de vendas diárias


```
@ 13,1  SAY  "MEDIA DE LIVROS POR P.VENDA "
@ 13,40 SAY  vendidos/vendas*1.0
@ 15,1  SAY  "MEDIA DE VALOR POR P.VENDA "
@ 15,40 SAY  valor/vendas
@ 17,1  SAY  "NUMERO DE TITULOS POR P.VEN "
@ 17,40 SAY  STR(entradas/vendas,10,1)
@ 19,1  SAY  "VALOR TOTAL PEDIDOS DE VENDA "
@ 19,40 SAY  valor
RELEASE fim,fimvendas,fimdetalhe,vendas,vendasdia,mpven;
        enconprim,tamsalto,termina
EJECT
SET FORMAT TO SCREEN
SELE SECO
USE
SELE PRIM
USE
RETURN
```

Programa 5 (Continuação)

Item 6 de Menu Apresenta os Pedidos de Vendas Pendentes (MENU5-6.CMD/.PRG)

O arquivo PEDABR, a lista de pedidos de vendas que não foram ainda preenchidos, é o arquivo que controla os processos que utilizam o sistema de pedidos de venda. Como os pedidos de vendas são eliminados deste arquivo após estarem preenchidos, ele não será muito grande.

De tempos em tempos vamos querer saber em que estado se encontra esse arquivo, que pedidos não foram preenchidos e a quem eles pertencem. O Programa 6 tem a finalidade de apresentar na tela esse estado, com a opção de uma cópia impressa.

São usados três arquivos para preparar esse relatório: PEDABR, VENDAS e CLIENTES. A Tela 3 mostra um exemplo do resultado desse programa.

Os itens apresentados são o número do pedido de venda, a data de entrada do pedido, o número do pedido de compra do cliente (se houver), o nome e o código de identificação do cliente.

Ao preparar esse relatório, passamos pelo arquivo PEDABR um registro (pedido de venda) por vez. Para cada registro neste arquivo, primeiramente selecionamos nossos arquivos VENDAS como o arquivo secundário, vamos ao fim do arquivo VENDAS e retrocedemos até encontrarmos o registro em que o número do pedido de venda corresponde ao número do pedido de venda em nosso arquivo primário. Então, apresentamos o número do pedido de venda, a data, o número do pedido de compra e o código do cliente, com o uso do comando ? (poderíamos também ter usado DISPLAY OFF).

Não apresentamos o nome, embora queiramos que ele seja apresentado na mesma linha em que os outros itens. Para obtê-lo, armazenamos o número do pedido de venda na variável de memória *mnumcln*. Agora selecionamos o arquivo CLIENTES indexado em IDCLIENTE. A passagem de um arquivo secundário para outro é lenta. Entretanto, a velocidade não é muito importante, já que o número de registros a ser processado é pequeno, com uma média de 20 por dia.

A seguir, buscamos o registro desejado no arquivo de clientes e, com o comando ??, apresentamos o nome do cliente. Esse comando é o mesmo que o de um sinal de interrogação, exceto que *não* produz um retorno de carro ou uma nova linha. Ele apresenta itens a partir da última posição do cursor ou da cabeça de impressão.

Se decidirmos não imprimir a tela, o comando ACCEPT será emitido (com < RETURN >), que fará com que a apresentação na tela pare. Isso evita que o programa se encerre e retorne ao menu antes que tenhamos terminado de ler as informações na tela.

Item 7 de Menu Apaga Registros de Pedidos de Venda Superados (MENU5-7.CMD/.PRG)

A fim de eliminar registros com mais de um ano, use o Programa 7 para limpar os dois arquivos de banco de dados VENDAS e DETVEND. Como nenhum desses arquivos está indexado, este programa é relativamente simples e muito mais rápido do que seria se arquivos tão grandes fossem indexados.

Os registros que queremos excluir dos dois arquivos se apresentam em blocos no início de cada um deles. Primeiramente excluímos os registros do arquivo VENDAS. Aqui podemos usar um critério simples de datas. Vamos, literalmente, eliminar tudo que tiver mais de um ano pela subtração de uma unidade ao ano, na data do sistema atual.

A data atual anteriormente armazenada na introdução do sistema de menu é a variável *mdata*. Essa variável está na forma DD/MM/AA. Para usar a data como parte de uma operação de comparação, precisamos invertê-la para a forma AA/MM/DD. Isso pode ser realizado fisicamente pela criação de variáveis desse tipo ou, logicamente, designando pelos nomes as unidades no interior da cadeia de caracteres de comando.

```
P. VEN.  #  DATA DE     P.  COM. #   NUM #   NOME

100010       03/01/85                100001  VOLUMES TECNICOS
100011       08/01/85    45          100002  LIVRARIA MUSEU
100013       08/01/85    89          100002  LIVRARIA MUSEU
100015       03/01/85    99          100003  LIVRARIA ODISSEIA
100015       08/01/85    45678/78    100001  VOLUMES TECNICOS
Teclar (RETURN) para voltar ao menu.
```

Tela 3. Tela de pedidos em aberto


```
* Programa ......:MENU5-6.CMD  (ou MENU5-6.PRG)
* Obs. ..........:Este programa fornece o Relatorio dos  Pedidos de Venda em
* ...............:aberto.

ERASE
@  1,30 SAY "PEDIDOS DE VENDA EM ABERTO DESDE "+DATE()
STORE "  " TO pergunta
@  3,1 SAY "DESEJA UMA COPIA IMPRESSA (Y/N)?  " GET  pergunta PICTURE  "!"
READ
@  3,0
IF pergunta = "Y"
   SET PRINT ON
   SET FORMAT TO PRINT
   SET MARGIN TO 15
   @ 6,1  SAY  "PEDIDOS DE VENDA EM ABERTO DESDE" +DATE()
   ?
ENDIF
? "P. VEND.  #     DATA DE      P. VEND.  #       NUM  #       NOME"
SELE PRIM
USE Pedabr
DO WHILE .NOT. EOF
   SELE SECO
   USE Vendas
   GO BOTTOM
   DO WHILE .NOT. NUMPVEN = P.NUMPVEN .AND.  #>0
      SKIP -1
   ENDDO
   ? NUMPVEN, DATA, NUMPED, NUMCLN
   STORE NUMCLN TO mnumcln
   USE Clientes INDEX Numcln
   FIND  &mnumcln
   ??  "  ", NOME
   SELE PRIM
   SKIP
ENDDO
IF .NOT. pergunta = "Y"
   ACCEPT  "Teclar (RETURN) para voltar ao menu. " TO pergunta
ENDIF
RELEASE mpven,pergunta,selecao,mnumcln
EJECT
SET MARGIN TO 0
SET PRINT OFF
SET FORMAT TO SCREEN
SELE SECO
USE
SELE PRIM
RETURN
```

Programa 6. MENU5-6.CMD/.PRG Relatório de pedidos de vendas em aberto

```
* PROGRAMA .......:MENU6.CMD  (ou MENU6.PRG)
* Obs. ...........:Este e o programa  para  preenchimento  de Faturas.
DO WHILE T
   STORE "  " TO escolha
   ERASE
   @ 2,30 SAY "DISTRIBUIDORA FERREIRA"
   @ 4,31 SAY "MENU PARA PREENCHIMENTO DE PEDIDOS"
   @ 6,34 SAY cdata
   @ 8,25 SAY "1 - Preencher um Pedido de Vendas"
   @ 9,25 SAY "2 - Apresentar os Pedidos Pendentes"
   @ 10,25 SAY "3 - Reimprimir uma Fatura"
   @ 15,25 SAY "0 - VOLTAR AO MENU PRINCIPAL"
   @ 18,25 SAY "  - ESCOLHA UMA"
   @ 18,24 SAY escolha
   READ NOUPDATE
   IF escolha = "0"
      RELEASE Ultrent,escolher
      RETURN
   ENDIF
   IF escolha $"1234567"
      STORE "MENU6-"+escolha TO escolher
      IF FILE ("&escolher..CMD")
         DO &escolher
      ENDIF
   ENDIF
ENDDO
```

Programa 7. MENU5-7.CMD/.PRG Apaga registros com mais de um ano

Em nosso exemplo, usamos esses dois procedimentos. Primeiramente, criamos a variável *subst data*, que é efetivamente do tipo AA/DD/MM/. O fato de as barras estarem no lugar errado não importa, desde que o elemento de comparação tenha as barras na mesma posição que *subst data*. A variável *substdata* se inicia com o ano, que é um a menos que o ano atual.

Os regitros são apagados, onde a data "reorganizada" corresponder a *subst data*. Usamos DELETE WHILE em vez de DELETE FOR para evitar a leitura de todo o banco de dados, o que acelera sensivelmente a operação.

Uma vez compactado o banco de dados VENDAS, é necessário retornar ao início e armazenar o primeiro número de pedido de venda na variável *mpven*. O comando GO TOP é o comando do dBASE II para posicionar o usuário no início do banco de dados. Se o banco de dados não estiver indexado, GO TOP levará ao Registro Um, o primeiro registro do banco de dados. Se o banco de dados estiver indexado, o comando levará até o primeiro registro lógico do banco de dados.

A seguir, selecionamos o arquivo DETVEND. Novamente, queremos excluir um bloco de registros no início do arquivo de bancos de dados. O comando USE nos posiciona no Registro Um no arquivo DETVEND. Queremos eliminar todos os registros até, mas não inclusive, o registro em que o número de pedido de venda está armazenado em *mpven*. Para isso, utilizamos novamente o comando DELETE WHILE.

A exclusão e a compactação de registros em bancos de dados de grande volume é uma operação lenta. A compactação exige a reordenação física do arquivo de banco de dados. Se excluirmos 100 registros desde o início do arquivo, o primeiro registro restante será o Registro 101.

Quando compactamos o arquivo por meio do comando PACK, o Registro 101 é fisicamente deslocado para a posição de disco anteriormente ocupada pelo Registro Um. O Registro 102 é deslocado para a posição do Registro Dois, e assim por diante. Esse tipo de operação em disco é lento. Em muitos sistemas de computação, especialmente nas unidades de disco Winchester, não há muita indicação de que o computador esteja realmente fazendo algo.

Em nosso programa exemplo, primeiramente apresentamos a mensagem de que o processo pode levar muito tempo. Então usamos o comando SET TALK ON para que a apresentação da atividade normal do dBASE II seja apresentada na tela, possibilitando ao usuário perceber alguma coisa do que o computador está fazendo. Temos que nos lembrar de restabelecer SET TALK OFF antes de sair deste programa.

Capítulo Nove

PREENCHIMENTO E FATURAMENTO DE PEDIDOS

Após inserir um pedido no computador, o passo seguinte será preenchê-lo e preparar uma fatura. A fatura é impressa para o cliente e armazenada no arquivo de faturas para ser usada por Contas a Receber no faturamento mensal.

Outra tarefa de preenchimento de pedidos será montar um sistema para livros com pedidos em pendência. Se um cliente solicitar livros que estão faltando no estoque, precisamos criar um registro informando que devemos ao cliente um certo número de cópias de um determinado livro ao preço do pedido original. Ao chegarem os livros, nós os enviamos ao cliente e preparamos uma fatura para cobrança.

O sistema de preenchimento de pedidos consistirá nos arquivos de pedidos de venda, arquivo de inventário, arquivo de clientes, arquivos de faturas e um arquivo de pedidos pendentes (ver Figura 1).

Neste capítulo, vamos examinar programas para converter os registros de pedidos de venda em faturas e novamente em registros de venda e um programa para imprimir a fatura do cliente. No Programa 1, o programa menu para o sistema de preenchimento de pedidos, essas tarefas são apresentadas como o Item 1, Arquivar um Pedido de Venda e o Item 3, Reimprimir uma Fatura. Os programas desenvolvidos nesses itens ilustram um dos mais importantes conceitos da programação estruturada: a chamada de outros programas.

O programa menu em si foi extraído de menus anteriores e modificado para formar o Menu de Preenchimento de Pedidos. O segundo grupo IF é de interesse especial nesse menu. Nesse grupo, há seleções de programas não existentes porque copiamos o menu do Menu Principal, no Capítulo 4. Para nos protegermos contra erros de sintaxe, usamos a função de arquivo do dBASE II:

```
IF FILE("nomedearquivo")
```


Esta função é usada para verificar a existência de um arquivo em um disco. O nome de arquivo deve incluir o identificador de arquivo e estar colocado entre os delimitadores e parênteses.

Em nosso exemplo, armazenamos uma parte do nome de arquivo na variável *escolher*. Poderíamos ter feito a concatenação de .CMD com *escolher* para que escolher contivesse todo o nome de arquivo. Então teríamos IF FILE ("*&escolher*) em vez de IF FILE ("*&escolher*"..CMD"). Se tivéssemos escolhido o Item 1 de Menu, por exemplo, seria então IF FILE("MENU6-1.CMD").

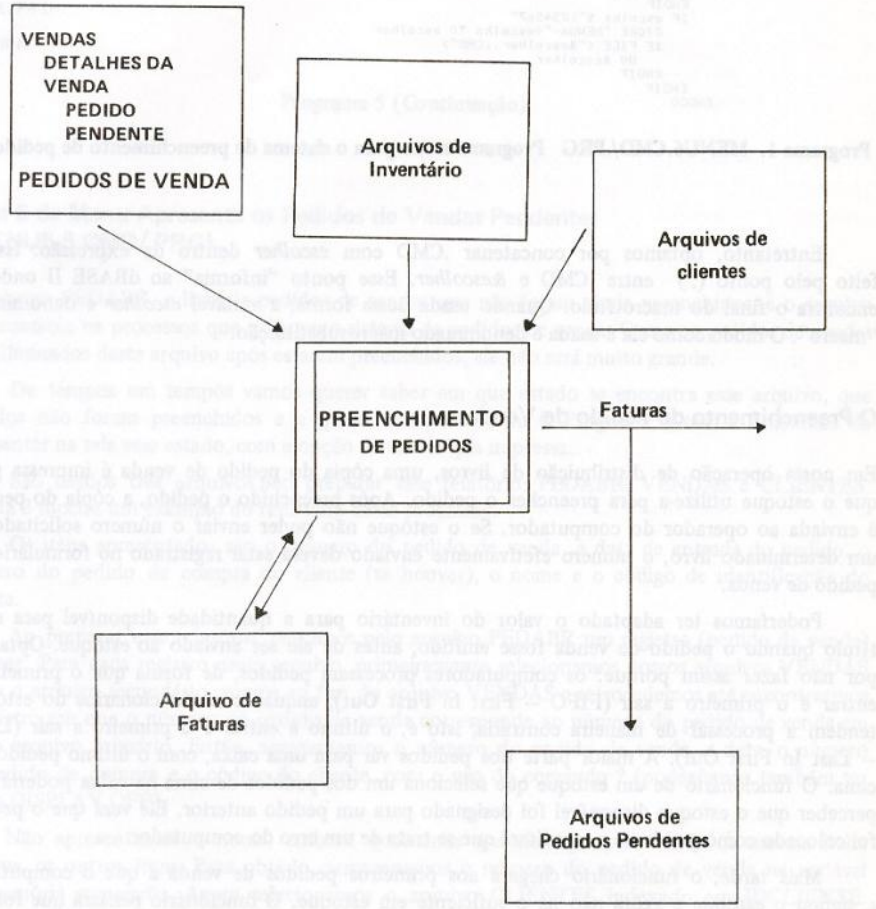


Figura 1. Diagrama do sistema de preenchimento de pedidos

```
* PROGRAMA .....:MENU6.CMD  (ou MENU6.PRG)
* Obs. .........:Este e o programa  para  preenchimento  de Faturas.
DO WHILE T
   STORE "  " TO escolha
   ERASE
   @ 2,30 SAY "DISTRIBUIDORA FERREIRA"
   @ 4,31 SAY "MENU PARA PREENCHIMENTO DE PEDIDOS"
   @ 6,34 SAY cdata
   @ 8,25 SAY "1 - Preencher um Pedido de Vendas"
   @ 9,25 SAY "2 - Apresentar os Pedidos Pendentes"
   @ 10,25 SAY "3 - Reimprimir uma Fatura"
   @ 15,25 SAY "0 - VOLTAR AO MENU PRINCIPAL"
   @ 18,25 SAY "  - ESCOLHA UMA"
   @ 18,24 SAY escolha
   READ NOUPDATE
   IF escolha = "0"
      RELEASE Ultrent,escolher
      RETURN
   ENDIF
   IF escolha $"1234567"
      STORE "MENU6-"+escolha TO escolher
      IF FILE ("&escolher..CMD")
         DO &escolher
      ENDIF
   ENDIF
ENDDO
```

Programa 1. MENU6.CMD/.PRG Programa menu para o sistema de preenchimento de pedidos

Entretanto, optamos por concatenar .CMD com *escolher* dentro da expressão. Isso é feito pelo ponto (.) entre .CMD e *&escolher*. Esse ponto "informa" ao dBASE II onde se encontra o final do macrotítulo. Quando usada dessa forma, a variável *escolher* é denominada "macro". O modo como ela é usada é denominado macrosubstituição.

O Preenchimento do Pedido de Venda

Em nossa operação de distribuição de livros, uma cópia do pedido de venda é impressa para que o estoque utilize-a para preencher o pedido. Após preenchido o pedido, a cópia do pedido é enviada ao operador do computador. Se o estoque não puder enviar o número solicitado de um determinado livro, o número efetivamente enviado deverá estar registrado no formulário de pedido de venda.

Poderíamos ter adaptado o valor do inventário para a quantidade disponível para cada título quando o pedido de venda fosse emitido, antes de ele ser enviado ao estoque. Optamos por não fazer assim porque: os computadores processam pedidos, de forma que o primeiro a entrar é o primeiro a sair (FIFO – First In First Out), enquanto os funcionários do estoque tendem a processar de maneira contrária, isto é, o último a entrar é o primeiro a sair (LIFO – Last In First Out). A maior parte dos pedidos vai para uma caixa, com o último pedido em cima. O funcionário de um estoque que seleciona um dos pedidos de cima na caixa poderia não perceber que o estoque disponível foi designado para um pedido anterior. Ele verá que o pedido foi colocado como pendente e acreditará que se trata de um erro do computador.

Mais tarde, o funcionário chegará aos primeiros pedidos de venda a que o computador designou o estoque e agora não há o suficiente em estoque. O funcionário pensará que foi um novo erro do computador. Todos os registros nesses pedidos são novamente corrigidos no escritório para adaptar o computador à realidade do estoque. É muito mais simples deixar o estoque controlar o inventário em primeiro lugar – especialmente porque provavelmente ele

o fará de qualquer forma. A adaptação das quantidades em estoque, após o pedido ser preenchido, corresponde melhor à realidade de nosso inventário. Também exige software mais simples.

Nos sete programas deste capítulo, o preenchimento de pedidos é manipulado pelos Programas 2, 3, 4 e 6. O programa principal neste grupo é o Programa 2. O Programa 3 é um auxiliar do Programa 2 e incorpora o programa de busca usado no último capítulo. A Figura 2 apresenta um diagrama do programa geral.

Examinando o Programa 2, notaremos que ele é constituído principalmente por vários testes de um ou de outro tipo. O restante do programa consiste em procedimentos para deslocar e manipular registros. O Programa 2 utiliza nove arquivos de banco de dados individuais, dos quais apenas três, CLIENTES, FATURAS e INV (inventário), estão indexados.

A seqüência de atividades nesse programa é a seguinte:

- Identificar o pedido de venda para o computador, digitar o número do pedido de venda.
- Verificar se foi fornecido o pedido de venda correto, apresentar o nome e o endereço do cliente.
- Verificar quanto a itens em pendência, fazer um processamento especial de pedidos pendentes.
- Atribuir um número de fatura à transação (pode haver mais de uma fatura por pedido de venda).
- Criar registros semelhantes a registros de pedidos de venda.
- Imprimir a fatura.

O Programa se inicia com a escolha de PEDABR como arquivo primário e o arquivo CLIENTES, indexado pelo código de identificação do cliente, como o arquivo secundário. O arquivo PEDABR contém somente os números dos pedidos de venda que não foram preenchidos.

```
* Programa .....:Menu6-1.CMD/.PRG
* Obs. .........:Programa  para  fechar  pedidos de  venda  e preparar
* ..............:faturas.

ERASE
STORE T TO programa
USE Pedabr
SELECT SECONDARY
USE Clientes INDEX Numcln
SELE PRIM
DO WHILE programa
   @ 1,20 SAY "SISTEMA DE FATURAS E PREENCHIMENTO DE PEDIDOS"
   STORE "      " TO mpven
   @ 3,1 SAY "ENTRE O NUMERO DO PEDIDO DE VENDA" GET mpven PICTURE;
   "999999"
   READ
   IF mpven="      "
      STORE F TO programa
      LOOP
```

Programa 2. MENU6-1.CMD/.PRG Programa para preparar faturas para pedidos preenchidos

```
   ENDIF
   LOCATE FOR NUMPVEN=mpven
   IF EOF
      @ 5,1 "PEDIDO DE VENDA" +mpven+" NAO ESTA EM ABERTO"
      LOOP
   ENDIF
   @ 5,0
   USE Vendas
   GO BOTTOM
   DO WHILE .NOT. mpven=NUMPVEN
      SKIP -1
   ENDDO
   STORE NUMCLN TO numcln
   SELECT SECONDARY
   FIND &mnumcln
   SELECT PRIMARY
   STORE NOME TO nome
   STORE ENDERECO TO endereco
   STORE TRIM(CIDADE) +", "+ESTADO+'   "+CEP TO cidade
   STORE ATN TO atn
   @ %,1 SAY m.nome
   @ %+1,1 SAY m.endereco
   @ %+i,1 SAY m.cidade
   @ %+2,0
   STORE " " TO pergunta
   DO WHILE .NOT. pergunta $ "YNyn"
      @ %,1 SAY "E O CLIENTE CORRETO (Y/N) ? " GET pergunta
      READ
   ENDDO ------------ para o loop de verificacao da resposta
   IF .NOT. pergunta $ "Yy"
      USE Pedabr
      ERASE
      LOOP
   ENDIF
   CLEAR GETS
   STORE "  " TO pergunta
   DO WHILE .NOT. pergunta $ "YNyn"
      @  %,1 SAY "O PEDIDO ESTA COMPLETAMENTE PREENCHIDO (Y/N) ?  " GET pergunta
      READ
   ENDDO ------------ para o loop de verificacao da resposta
   DO Menu6-1a
   STORE mpven TO retempven
   STORE m.fatura TO mpven
   STORE DATE() TO data
   IF pergunta$"Nn"
      DO Menu6-1b
   ENDIF
   DO Menu6-3p
   USE Faturas
   INDEX I-numcln
   APPEND BLANK
   REPLACE  FATURA WITH m.fatura,  DATA WITH DATE(),  NUMCLN  WITH numcln;
         NUMPED WITH mnumped, QUANTIA WITH m.quantia
   USE Pedabr
   LOCATE FOR retempven
   DELETE
   PACK
   ERASE
   SELECT SECONDARY
   USE Clientes INDEX Numcln
   SELECT PRIMARY
ENDDO ---------------------- para o loop do programa principal
RELEASE programa,mpven,mnumcln,nome,cidade,endereco,atn,pergunta;
       fatura,termina,tamsalto,enconprim,retempven,data,quantia,pagina;
       linha,mrsbn
CLOSE SECO
USE
CLOSE PRIM
RETURN
```

Programa 2 (Continuação)

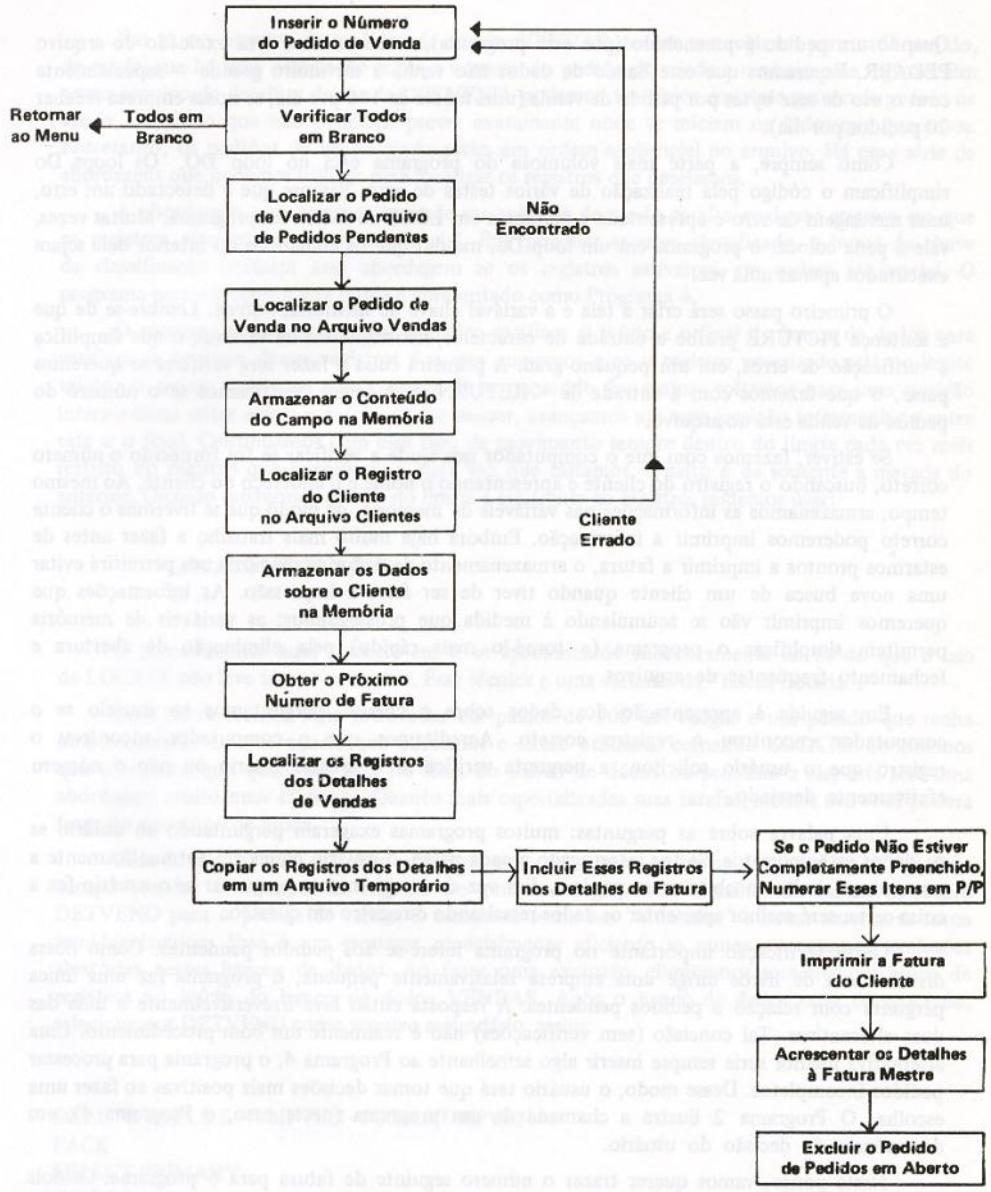


Figura 2. Diagrama do preenchimento do pedido

Quando um pedido é preenchido (por esse programa), o seu número será excluído do arquivo PEDABR. Esperamos que este banco de dados não venha a ser muito grande – especialmente com o uso de sete bytes por pedido de venda (uma média de 140 por dia, se nossa empresa receber 20 pedidos por dia).

Como sempre, a parte mais volumosa do programa está no loop DO. Os loops Do simplificam o código pela realização de vários testes de erro. Sempre que é detectado um erro, uma mensagem de erro é apresentada e voltamos em LOOP ao início do programa. Muitas vezes, vale a pena colocar o programa em um loop Do, mesmo que os comandos no interior dele sejam executados apenas uma vez.

O primeiro passo será criar a tela e a variável chave de memória, *mpven*. Lembre-se de que a sentença PICTURE proíbe a entrada de caracteres, não-numérica na variável, o que simplifica a verificação de erros, em um pequeno grau. A primeira coisa a fazer será verificar se queremos parar, o que fazemos com a entrada de <RETURN>. A seguir, verificamos se o número do pedido de venda está no arquivo.

Se estiver, fazemos com que o computador nos ajude a verificar se foi fornecido o número correto, buscando o registro do cliente e apresentando o nome e o endereço do cliente. Ao mesmo tempo, armazenamos as informações nas variáveis de memória, de modo que se tivermos o cliente correto poderemos imprimir a informação. Embora haja muito mais trabalho a fazer antes de estarmos prontos a imprimir a fatura, o armazenamento de dados na memória nos permitirá evitar uma nova busca de um cliente quando tiver de ser feita a impressão. As informações que queremos imprimir vão se acumulando à medida que prosseguimos; as variáveis de memória permitem simplificar o programa (e torná-lo mais rápido) pela eliminação da abertura e fechamento freqüentes de arquivos.

Em seguida à apresentação dos dados sobre o cliente, perguntamos ao usuário se o computador encontrou o registro correto. Acreditamos que o computador encontrou o registro que o usuário solicitou; a pergunta verifica se o usuário inseriu ou não o número efetivamente desejado.

Uma palavra sobre as perguntas: muitos programas exageram perguntando ao usuário se os dados estão corretos. Se for interrogado a cada passo, o usuário começará automaticamente a responder "sim" e o objetivo se perderá. Em vez de simplesmente perguntar se o usuário fez a coisa certa, será melhor apresentar os dados ressaltando o registro em questão.

Outra verificação importante no programa refere-se aos pedidos pendentes. Como nossa distribuidora de livros dirige uma empresa relativamente pequena, o programa faz uma única pergunta com relação a pedidos pendentes. A resposta então leva irreversivelmente a uma das duas alternativas. Tal concisão (sem verificações) não é realmente um bom procedimento. Uma alternativa melhor seria sempre inserir algo semelhante ao Programa 4, o programa para processar pedidos incompletos. Desse modo, o usuário terá que tomar decisões mais positivas ao fazer uma escolha. O Programa 2 ilustra a chamada de um programa (neste caso, o Programa 4), em decorrência da decisão do usuário.

Neste ponto, vamos querer trazer o número seguinte de fatura para o programa. Os dois métodos principais para reter um número de fatura são gravá-lo em um arquivo de memória em disco (.MEM) ou em um banco de dados. Optamos por usar um banco de dados com um registro em um campo, FATURA. O nome de nosso banco de dados é CONTROLES.

A técnica de se usar um banco de dados para armazenar vários parâmetros de controle usados em um sistema de programa deve ser avaliada com relação a cada sistema de software. A tarefa será abrir o arquivo, extrair o parâmetro, aumentá-lo e fechar o arquivo. Embora seja tentador (e fácil) abrir o sistema uma única vez e operá-lo inteiramente a partir de variáveis, se acontecer algo errado – se alguém tropeçar no fio de abastecimento de energia do computador, por exemplo – a recuperação poderá ser mais difícil do que se tivéssemos um único aumento de parâmetro para recuperar.

Observe que o número de fatura está armazenado como uma cadeia de caracteres.

O passo seguinte será usar o arquivo DETVEND, que contém em detalhes os registros de vendas sobre o pedido de venda que está sendo preenchido. O uso de DETVEND automaticamente retém o arquivo CONTROLE anterior e o grava novamente no disco. Para encontrar o registro desejado em DETVEND, "chamamos" o programa de busca pseudobinário BUSCA (Programa 4, no Capítulo 8). Isso nos posiciona no primeiro registro com o número de pedido de venda que solicitamos.

```
* Programa ---------- : MENU6-1A.CMD/.PRG
* Obs. -------------- : chamado de Menu6-1, processa pedidos completa-
* ------------------- : mente preenchidos. Os bancos de dados iniciais
* ------------------- : vendas prim e clientes second.

SECOND SECONDARY
USE controles
IF # = 0
   APPEND BLANK
   REPLACE FATURA WITH 200000
ENDIF
REPLACE FATURA WITH FATURA+1
STORE STR(fatura,6) TO fatura
STORE NUMPED TO numped
SELECT PRIMARY
USE Detvend
Do Busca
* ----------: Neste ponto, o pedido de venda se subdivide em duas
* ----------: partes -- os itens faturados e os pedidos pendentes.
* --- Nao se esqueca de acrescentar a quantidade solicitada em
* --- algum ponto.
* --- O arquivo Faturado contem o pedido completo com o campo
* --- Qtidenv acrescentado.
COPY TO arqT WHILE NUMPVEN=mpven
USE arqT
REPLACE ALL NUMPVEN WITH m.fatura,QTIDENV WITH QTID
USE Faturado
APPEND FROM arqT
DELETE FILE arqT
RETURN
```

Programa 3. MENU6-1A.CMD/.PRG. Programa para processar pedidos completos. Programa auxiliar do MENU6-1.

O objetivo aqui é o de passar registros de venda ou para o arquivo de pedidos pendentes ou para o arquivo de faturas FATURADO, que é o equivalente ao arquivo DETVEND no sistema de faturas. A estrutura do arquivo FATURADO é a mesma de DETVEND, exceto pela adição de um campo QTIDENV, a quantidade efetivamente enviada. Na maioria dos casos, a quantidade enviada (QTIDENV) será a mesma que a quantidade solicitada (QTID). A Figura 3 mostra o plano do banco de dados para os dois arquivos de faturas.

Primeiramente passamos todos os registros no pedido de venda em questão para um arquivo temporário. Então, "anexamos" os registros no arquivo temporário no final do arquivo FATURADO. A Figura 4 ilustra isso.

O processo de passar os registros pode parecer um pouco tortuoso, mas é em última análise mais rápido do que a abordagem alternativa, que seria USE FATURADO e APPEND FROM DETVEND = *mpven*. Embora essa abordagem alternativa seja mais simples, ela faz com que o dBASE II leia DETVEND por inteiro. Como DETVEND é um arquivo grande (com mais de um megabyte), essa abordagem "mais simples" efetivamente levará mais tempo. O processo de cópia que utilizaremos lerá somente os registros que nos interessam.

Em nosso plano para o arquivo FATURADO, usamos o nome de campo NUMPVEN para o campo que contém os números de índices. Isso nos permite usar a operação APPEND FROM para acrescentar os registros provenientes do arquivo temporário ARQ T ao arquivo FATURADO: aqui vamos precisar do arquivo inteiro, assim APPEND FROM será a opção "mais rápida".

Observe que substituímos o número de pedido de venda no arquivo de índice antes de passar esses registros para o arquivo de faturas. Isso é feito prevendo a possibilidade de que o número do pedido de venda seja o mesmo que o último número de fatura usado. Quando os registros tiverem sido passados para o arquivo de faturas, o campo QTIDENV será zero porque não há um campo QTIDENV no arquivo de vendas. Inicialmente, a quantidade enviada é atribuída à quantidade solicitada. Isso nos permite modificar a quantidade enviada como uma exceção, pois a maioria dos itens está completamente preenchida. Nós tiramos proveito dessa situação quando entramos os dados de pedidos pendentes, se houver.

"Chamamos" o Programa 4, para inserir informações sobre os pedidos pendentes, somente se tivermos inserido um "não" como resposta à pergunta: "O pedido está completamente preenchido?" O Programa 4 permite-nos identificar registros onde houver um pedido pendente, por meio de uma técnica relativamente simples. Solicitamos ao usuário que entre cada número ISBN que esteja pendente. Localizamos o registro fazendo uma busca no banco de dados próximo ao fim, já que sabemos que estamos lidando com os últimos registros do banco de dados FATURADO. Então, permitimos ao usuário modificar a quantidade enviada (QTIDENV).

Esse processo continua até que o usuário tecle <RETURN> em vez do número ISBN. Nesse ponto, o projetista do programa deve tomar uma decisão relativa a "fatores humanos". O usuário tem muito maior possibilidade de teclar inadvertidamente <RETURN> que de digitar uma resposta mais ponderada como PARE ou FIM. Podemos, de modo igualmente fácil, detectar um PARE ou um FIM na variável de memória, da mesma forma que detectamos um "tudo branco". Em ambos os casos, deve-se prever a possibilidade de editar esses arquivos para corrigir erros. Isso nem sempre está presente em programas para usuários com experiência no dBASE II, mas para os que não têm muita experiência é uma necessidade.


```
ARQUIVO DE BANCO DE DADOS: FATURAS

                                  NOME DE CAMPO   TIPO    TAM     DEC
Numero de fatura                  FATURA          C       006
Numero de identificacao do cliente NUMCLN         C       006
Numero do pedido de comp.do cliente NUMPED        C       008
Data da fatura                    DATA            C       008
Valor da fatura                   QUANTIA         N       008     002


ARQUIVO DE BANCO DE DADOS: FATURADO

Numero da fatura                  NUMPVEN         C       006
Numero ISBN                       ISBN            C       013
Quantidade solicitada             QTIDADE         N       003
Quantidade enviada                QTIDENV         N       003
Preco                             PRECO           N       006
```

Figura 3. Plano do banco de dados para arquivos de fatura

```
* Programa ------- Menu 6-1B.cmd/ .prg
* Obs. ----------- Programa para entrar pedidos pendentes. O arquivo
* ---------------- em uso e Faturado

STORE "x" TO misbn
ERASE
DO WHILE .NOT. misbn = "    "
   STORE "              " TO misbn
   ?
   @ 23,1 SAY "NUMERO ISBN" GET misbn
   READ
   IF .NOT. misbn = "      "
      GO BOTTOM
      DO WHILE mpven=NUMPVEN .AND. .NOT. ISBN=misbn .AND. .NOT. #=0
         SKIP -1
      ENDDO
      IF .NOT. misbn=ISBN
         @ 23,30 SAY "NAO LOCALIZADO O ISBN "+misbn
         LOOP
      ENDIF
      @ 23,30 SAY "FORNECER A QUANTIDADE ENVIADA" GET QTIDENV
      READ
      IF QTIDENV > QTIDADE
         @ 23,30 SAY "ERRO PROVAVEL NA ENTRADA"
      ENDIF
   ENDIF -- para o caso de nao ser fornecido um numero
ENDDO
GO BOTTOM
DO WHILE mpven=NUMPVEN .AND. .NOT. ISBN=misbn .AND. .NOT. #=0
   SKIP -1
ENDDO
SKIP
COPY NEXT 65000 TO arqT FOR QTIDENV < QTIDADE
USE Pedpend
APPEND FROM arqT
REPLACE NUMPVEN WITH retempven FOR NUMPVEN=m.fatura
USE Faturado
DELETE FILE ARQT
RETURN
```

Programa 4. MENU6-1B.CMD/.PRG Programa para processar pedidos incompletos. Programa auxiliar do MENU6-1

VENDAS

PVEN #	NUM CLN	DATA	PCOM #
221212	123456	09/21/83	A282
221213	837211	09/21/83	129-83
221214	161171	09/21/83	1876-11

DETVEND

PVEN #	ISBN	QTID	PREÇO
221212	848374	7	643
221213	123456	10	895
221213	678435	5	714
221213	897343	30	934
221214			

PEDABR

PVEN#
221213

FATURAS

FATURA#	NUMCLN	DATA PCOM #	QUANTIA
121613	123456	09/21/81	24.95
121616	837211	09/22/83	86.21

FATURADO

PVEN #	ISBN	QTID	PREÇO	QTID ENV
121615	845763	11	9.45	6
121616	123456	10	8.95	5
121616	674939	5	7.14	5
121616	894737	30	6.00	30

PED PEND

PVEN #	ISBN	QTID	PREÇO	QTID ENV
221213	123456	10	8.95	5

Figura 4. Representação da passagem dos registros de pedidos de vendas para a fatura e novamente para os arquivos de pedidos pendentes

Estando os registros "editados", precisamos passar cópias deles para o arquivo de pedidos pendentes, PEDPEND. Retemos esses registros em FATURADO para podermos reimprimir a fatura a qualquer momento. A passagem deles novamente para o arquivo de pedidos pendentes é feita usando-se um arquivo temporário como meio de transferência. Isso permite substituir o número da fatura pelo número do pedido de venda, o que então vincula registros de pedidos pendentes ao pedido de venda original, permitindo-nos responder ao cliente sobre o andamento de seu pedido.

Uma vez de volta ao programa principal, armazenamos a variável de pedido de venda *mpven* em *retermpven*, armazenamos a variável *fatura* em *mpven* e armazenamos a data do sistema em *data*. Essas operações são realizadas de forma a podermos usar o mesmo programa, o Programa 6,

```
                    ---- FATURA ----

DISTRIBUIDORA FERREIRA                                  1 AGO 1985
Rua Italiana, 10150          Fatura Numero 200048
SAO PAULO - SP               data: 01/08/85

LIVRARIA ODISSEIA            NUMERO DO CLIENTE 1000003
Av.  Caetes,  43             REF. PEDIDO DE COMPRA 1245678
JAU - SP 97742
ATN: AGUIAR

LIVRO SOLICITADO/ISBN                  QTID      PRECO      TOTAL

BARDOLATRIA                              21      10.50     220.50
 0-201-14471-9

A PRINCESA E A ERVILHA                   22      11.00     242.00
 0-911979-00-X

O IDOLO DO PREGUICOSO                    23      11.50     264.50
 0-8359-9517-8

POESIA                                   24      12.00     288.00
 0-89588-047-4

DIREITOS PRIMITIVOS                      25      12.50     312.50
 0-13-110163

MANUAL DO BASIC                          26      13.00     338.00
 0-87835-025-X

VIAGEM PELO AVON                         27      13.50     364.50
 0-89588-048-2

A BOA RAINHA ANA                         28      14.00     392.00
 0-931988-84-5

GUIA DE COMPUTACAO PARA O COMPLETO       29      14.50     420.50
IDIOTA
 0-89588-042-3

PROGRAMA PARA NATACAO NADA OU AFUNDA     30      15.00     450.00
 0-87588-066-0

                                       Total               3292.50
```

Figura 5. Exemplo de fatura

para imprimir as faturas aqui e na seleção de Menu 3, Reimprimir uma Fatura. Discutiremos o programa de impressão na seção sobre Reimprimir uma Fatura. A Figura 5 mostra a fatura impressa pelo programa.

Após imprimir a fatura, acrescentamos um registro ao arquivo de fatura principal FATURAS. A operação de substituição utiliza uma variável quantia, que foi criada pelo programa de impressão.

Agora voltamos ao arquivo de banco de dados inicial, PEDABR (a lista de pedidos em aberto) e "fechamos" o pedido apagando o registro correspondente ao número de pedido de venda que acabamos de preencher. Podemos, então, preencher o pedido seguinte ou então retornar ao Menu de Preenchimento.

Reimpressão de Faturas

Há diversas razões possíveis para querermos reimprimir uma fatura, uma delas e não a menos importante, é quando a impressora "enlouquece" e amassa o papel durante a impressão original. É necessário que possamos reimprimir a maioria dos resultados em papel de nosso sistema de contabilidade.

Em nosso exemplo, podemos reimprimir a fatura depois de algum período de tempo. No Programa 5, o programa para reimprimir faturas, testamos o número de fatura fornecido para ver se está no banco de dados, pelo simples exame do primeiro e do último registros. A Figura 6 mostra um diagrama do Programa 5.

Em nosso sistema de contabilidade descartamos as faturas que foram faturadas, ao final de cada mês. Isso nos permite utilizar o comando LOCATE para localizar o seu registro. Esse comando sempre "encontrará" o registro pesquisando todo o banco de dados. Em um sistema interativo, com um grande banco de dados, essa operação poderá levar mais tempo do que o usuário gostaria.

Se o banco de dados fosse realmente pequeno, poderia dar resultado mais rápido usar simplesmente o comando LOCATE como o teste para verificar a existência das tarefas. Em nosso exemplo, supusemos que o teste de limite leva muito menos tempo que o comando LOCATE. Se não fosse assim, teria sido melhor usar LOCATE.

Mesmo que o número da fatura esteja no limite válido de números de fatura, ele poderá *não* estar no arquivo de banco de dados. Se não estiver, temos que realizar o teste para efetivamente encontrar a fatura.

Para podermos imprimir a fatura depois de encontrá-la, armazenamos todos os parâmetros que a descrevem em variáveis de memória. Precisaremos desses parâmetros em vários pontos dentro do programa de impressão, e será mais conveniente obtê-los da memória do que reabrir o arquivo de faturas.

A seguir, localizamos o registro do cliente e armazenamos todos os parâmetros do cliente em variáveis de memória. Novamente, utilizamos a memória para facilitar o trabalho posterior.

Um dos motivos de usarmos esses procedimentos é permitir que essa operação de impressão e a operação de impressão original da fatura, o Item 1 de Menu, usem o mesmo programa de impressão.

Para imprimir a fatura desejada precisaremos das variáveis de memória armazenadas e dos arquivos de bancos de dados FATURADO e INV (inventário). Esses dois arquivos contêm os detalhes sobre as transações.

Na primeira página da fatura (Figura 5), imprimimos um cabeçalho de página. Esse cabeçalho é impresso pela chamada ao Programa 7. Nas páginas subseqüentes, se houver, imprimimos o número de página da fatura e o número da fatura. Você deverá observar que a data impressa é a da fatura original e não a data atual.

```
* Programa -------- Menu6-3.CMD/ .PRG
* Obs. ------------ Programa para reimprimir faturas

ERASE
STORE T TO programa
USE Faturas
SELECT SECONDARY
USE Clientes INDEX Numcln
SELE PRIM
DO WHILE programa
   @ 1,30 SAY "REIMPRIMIR UMA FATURA"
   STORE "      " TO mpven
   @ 3,1 SAY "ENTRAR O NUMERO DA FATURA" GET mpven PICTURE "999999"
   READ
   IF mpven="      "
      STORE F TO programa
      LOOP
   ENDIF
   GO TOP
   STORE FATURA TO inicia
   GO BOTTOM
   STORE FATURA TO termina
   IF FATURA >= inicia .AND. FATURA <= termina
      LOCATE FOR FATURA =mpven
   ENDIF
   IF EOF .OR. FATURA > termina .OR. FATURA < inicia
      @ 5,1 SAY "FATURA " +mpven+" NAO ESTA DISPONIVEL PARA IMPRESSAO"
      LOOP
   ENDIF
   @ 5,0
   STORE NUMCLN TO numcln
   STORE NUMPED TO numped
   STORE DATA TO data
   STORE FATURA TO fatura
   SELECT SECONDARY
   FIND &mnumcln
   SELECT PRIMARY
   STORE NOME TO nome
   STORE ENDERECO TO endereco
   STORE TRIM(CIDADE) +", "+ESTADO+"   "+CEP TO cidade
   STORE ATN TO atn
   DO Menu6-3p
   ERASE
   SELECT SECONDARY
   USE Clientes INDEX Numcln
   SELECT PRIMARY
   USE Faturas
ENDDO ---------- para o loop do programa principal
RELEASE programa,mpven,inicia,termina,mnumcln,numped,data,fatura,nome;
      endereco,cidade,atn,quanti,pagina,encompra,linha,mrsbn
SELE SECO
USE
SELE PRIMARY
RETURN
```

Programa 5. MENU6-3.CMD/.PRG Parte principal do programa para imprimir faturas

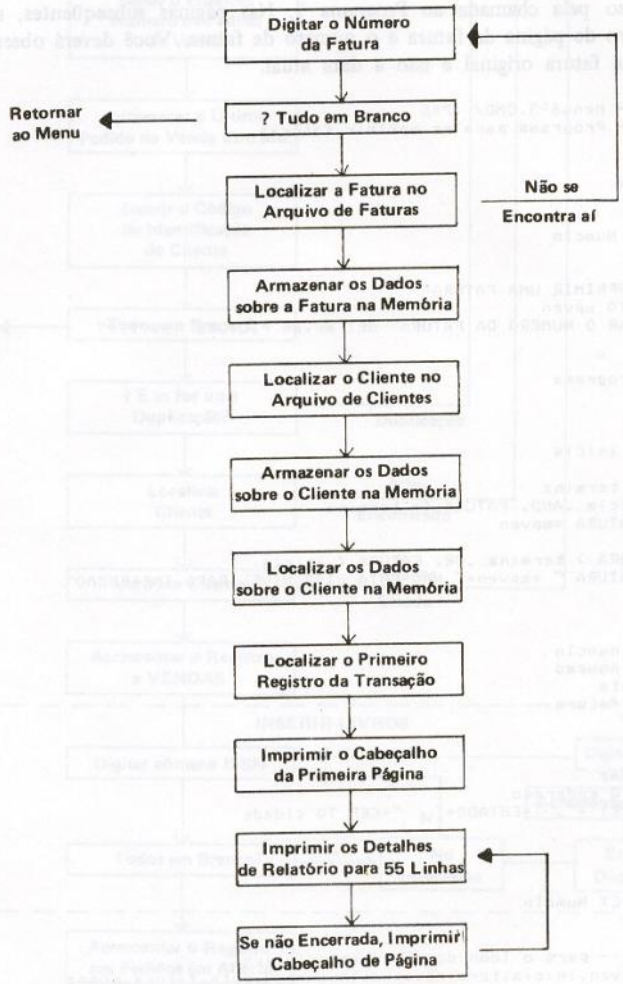


Figura 6. Diagrama do programa para reimprimir faturas


```
* Programa ------------ MENU6-3P.CMD/.PRG
* Obs. ---------------- programa para imprimir faturas

SET FORMAT TO PRINT
SET MARGIN TO 15
ERASE
STORE 0 TO quantia,pagina
SELECT SECONDARY
USE Inv INDEX Isbn
SELECT PRIMARY
USE Faturado
DO BUSCA
DO WHILE NUMPVEN=MPVEN .AND. .NOT. EOF
   STORE pagina+1 to pagina
   IF Pagina=1
      Do cabecalho
      @ 18,1 SAY "LIVRO SOLICITADO/ISBN                    QTIDADE    PRECO;
TOTAL "
      STORE 20 TO Linha
   ELSE
      @ 6,1 SAY "PAGINA "+STR(pagina,2)
      @ 7,1 SAY "FATURA # "+FATURA
      @ 18,1 SAY "LIVRO SOLICITADO/ISBN                    QTIDADE    PRECO:
TOTAL "
      STORE 20 TO linha
   ENDIF
   DO WHILE linha < 55 .AND. NUMPVEN=mpven .AND. .NOT. EOF
      STORE P.ISBN TO misbn
      SELECT SECONDARY
      FIND &misbn
      @ linha,1 SAY TITULO+" "+STR(QTIDENV,3)+" "+STR(PRECO,10,2)+STR;
                    (PRECO*QTIDENV,10,2)
      @ linha+1,5 SAY misbn
      IF QTIDENV < QTIDADE
         @ LINHA +1,42 SAY "P/P"
      ENDIF
      STORE linha+3 TO linha
      STORE PRECO*QTIDADE+QUANTIA TO QUANTIA
      IF SELECT="MENU6-1"
         REPLACE QTIDDISP WITH QTIDDISP-QTIDENV, VENDESTAN WITH;
                 VENDESTAN+QTIDENV, ULTVEN WITH DATE() NOUPDATE
      ENDIF
      SELECT PRIMARY
      SKIP
   ENDDO ---------- loop interior
   IF NUMPVEN # mpven .OR. EOF
      @ linha+1,47 SAY "Total Devido"
      @ linha+1,56 SAY STR(m.quantia,10,2)
   ENDIF
   EJECT
ENDDO ------- LOOP EXTERIOR
SET FORMAT TO SCREEN
SET MARGIN TO 0
RETURN
```

Programa 6. MENU6-3P.CMD/.PRG Rotina de impressão para o MENU6-1 e MENU6-3

```
* cabecalho ------------ imprime cabecalho para o Menu6-3P

@ 5,25 SAY "--- FATURA ---"
@ 6,1  SAY "DISTRIBUIDORA FERREIRA "
@ 6,55 SAY cdata
@ 7,1  SAY "RUA ITATIAIA 10150"
@ 7,45 SAY "Numero de fatura "+FATURA
@ 8,1  SAY "SAO PAULO - SP 90230"
@ 8,45 SAY "Data de: "+m.data
@ 13,1 SAY m.nome
@ 13,44 SAY "NUMERO DO CLIENTE "+mnumcln
@ 14,1 SAY m.endereco
IF .NOT. numped = "    "
   @ 14,38 SAY "REF: PEDIDO DE COMPRA "+numped
ENDIF
@ 15,1 SAY m.cidade
IF .NOT.ATN = "    "
   @ 16,1 SAY "ATN: "+atn
ENDIF
```

Programa 7. Programa para imprimir cabeçalhos de faturas

A parte mais volumosa da fatura é impressa pelo loop interior que também verifica quantas linhas foram impressas. A cada passagem pelo loop, processamos um registro do arquivo FATURADO.

As informações detalhadas impressas consistem em dados provenientes de dois bancos de dados. O título provém do banco de dados de inventário; a quantidade enviada, o preço e o número ISBN provêm do banco de dados de faturas. Observe que neste caso imprimimos os dados em duas linhas. Não há espaço de linha suficiente no tipo "pica" de dez caracteres por polegada para poder imprimir cada registro em uma única linha e ainda manter a margem.

Nós indicamos os clientes com pedidos pendentes pela impressão de P/P sob a quantidade enviada. Observe que não imprimimos a quantidade solicitada.

Se estivermos imprimindo a fatura proveniente da Seleção 1 de Menu, também vamos querer a essa altura atualizar os campos de inventário QTIDDISP, VENDESTAN e ULTVEN. REPLACE NOUPDATE evita que o arquivo de índice ISBN seja testado e economiza um pouco de tempo.

Quando saímos do loop interior, testamos se a operação foi completada, antes de emitir a página. Após imprimir registros de fatura, o valor total da fatura é impresso.

Capítulo Dez

FATURAMENTO DE CONTAS A RECEBER

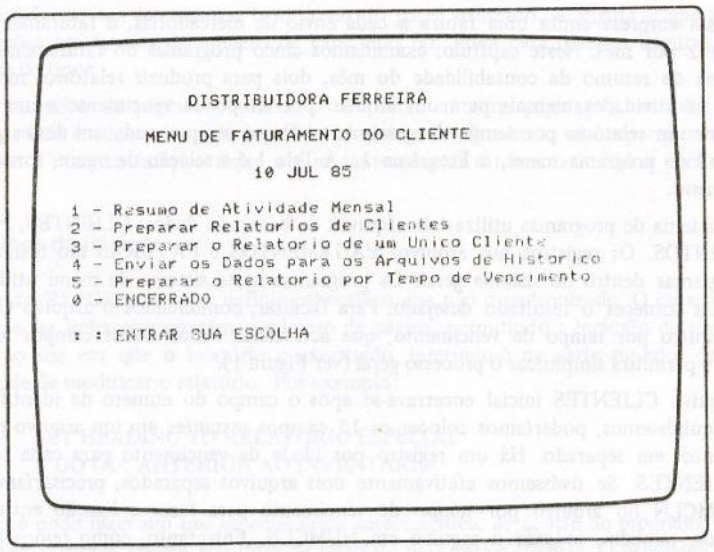
Embora nossa empresa emita uma fatura a cada envio de mercadorias, o faturamento final é feito uma vez por mês. Neste capítulo, examinamos cinco programas do faturamento mensal: um programa de resumo da contabilidade do mês, dois para produzir relatórios mensais, um para enviar as atividades mensais para um arquivo por tempo de vencimento e um programa para produzir um relatório por tempo de vencimento. Como sempre, cada um desses programas é "chamado" de programa menu, o Programa 1. A Tela 1 é a seleção de menu, fornecida pelo programa menu.

Esse sistema de programas utiliza três arquivos de banco de dados: CLIENTES, FATURAS e PAGAMENTOS. Os registros nos arquivos PAGAMENTOS e FATURAS são fornecidos por outros programas dentro do sistema geral. Os programas neste sistema de menu utilizam esses arquivos para fornecer o resultado desejado. Para facilitar, combinamos o arquivo CLIENTES com um arquivo por tempo de vencimento, que acrescenta vários novos campos ao arquivo CLIENTES e permitirá simplificar o processo geral (ver Figura 1).

O arquivo CLIENTES inicial encerrava-se após o campo do número de identificação do cliente. Se quiséssemos, poderíamos colocar os 15 campos restantes em um arquivo por tempo de vencimento, em separado. Há um registro por idade de vencimento para cada registro no arquivo CLIENTES. Se tivéssemos efetivamente dois arquivos separados, precisaríamos de um campo NUMCLN no arquivo por tempo de vencimento para fazer a ligação entre os dois; precisaríamos, também, indexar o arquivo em NUMCLN. Entretanto, como temos suficientes arquivos disponíveis em nosso arquivo CLIENTES, será mais conveniente combiná-los, como fizemos. Embora o arquivo de índice atual NUMCLN funcione com um arquivo por tempo de vencimento separado, não podemos utilizar o mesmo arquivo de índice tanto para o arquivo primário como para o secundário.

Os campos por tempo de vencimento, M1CHG até M6CHG, contêm os subtotais de faturas desde os últimos cinco meses. Os campos de M1PAG até M6PAG referem-se aos subtotais de pagamentos. HISTÓRIC combina pagamentos e faturas de todos os meses anteriores. A cada mês, deslocamos todos os dados referentes ao mês (M1CHG se tornará M2CHG etc.). DATADAFAT é a data do faturamento. SALDO é o total devido na ocasião do faturamento.

Os planos do banco de dados para o arquivo PAGAMENT e para o arquivo FATURAS são apresentados pela Figura 2. Há um bom motivo para se combinar os dois arquivos, já que o tempo necessário para executar os programas será reduzido significativamente (nos computadores do tipo PC, que utilizam o DOS 2.0, a abertura e fechamento de arquivos é um procedimento que particularmente consome tempo). Embora a combinação de arquivos economize tempo, não a utilizamos porque queremos demonstrar o uso de múltiplos arquivos.



Tela 1. Tela do menu de faturamento do cliente


```
* Programa ------:MENU7 .CMD (ou MENU7 .PRG)
* Obs. ----------:Este e o programa do Menu de Faturamento
USE Clientes INDEX Clientes
STORE " " TO selecao
DO WHITLE .NOT. selecao = "0"
   ERASE
   STORE " " TO selecao
   @ 3,25 SAY "DISTRIBUIDORA FERREIRA"
   @ 5,23 SAY "MENU DE FATURAMENTO DO CLIENTE"
   @ 7,30 SAY cdata
   @ 10,20 SAY "1 - Resumo de Atividade Mensal"
   @ 11,20 SAY "2 - Preparar os Relatorios de Clientes"
   @ 12,20 SAY "3 - Preparar o Relatorio de um Unico Cliente"
   @ 13,20 SAY "4 - Enviar dos dados para os Arquivos de Historico"
   @ 14,20 SAY "5 - Preparar o Relatorio por Tempo de Vencimento"
   @ 15,20 SAY "0 - ENCERRADO
   @ 20,24 SAY "ENTRAR SUA ESCOLHA"
   @ 20,19 GET selecao
   READ
   DO CASE
      CASE selecao = "1"
         Do Menu7-1
      CASE selecao = "2"
         Do Menu7-2
      CASE selecao = "3"
         Do Menu7-3
      CASE selecao = "4"
         Do Menu7-4
      CASE selecao = "5"
         Do Menu7-5
   ENDCASE
ENDDO
RETURN
```

Programa 1. MENU7.CMD/.PRG Programa do menu de faturamento

DESCRICAO	NOME DO CAMPO	TIPO	TAM	DEC
NOME DE CLIENTE	NOME	C	030	
A ATENCAO DE	ATN	C	030	
ENDERECO	ENDERECO	C	025	
CIDADE	CIDADE	C	020	
ESTADO	ESTADO	C	002	
CEP	CEP	C	005	
TELEFONE (999)-999-9999	TEL	C	014	
DATA EM QUE SE TORNOU CLIENTE	DATA	C	008	
NUMERO DE IDENTIFICACAO DO CLIENTE	NUMCLN	C	006	
SALDO ATUAL	SALDO	C	008	002
DATA DO ULTIMO FATURAMENTO	DATADAFAT	C	008	
FATURAS DIO ULTIMO FATURAMENTO	M1CHG	N	008	002
FATURAS DO MES ANTERIOR	M2CHG	N	008	002
FATURAS DE DOIS MESES ATRAS	M3CHG	N	008	002
FATURAS DE TRES MESES ATRAS	M4CHG	N	008	002
FATURAS DE QUATROS MESES ATRAS	M5CHG	N	008	002
FATURAS DE CINCO MESES ATRAS	M6CHG	N	008	002
PAGAMENTOS REF.ULTIMO FATURAMENTO	M1PAG	N	008	002
PAGAMENTOS REF.MES ANTERIOR	M2PAG	N	008	002
PAGAMENTOS RES.DOIS MESES ATRAS	M3PAG	N	008	002
PAGAMENTOS RE,TRES MESES ATRAS	M4PAG	N	008	002
PAGAMENTOS RE.QUATRO MESES ATRAS	M5PAG	N	008	002
PAGAMENTOS REF.CINCO MESES ATRAS	M6PAG	N	008	002
RESUMO DA CONTA ANTERIOR	HISTORIC	N	008	002

TAMANHO DO REGISTRO 261 BYTES

INDICES NOME ----------CLIENTES.NDX
NUMCLN---------NUMCLN.NDX
CEP+NOME-------CEP.NDX

Figura 1. Plano do banco de dados para CLIENTES DBF (combinando o arquivo de clientes com o de tempo)

DESCRICAO	NOME DO CAMPO	TIPO	TAM	DEC
NUMERO IDENTIFICACAO CLIENTE	NUMCLN	C	006	
DATA DO PAGAMENTO	DATA	C	008	
TOTAL DO PAGAMENTO	QUANTIA	N	008	002

TAMANHO DO REGISTRO 23 BYTES

INDICES NUMCLN----------P-NUMCLN.NDX

Figura 2A. Plano do banco de dados para o arquivo de banco de dados PAGAMENT

DESCRICAO	NOME DO CAMPO	TIPO	TAM	DEC
NUMERO IDENTIFICACAO CLIENTE	NUMCLN	C	006	
NUMERO DA FATURA	FATURA	C	006	
DATA DO PAGAMENTO	DATA	C	008	
TOTAL DO PAGAMENTO	QUANTIA	N	008	002

TAMANHO DO REGISTRO 29 BYTES

INDICES NUMCLN---------I-NUMCLN.NDX

Figura 2B. Plano do banco de dados para o arquivo de banco de dados FATURAS

Item 1 de Menu Resumo da Atividade Mensal (MENU7-1.CMD/.PRG)

Para fazer o faturamento, vamos precisar de um relatório que resuma a atividade financeira de cada cliente. Esse relatório, fornecido pelo Programa 2, é apresentado pela Figura 3. A Figura 4 mostra um diagrama lógico do Programa 2. Esse programa, em especial, exemplifica o uso do comando SUM bem como o uso de arquivos múltiplos para imprimir uma linha de texto.

Para obter esse registro, usamos os três arquivos em cada linha do relatório. O arquivo primário, CLIENTES, fornece o nome do cliente e o saldo atual. O arquivo secundário, PAGAMENT, fornece os pagamentos recebidos durante o mês; o arquivo secundário FATURAS, fornece os débitos do mês ao cliente.

O programa utiliza dois loops: o loop exterior passa por todo o banco de dados de clientes, um registro por vez; o loop interior lida com o fato de haver mais clientes do que será possível colocar em uma única página. Em cada página, precisamos aumentar o número de página e imprimir o cabeçalho. Tendo terminado a página, precisamos passar a impressora para a seguinte.

Ao inserirmos o loop interior, selecionamos o arquivo primário, CLIENTES. O passo seguinte será selecionar o arquivo secundário. A seguir, armazenamos o número de identificação do cliente proveniente do arquivo primário, na variável de memória *mnumcln*. Observe que pudemos fazer isso mesmo já tendo escolhido o arquivo secundário. Isso porque fizemos NUMCLN ser precedido por P. (de primário). O arquivo secundário também possui um campo NUMCLN. Se não tivéssemos precedido NUMCLN por um P., teríamos problemas, já que iríamos armazenar na variável o número de identificação do cliente proveniente do arquivo secundário (se tivéssemos feito o armazenamento antes de inserir o arquivo secundário, poderíamos ter omitido o P. do comando). É um bom procedimento usar sempre o designador de arquivo P. ou S., quando há um campo comum tanto ao arquivo primário como ao secundário.

Para obter o valor do pagamento e o total das faturas de cada cliente, usamos o comando SUM. Qualquer comando que puder ser usado com a sentença FOR também o poderá com a sentença WHILE.

SUM TOTAL DO *pagcln* WHILE
- S.NUMCLN = *mnumcln*

equivale a

STORE 0 TO *pagcln*
DO WHILE S.NUMCLN = *mnumcln*
- .AND..NOT. EOF

STORE *pagcln* + TOTAL TO *pagcln*
SKIP
ENDDO

A escolha da melhor alternativa dependerá do computador que estiver sendo usado, do número de registro para cada cliente e de o usuário estar ou não fazendo mais que somar quantias em cada registro. Em geral, se houver poucos registros, a segunda abordagem será um pouco mais rápida. Isso porque o comando SUM está no arquivo overlay do dBASE II. Se precisarmos reposicionar a cabeça do disco para ler o comando para a memória, vamos perder tempo. Entretanto, se houver muitos registros para cada cliente, o comando SUM será mais rápido.

```
Pagina 1                                        01  AGO  1983

                        DISTRIBUIDORA FERREIRA

                    RESUMO DA ATIVIDADE MENSAL

NOME DO CLIENTE          SALDO      PAGAMENTOS      FATURAMENTO     NOVO
                         INICIAL    RECEBIMENTOS    ATUAL           SALDO
-------------------------------------------------------------------------

LIVRARIA ATENAS          125.00        275.00          619.34        469.34

LIVRARIA ODISSEIA        100.00        100.00          234.56        234.56

VOLUMES TECNICOS              0                                          0

TOTAIS                   225.00        375.00          853.90        703.90
```

Figura 3. Relatório da atividade mensal


```
* Programa -----------:MENU7-1.CMD (ou MENU7-1.PRG)

* Obs.----------------:Este programa prepara o Resumo da atividade Mensal.

STORE 0 TO saltotal,pagtotal,dbttotal,pagina
SET FORMAT TO PRINT
DO WHILE .NOT.EOF
   STORE pagina+1 TO pagina
   @ 6,15 SAY "Pagina" + STR (pagina,1)
   @ 6,75 SAY cdata
   @ 8,41 SAY "DISTRIBUIDORA FERREIRA"
   @ 10,41 SAY "RESUMO DA ATIVIDADE MENSAL"
   @ 12,15 SAY  "                                SALDO    PAGAMENTOS   FATURAMENTO:
   NOVO"
   @ 13,15 SAY  "NOME DO CLIENTE                 ATUAL    RECEBIDOS    ATUAL ;
   SALDO"
   @ 14,15 SAY  "-------------------------------------------------------------------;
   ---------------------"
   STORE 15 TO linha
   DO WHILE linha < 56 .AND. .NOT. EOF
      SELE SECO
      STORE P. NUMCLN TO mnumcln
      USE Pagament INDEX P-numcln
      FIND &mnumcln
      SUM QUANTIA TO pagcln WHILE S.NUMCLN = mnumcln
      USE Faturas INDEX I-numcln
      FIND &mnumcln
      SUM QUANTIA TO dbtscln WHILE S.NUMCLN = mnumcln
      @ linha,15 SAY NOME
      @ linha,45 SAY STR(SALDO,8,2)
      IF .NOT. pagcln = 0
         @ linha,57 SAY STR(pagcln,8,2)
      ENDIF
      IF .NOT. dbtscln = 0
         @ linha ,68 SAY STR (dbtscln,8,2)
      ENDIF
      @ linha,78 SAY STR(SALDO-pagcln+dbtscln,8,2)
      STORE linha+1 TO linha
      STORE saltotal+SALDO TO saltotal
      STORE pagtotal+pagcln TO pagtotal
      STORE dbttotal+dbtscln TO DBTTOTAL
      SELE PRIM
      SKIP
   ENDDO
   IF EOF
      STORE linha+2 To linha
      @ linha,15 SAY "TOTAIS"
      @ linha,44 SAY STR (saltotal,9,2)
      @ linha,56 SAY STR (pagtotal,9,2)
      @ linha,67 SAY STR (dbttotal,9,2)
      @ linha,77 SAY STR (saltotal-pagtotal+dbttotal,9,2)
   ENDIF
   EJECT
   ENDDO
   RELE saltotal,pagtotal,pagcln,dbtscln,pagina
   SET FORMAT TO SCREEN
   SELE SECO
   USE
   SELE PRIM
   RETURN
```

Programa 2. MENU7-1.CMD/.PRG Programa de resumo da atividade mensal

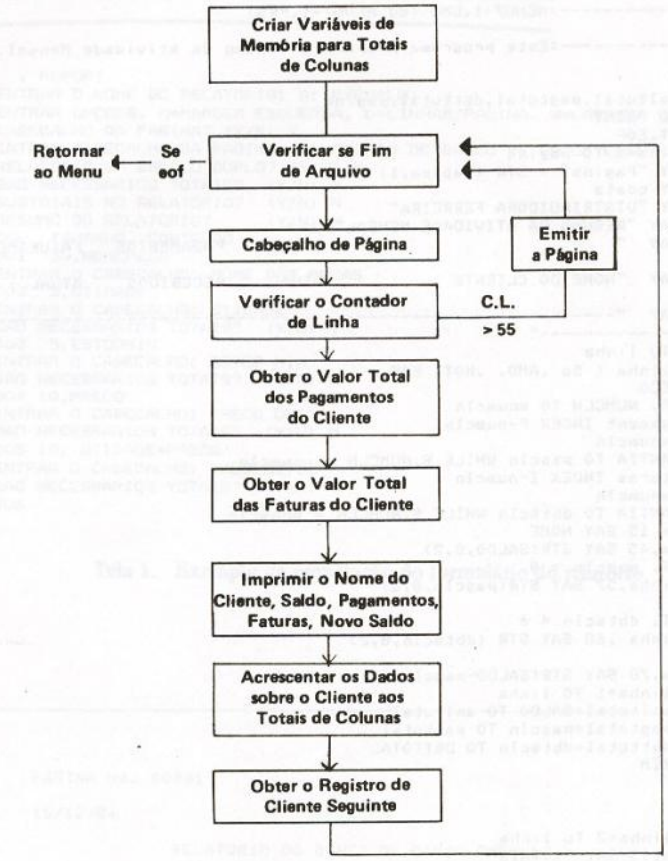


Figura 4. Diagrama do Programa de Resumo de Atividades

Observe que em nenhum desses casos precisamos nos preocupar em buscar o número de identificação do cliente no arquivo secundário. Se não encontramos o número de identificação, o banco de dados secundário fica posicionado no Registro 0. O conteúdo do campo NUMCLN não pode corresponder a *mnumcln*. Igualmente, ao usar SUM WHILE não precisamos nos preocupar quanto ao final do arquivo.

Tendo acumulado os pagamentos e os subtotais das faturas, imprimimos a linha. Neste exemplo, optamos por não imprimir os pagamentos dos clientes nem suas faturas, se seu valor for zero. A não-impressão de saldos em zero faz com que eles sejam ressaltados e fiquem mais

fáceis de se visualizar do que se os imprimíssemos. Observe que, quando imprimimos o novo saldo, usamos um campo e duas variáveis de memória no interior dessa instrução para imprimir.

```
@ linha,78 say str(SALDO–pagcln+
■ dbtscln,8,2)
```

A seguir, acrescentamos cada um desses itens à coluna apropriada de totais. Há quatro colunas. Para imprimir a quarta coluna de total basta apenas conhecer os outros três totais.

Agora retornamos ao arquivo primário e avançamos um único registro.

Este programa não modifica o conteúdo de nenhum dos bancos de dados nem afeta os índices. Ele pode ser executado tantas vezes quanto necessário durante o mês.

Item 2 de Menu Relatório Mensal do Cliente (MENU7-2.CMD/.PRG)

Para o faturamento mensal do cliente queremos um relatório em duas páginas. A primeira é um resumo simples das atividades mensais do cliente. Essas faturas são impressas "em tríades" – cada linha do relatório com três faturas. Uma amostra de relatório obtida com o programa para faturamento mensal está nas Figuras 5 e 6.

O Programa 3 é basicamente o mesmo que o do resumo da atividade do mês, exceto que estamos usando muito mais formatação de impressão. Fora isso, a estrutura lógica é muito semelhante. Novamente imprimiremos os relatórios dos clientes em ordem alfabética, o que facilita na correspondência das instruções com as etiquetas impressas pelo programa de impressão de etiquetas. Se dobrarmos os relatórios corretamente, poderemos usar envelopes com janelas de papel transparente em vez de etiquetas. Embora custem um pouco mais que os envelopes comuns, esses envelopes têm uma aparência mais profissional e permitem poupar tempo no preparo do faturamento mensal.

Grande parte do Programa 3 versa sobre a impressão de cabeçalhos de páginas. Se tivéssemos usado um cabeçalho de página padronizado nos outros programas, como no de pedidos de livros, poderíamos economizar 20 linhas somente neste programa. Este é um dos objetivos da programação estruturada. Com um pequeno programa para impressão de cabeçalhos, podemos substituir o código de cabeçalho de página por DO CABEÇALHO. Um número menor de linha de código por programa aumenta a clareza, tornando-o mais fácil de ser entendido.

Quando "chamamos" um outro programa como um programa cabeçalho, a abertura do arquivo de comando leva uma fração de segundo. Neste exemplo específico, o tempo limitado necessário para abrir um arquivo seria de qualquer forma perdido com a impressão e a manipulação do papel. Quando fazemos um grande número de chamadas ao arquivo comando, o tempo de abertura poderá afetar significativamente a velocidade geral de execução. Tudo isso significa que devemos contrabalançar com um pouco de bom senso o nosso esforço por diminuir o tamanho do programa.

O posicionamento da impressão de cada fatura na Página 2 do relatório mensal do cliente é realizada por DO CASE. A cada passagem pelo loop, avaliamos o valor da última coluna e adaptamos o conteúdo das variáveis *linha* e *col*, adequadamente.

```
* Programa --------: MENU7-2 .CMD ( OU MENU7-2.PRG)
* Obs:-------------: Este programa executa Relatorios Mensais
SET FORMAT TO PRINT
GO TOP
DO WHILE .NOT. EOF
   STORE NUMCLN TO mnumcln
   @ 6,15 SAY "DISTRIBUIDORA FERREIRA"
   @ 6,70 SAY cdata
   @ 7,15 SAY "Rua Itatiaia 10150"
   @ 8,15 SAY "SAO PAULO, SP 90230"
   @ 13,15 SAY NOME
   @ 13,50 SAY "NUMERO DO CLIENTE "+ NUMCLN
   @ 14,15 SAY ENDERECO
   @ 15,15 SAY TRIM(CIDADE)+", "+ESTADO+"   "+CEP
   IF .NOT. ATN = "  "
      @ 16,15 SAY "ATN: "+ATN
   ENDIF
   @ 18,45 SAY "RELATORIO"
   @ 19,15 SAY "-------------------------------------------------------------------
   ----------"
   @ 21,15 SAY "SALDO DEVEDOR DESDE: "+DATADAFAT
   @ 21,50 SAY  STR(SALDO,8,2)
   STORE NUMCLN TO mnumcln
   SELE SECO
   USE Faturas INDEX I-numcln
   FIND &mnumcln
   SUM QUANTIA TO dbtscln WHILE NUMCLN=mnumcln
   @ 22,15 SAY "DEBITA DESDE "+DATADAFAT+" ATE " + DATE()
   @ 22,50 SAY STR(dbtscln,8,2)
   USE Pagament INDEX P-numcln
   @ 23,15 SAY "PAGAMENTOS RECEBIDOS"
   FIND &mnumcln
   STORE 23 TO linha
   STORE 0 TO pagcln
   DO WHILE NUMCLN=mnumcln .AND. .NOT. EOF
      @ linha,50 SAY STR (QUANTIA,8,2)+"   "+DATA
      STORE QUANTIA+pagcln TO pagcln
      SKIP
      STORE linha+1 TO linha
   ENDDO
   @ linha+2,15 SAY "SALDO DEVEDOR ATUAL:"
   @ linha+2,50 SAY STR(SALDO-pagcln+dbtscln,8,2)+"   CONDICOES: 10 DIAS
                                                      DA DATA
   SELE PRIM
   EJECT
   @ 6,15 SAY "DISTRIBUIDORA FERREIRA"
   @ 6,70 SAY cdata
   @ 7,15 SAY "Rua Itatiaia 10150"
   @ 8,15  SAY SAO PAULO, SP 90230
   @ 13,15 SAY NOME
   @ 13,50 SAY "NUMERO DO CLIENTE "+NUMCLN
   @ 14,15 SAY ENDERECO
   @ 15,15 SAY TRIM(CIDADE)+","+ESTADO+"     "+CEP
   IF .NOT .ATN=" "
      @ 16,15 SAY "ATN: "+ATN
   ENDIF
   @ 18,45 SAY "DETALHES DO RELATORIO"
   @ 19,15 SAY "-------------------------------------------------------------------
----------"
```

Programa 3. MENU7-2.CMD/.PRG Programa para relatórios mensais


```
   @ 20,15 SAY "FATURA                              FATURA
FATURA"
   @ 21,15 SAY "NO.       DATA       DEBITAR    NO.       DATA       DEBITAR
NO.      DATA          DEBITAR"
   @ 22,15 SAY "-------------------------------------------------------------------
----------"
   SELE SECO
   USE Faturas INDEX I-numcln
   FIND &mnumcln
   STORE 24 TO linha
   STORE 15 TO col
   DO WHILE NUMCLN=mnumcln .AND. .NOT. EOF
      @ linha,col SAY FATURA NO
      @ linha,col+7 SAY DATA
      @ linha,col+16 SAY STR (QUANTIA,7,2)
      DO CASE
         CASE col=15
              STORE 42 TO col
         CASE col=42
              STORE 69 TO col
         CASE col=69
              STORE L5 TO col
              STORE linha+1 TO linha
      ENDCASE
      SKIP
   ENDDO
   EJECT
   SELE PRIM
   SKIP
ENDDO
SET FORMAT TO SCREEN
SELE SECO
USE
SELE PRIM
RELEASE mnumcln,linha,col,dbtscln,pagcln
RETURN
```

```
DISTRIBUIDORA FERREIRA                          01  AGO 1983
Rua ITATIAIA. 10150
SAO PAULO, SP 90230


LIVRARIA ATENAS                    CLIENTE NUMERO  100002
Rua Monte Verde. 2731
SAO PAULO, SP 21223
ATN: SERGIO LEMOS

                                RELATORIO
-------------------------------------------------------------------------------

SALDO DEVEDOR DESDE: 30/06/85                    125.000
DEBITAR DESDE 30/06/85  ATE  08/01/85            619.340
PAGAMENTOS RECEBIDOS                             125.000  05/07/85
                                                 150.000  15/07/83


SALDO DEVEDOR ATUAL:                             469.340  CONDICOES: 10 DIAS
                                                          DA DATA
```

Figura 5. Relatório mensal do cliente, página 1

```
DISTRIBUIDORA FERREIRA                          01  AGO  1983
R.Itatiaia  10150
SAO PAULO,SP 90230

LIVRARIA ATENAS                        CLIENTE NUMERO 10002
R.Monte VERDE,2731
SAO PAULO, SP 21223
ATN: SERGIO LEMOS

                          DETALHES DO RELATORIO
-------------------------------------------------------------------------------
FATURA                    FATURA                    FATURA
NO.      DATA    DEBITAR  NO.      DATA    DEBITAR  NO.      DATA    DEBITAR
-------------------------------------------------------------------------------

892345   15/7/85 153.450   456283   31/07/85 485.090
```

Figura 6. Relatório mensal do cliente, página 2

```
* Programa ---------------:MENU7-3 .CMD (ou menu7-3.PRG)
* OBS.:-------------------:Este programa imprime o relatorio para um Unico
* ----------------------- Cliente.

STORE T TO imprimir
ERASE
DO WHILE imprimir
   STORE "                                          " TO clientes
   @ 5,10 SAY "IMPRIMIR RELATORIOS INDIVIDUAIS DE CLIENTES"
   @ 7,10 SAY "NOME DO CLIENTE" GET clientes
   READ NOUPDATE
   FIND &clientes
   IF clientes = "    "
      STORE F TO imprimir
      LOOP
   ENDIF
   IF  #=0
      @ 9,1 SAY "NAO FOI ENCONTRADO O CLIENTE COMO FORNECIDO"
      LOOP
   ENDIF
   @ 9,0
   COPY NEXT 1 TO ARQTEMP
   USE ARQTEMP
   DO Menu7-2
   USE Clientes INDEX Clientes
   CLEAR GETS
ENDDO
DELETE FILE ARQTEMP
RELEASE clientes,imprimir
RETURN
```

Programa 4. MENU7-3.CMD/.PRG Programa para relatórios de clientes escolhidos

Este programa não afeta o conteúdo de nenhum dos bancos de dados e pode ser executado sempre que desejado. Em geral, não é aconselhável um conteúdo de banco de dados afetado pelo programa de impressão porque se houver problemas com a impressora durante a execução do programa, não poderemos ajustá-la e recomeçar. Sempre que possível, deve-se tentar separar a impressão da mudança do banco de dados.

Item 3 de Menu Imprimir Relatórios Mensais sobre os Clientes Escolhidos (MENU7-3.CMD/.PRG)

O Programa 3 é um programa de produção: podemos ligá-lo e nos afastarmos enquanto ele imprime o faturamento mensal. Mas o que poderá ser feito se acontecer algo a um dos relatórios durante a impressão?.

Neste caso, usamos o Programa 4, um programa para imprimir os relatórios mensais dos clientes escolhidos.

Neste exemplo, é possível a escolha por meio do comando de cópia. Como cada um dos programas nesse sistema de menu supõe que o arquivo CLIENTES (indexado pelo nome dos clientes) está sendo usado quando o programa é chamado, podemos trapacear um pouquinho.

Podemos ir ao registro desejado dentro do arquivo CLIENTES e copiar o registro que queremos no arquivo ARQTEMP. Então chamamos o Programa 3, MENU7-2. Embora o MENU7-2 seja usado para imprimir relatórios para todos os registros no arquivo CLIENTES, ele não tem condições de saber que não se está usando CLIENTES. Assim, ele imprime relatórios para todos os registros no arquivo ARQTEMP (um registro).

Item 4 de Menu Envia as Atividades para o Arquivo por Tempo de Vencimento – Clientes (MENU7-4.CMD/.PRG)

Esse é o programa mais interessante neste sistema de menu. O Programa 5 armazena os resumos dos pagamentos e faturas para o arquivo CLIENTES. Os arquivos PAGAMENT e FATURAS são então esvaziados para iniciar as atividades do mês seguinte.

Uma vez estando o programa executado até o fim, o faturamento mensal o processa por inteiro. Não poderemos fazer o faturamento se acidentalmente executarmos esse programa em primeiro lugar. Ao mesmo tempo, não queremos enviar automaticamente quando fizermos o faturamento. Para que a operação seja bem-sucedida, vamos depender de uma combinação de bom senso por parte do operador e uma pequena ajuda do programa. Quando o faturamento for encerrado, o operador deverá executar esse programa antes que novos itens sejam fornecidos, quer ao arquivo PAGAMENT quer ao arquivo FATURAS. Podemos oferecer proteção razoável contra a sobreposição de faturamentos, verificando a data do último faturamento no arquivo do cliente e usando um código nos programas para acrescentar registros de pagamento e faturas. Para evitar o envio prematuro, possibilitamos ao operador digitar um código para acionar a execução do programa.

Durante a entrada do programa, o usuário é informado de que este está sendo usado. Ele deve deliberadamente agir de modo a executá-los (isso evita que o programa seja executado se uma única tecla errada for pressionada no menu). Optamos pela entrada da seqüência ABCDEF para o programa poder rodar. Essa seqüência é simples o suficiente para não ser desagradável e suficientemente longa para evitar uma ação errada espontânea.

A primeira coisa que queremos fazer é mudar os dados em cada campo mensal, passando-os cada um para o campo anterior em um mês. O débito mais antigo é somado e o crédito mais antigo é subtraído do campo do histórico. Em termos rigorosos, não precisamos realmente do saldo e da data dos campos de faturamento. O primeiro pode ser calculado com os dados nos campos mensais e do histórico. O último pode ser manipulado como uma variável permanentemente retida no arquivo DATA.MEM. Mas, como temos somente perto de duas centenas de clientes em nosso arquivo, a perda relativa ao transporte desses itens é insignificante. Se estivéssemos lidando com milhares de clientes, iríamos querer calcular cuidadosamente como armazenar essas informações.

Utilizamos REPLACE ALL <exp> para fazer a mudança dos campos. A ordem usada para isso – dos dados mais antigos para os mais novos – é importante. Observe que desligamos o *índice* antes de realizar as substituições, o que significa que estamos trabalhando com a ordenação física dos registros. Com o índice ligado, as substituições ocorreriam na ordem lógica, controladas pelo arquivo de índice. Exceto em um caso patológico, o trabalho com a ordenação lógica exigiria muito mais movimento da cabeça do disco. Mesmo com duas centenas de registros, o tempo necessário seria significativo. Vários minutos é um longo tempo, se compararmos a atividade com o índice ligado com a atividade quando ele está desligado.

Aqui, REPLACE ALL <exp> é um comando especificamente longo: quatro linhas separadas por ponto e vírgula (como você deve se lembrar, podemos ter até 254 caracteres em um comando e, no final de uma linha, marcamos a continuação do comando com sinais de ponto e vírgula).

Observe que os campos para o mês atual estão em zero. Isso prevê os casos de clientes que não tiveram qualquer atividade no mês em curso.

O passo seguinte será usar o comando TOTAL para criar um arquivo temporário contendo os subtotais para o número de cada cliente. Como o arquivo secundário, FATURAS, é indexado de acordo com o número do cliente, o arquivo temporário FATTOT possui um registro para número de cliente no arquivo FATURAS. A estrutura de FATTOT é a mesma que a de FATURAS. O conteúdo do campo QUANTIA contém o subtotal de QUANTIA para cada número de cliente. Mais uma vez, se o arquivo for grande, esse processo poderá ser lento.

Uma palavra de advertência: quando usamos o comando TOTAL dessa forma, os campos que são totalizados devem ser suficientemente grandes para conter a soma. O dBASE II não modifica o tamanho do campo se houver excesso. Infelizmente, até quando esta obra foi publicada não havia ainda uma indicação de erro.

Não podemos usar UPDATE diretamente. Esse comando nos permite acréscimos ao conteúdo de um campo somente quando os nomes dos campos são idênticos.

Uma vez que tenhamos o arquivo de subtotais FATTOT, vamos querer eliminar os conteúdos atuais do arquivo de faturas. O método direto será:


```
DELETE ALL
PACK
```

Aqui, novamente, o uso de um arquivo de índice funciona contra nós. Um arquivo com 1000 registros (em um disco flexível) pode levar cerca de meia hora para executar a exclusão e a compactação. Desligar o índice pode reduzir o tempo em cerca de três minutos. A técnica que estamos usando reduz o tempo em alguns segundos.

```
* Programa --------:MENU7-4.CMD (ou MENU7-4 .PRG)
* Obs.-------------:Este programa envia relatorios de contabilidade mensais
* -----------------:de volta ao arquivo de clientes como um resumo.
ERASE
TEXT

     ***** ADVERTENCIA ******** ADVERTENCIA******** ADVERTENCIA*****

        VOCE ENTROU O PROGRAMA PARA ENVIAR PAGAMENTOS E FATURAS

     OS PAGAMENTOS E AS FATURAS DIARIOS SERAO ELIMINADOS APOS O ENVIO

ENDTEXT

? "   DATA DE HOJE: "+DATE()+"    ULTIMO FATURAMENTO:"+DATADAFAT
STORE "       " TO ativa
@ 11,10 SAY "ENTRAR ABCDEF e ENVIAR,qualquer outra coisa p/ ABORTAR" GET ativa
READ NOUPDATE
IF ativa="ABCDEF"
   SET INDEX TO
   REPL ALL HISTORIC WITH HISTORIC+M6CHG-M6PAG WITH M5PAG,;
        M5PAG WITH M4PAG,M4PAG WITH M3PAG,M3PAG WITH M2PAG,M2PAG WITH M1PAG
   REPL ALL M6CHG WITH M5CHG,M5CHG WITH M4CHG,M4CHG WITH M3CHG;
        M3CHG WITH M2CHG,M2CHG M1CHG,M1CHG WITH 0
   SELE SECO
   USE Faturas INDEX I-numcln
   TOTAL ON NUMCLN TO FATTOT
   COPY STRU TO Novafat
   USE
   DELETE FILE Faturas
   RENAME Novafat TO Faturas
   USE Faturas
   INDEX ON NUMCLN TO I-numcln
   SELE PRIM
   SET INDEX TO Numcln
   NOUPDATE FROM FATTOT ON NUMCLN RANDOM REPLACE M1CHG WITH QUANTIA
   SELE SECO
   USE Pagament INDEX P-numcln
   TOTAL ON NUMCLN TO Fattot
   COPY STRU TO Novopag
   USE
   DELETE FILE Pagament
   RENAME Novopag TO Pagament
   USE Pagament
   INDEX ON NUMCLN TO P-numcln.
   SELE PRIM
   UPDATE FROM Fattot ON NUMCLV RANDOM REPLACE M1PAG WITH QUANTIA
   SET INDEX TO
   REPLACE ALL SALDO WITH SALDO-M1PAG+M1CHG,DATADAFAT WITH DATE()
   SET INDEX TO Clientes
   DELE FILE FATTOT
ENDIF
RETURN
```

Programa 5. MENU7-4.CMD/.PRG Programa para enviar pagamentos e faturas para o arquivo de tempo dos clientes

Podemos nos livrar de tudo, exceto da estrutura; assim, copiamos a estrutura em um arquivo temporário. Então apagamos o arquivo de faturas. Observe que o comando USE é usado sozinho. Não podemos excluir o arquivo que estamos usando e não podemos renomeá-lo. Assim, voltamos a um ponto em que não estamos usando nenhum dos arquivos, excluímos FATURAS e renomeamos NOVAFAT para FATURAS. Observe que não precisamos regularizar o arquivo de índice I-NUMCLN, que ainda acredita que há 1000 registros. A solução aqui será usar o comando INDEX. A indexação de um arquivo sem registros é bastante rápida.

Agora voltamos ao nosso arquivo primário e escolhemos o índice para o número de cliente NUMCLN. Isso nos permite usar o comando UPDATE para atualizar o arquivo de clientes com os dados armazenados em FATTOT.

Nós repetimos o processo para o arquivo PAGAMENT, desligamos o índice e fazemos nossa última substituição.

Os comandos UPDATE e TOTAL são operações relacionais. Ambos são mais rápidos que se tentar escrever um código para se obter esse mesmo resultado. Eles não podem ser usados em todos os casos, mas são muito eficientes, quando aplicáveis.

Item 5 de Menu O Relatório por Tempo de Vencimento (MENU7-5.CMD/.PRG)

O relatório por tempo de vencimento apresenta os subtotais dos pagamentos e faturas por um período de tempo, de modo que podemos determinar quem deixou de fazer os pagamentos. Em nossa empresa, desde que o cliente faça um pagamento no mês, estará tudo acertado.

O programa para fornecer o relatório (Programa 6) é muito simples. Cada registro de cliente tem duas linhas: as faturas estão na linha de cima e os pagamentos na linha de baixo. O relatório preparado por esse comando é apresentado pela Figura 7.

NOME DO CLIENTE	SALDO	ULT.MES	30-60	60-90	90-120
LIVRARIA ATENAS	125.00	0.00	275.00	200.00	210.00
	01/08/83	275.00	200.00	210.00	220.00
LIVRARIA ODISSEIA	100.00	100.00	100.00	0.00	0.00
	01/08/83	100.00	0.00	0.00	0.00
VOLUMES TECNICOS	0.00	100.00	110.00	120.00	130.00
	01/08/83	110.00	120.00	130.00	140.00

Figura 7. Relatório por tempo de vencimento


```
* Programa----------:MENU7-5.CMD 9(ou MENU7-5.PRG)
* Obs---------------:Este programa imprime o Relatorio Tempo

GO TOP
SET INDEX TO c
SET MARGIN TO 15
SET FORMAT TO PRINT
DO WHILE .NOT. EOF
   @ 6,32 SAY "RELATORIOS POR DATA DE VENCIMENTOS PARA "+cdata
   @ 8,1 SAY "NOME DO CLIENTE                          SALDO     ULT.MES  30-60;
     60-90    90-120"
   STORE 10 TO linha
   DO WHILE linha < 55 .AND. .NOT. EOF
      @ linha, 1  SAY NOME
      @ linha,31  SAY SALDO
      @ linha,40  SAY M1CHG
      @ linha,50  SAY M2CHG
      @ linha,60  SAY M3CHG
      @ linha,70  SAY M4CHG
      @ linha+1,31  SAY DATADAFAT
      @ linha+1,40  SAY M1PAG
      @ linha+1,50  SAY M2PAG
      @ linha+1,60  SAY M3PAG
      @ linha+1,70  SAY M4PAG
      STORE linha+3 to LINHA
      SKIP
   ENDDO
   EJECT
   SET FORMAT TO SCREEN
   SET MARGIN TO 0
   RETURN
```

Programa 6. MENU7-5.CMD/.PRG Imprime o relatório por data de vencimento

Parte III

TÓPICOS ESPECIAIS

Capítulo Onze

COMO UTILIZAR O COMANDO REPORT

O comando REPORT merece um capítulo especial. A dificuldade que muitos de nós temos em utilizá-lo em toda sua capacidade é provavelmente a razão por que aprendemos a programar com o dBASE II antes de mais nada. O comando REPORT é extremamente potente, muito mais do que parece a princípio.

Modificação de Relatórios

Quando criamos um relatório, o dBASE II nos faz uma série de perguntas simples. Nossas respostas a essas perguntas são armazenadas como um arquivo de relatório (.FRM) em disco, de modo que quando fizermos solicitações ao relatório não teremos de passar pelas perguntas. Esse formulário de relatório pode ser modificado usando-se MODIFY COMMAND, ou a maioria dos processadores de texto, desde que se escolha o modo de processamento de texto adequado – por exemplo, a opção N, no WordStar.

No Capítulo 1, Tela 13, demonstramos o processo de criação de um formulário de relatórios, que foi aqui reproduzido na Figura 1, para facilitar a consulta. O relatório não está mal feito, mas gostaríamos de melhorar sua estética fazendo MIN avançar em um espaço, deslocando as entradas de coluna de ESTOQ MIN sob o "T" de ESTOQ, e também de colocar os títulos de coluna PREÇO UNIT E VALOR ESTOQ em duas linhas.

Para isso chamamos:

```
.MODIFY COMMAND EXEMPLO.FRM
```

Isso produzirá uma apresentação como a da coluna esquerda na Tela 2. Observe que essa apresentação contém somente as respostas às perguntas apresentadas na Tela 1. Podemos fazer as modificações que queremos movendo o cursor para a posição desejada na tela e digitando as mudanças. A coluna da direita apresenta o relatório revisado, com as modificações sublinhadas.

```
. REPORT
ENTRAR O NOME DO RELATORIO: B: EXEMPLO
ENTRAR OPCOES. M=MARGEM ESQUERDA. L=LINHAS/PAGINA. W=LARGURA DA PAGINA
CABECALHO DA PAGINA? (Y/N) Y
ENTRAR CABECALHO DA PAGINA: RELATORIO DE BANCO DE DADOS EXEMPLO
RELATORIO EM ESPACO DUPLO? (Y/N) N
SAO NECESSARIOS TOTAIS?  (Y/N) Y
SUBTOTAIS NO RELATORIO?  (Y/N) N
RESUMO DO RELATORIO?     (Y/N) N
COL  TAMANHO, CONTEUDO
001  30,NOMEPECA
ENTRAR O CABECALHO: NOME DAS PECAS
002  3,QTIDADE
ENTRAR O CABECALHO: QTIDADE
SAO NECESSARIOS TOTAIS?  (Y/N) Y
003  5,ESTOQMIN
ENTRAR O CABECALHO: ESTOQ MIN
SAO NECESSARIOS TOTAIS?  (Y/N) N
004 10,PRECO
ENTRAR O CABECALHO: PRECO UNIT
SAO NECESSARIOS TOTAIS?  (Y/N) N
005 10, QTIDADE*PRECO
ENTRAR O CABECALHO: VALORESTOQ
SAO NECESSARIOS TOTAIS?  (Y/N) Y
006
```

Tela 1. Exemplo da preparação do formulário do relatório

PAGINA NO. 00001

15/12/84

RELATORIO DO BANCO DE DADOS EXEMPLO

NOME DA PECA	QTIDADE	ESTOQ MIN	PRECO UNIT	VALOR ESTOQ
ALICATES, DE FIO 3/4"	11	5	8.150	42.290
CHAVE DE PARAFUSO, DE CAIXA 3/4"	8	5	6.190	49.520
CHAVE DE PARAFUSO, DE CAIXA 5/8"	4	6	4.290	17.160
MARTELO, DE BOLA 1,5"	5	3	8.150	40.750
MARTELO, DE ORELHA 3/4"	6	6	10.880	65.280
** TOTAL **	34			215.500

Figura 1. Relatório do banco de dados Exemplo


```
Y                                   Y
RELATORIO DE BANCO DE DADOS EXEMPLO RELATORIO DE BANCO DE DADOS EXEMPLO
N                                   N
Y                                   Y
N                                   N
N                                   N
20,NOMEPECA                         20,NOMEPECA
NOME DAS PECAS                      NOME DAS PECAS
3,QTIDADE                           3,QTIDADE
QTIDADE                             QTIDADE
Y                                   Y
5,ESTOQMIN                          5,STR(ESTOQMIN,3)
ESTOQ MIN                           ESTOQ MIN
N                                   10,PRECO
10,PRECO                            > PRECO; UNIT
PRECO UNIT                          N
N                                   10,QTIDADE*PRECO
10,QTIDADE*PRECO                    > VALOR; ESTOQ
VALORESTOQ                          Y
Y
```

Tela 2. Antes e depois do uso do comando EXEMPLO.FRM

As mudanças nos títulos do estoque mínimo foram realizadas simplesmente colocando-se um espaço antes de MIN e deslocando-se um espaço depois dele. Os números foram centralizados sob o "T" pelo uso de STR(ESTOQMIN.3). STR(NOMECAMPO,NÚMERO) faz com que o dBASE II lide com um campo numérico como se ele fosse um campo de caracteres. O valor do NÚMERO pode ser maior ou igual ao do tamanho do campo. Pudemos proceder assim porque não vamos somar a coluna. Observe que precisamos apagar a linha seguinte que continha um "N" em resposta a uma pergunta quanto aos totais. Simplesmente apagando o "N" teríamos encerrado o relatório nessa coluna.

Os títulos PREÇO UNIT e VALORESTOQ foram separados em duas linhas pela inserção de um sinal de ponto e vírgula no lugar em que queríamos a interrupção de linha. Esses títulos estavam alinhados à direita ou colocados do lado direito de uma coluna de dez espaços com o uso de ">". O sinal "<" teria feito o título na extremidade esquerda da coluna (naturalmente, poderíamos nos poupar desse trabalho se de início tivéssemos fixado os títulos como se tivessem cinco espaços). Observe que não precisávamos de um ponto e vírgula para o título ESTOQ MIN. Ele foi acrescentado porque o tamanho do título era maior que o da coluna. Entretanto, deixamos um espaço antes e depois de MIN para preencher a disposição em cinco espaços. A Figura 2 mostra o relatório depois de adaptado.

Impressão de Itens no Sentido Vertical

Muitas vezes desejamos imprimir relatórios com campos dispostos verticalmente. Por exemplo:

Aguiar, Alice A.	Resid. 213-555-3700
Av. Nova Iorque, 45	Comerc. 213-567-1212
São Paulo, SP	
13090	
Almeida, Sérgio V.	Resid. 454-123-4567
etc.	etc.

Suponhamos que o banco de dados tenha os seguintes campos:

NOME	C	30
ENDEREÇO	C	25
CIDADE	C	20
ESTADO	C	2
CEP	C	5
TELRESID	C	12
TELCOMERC	C	12

Ao estabelecer o formato de nosso relatório, podemos dispor esses campos verticalmente em três colunas. Quando recebemos a solicitação para digitar o tamanho e o conteúdo, a primeira coluna é definida assim:

```
30,NOME + ENDEREÇO + '; '
■ + TRIM(CIDADE) + ','+ESTADO + '; ' + CEP
```

O valor "30" se aplica a todas as linhas na primeira coluna. Se o tamanho do conteúdo da coluna for maior que o da coluna especificada, como neste caso, o item será expresso em várias linhas da coluna. Todos esses itens são campos de caracteres. Isso significa que podemos concatená-los (unificá-los) com o uso do sinal de somar (+). O tamanho de coluna foi escolhido de acordo com o tamanho do maior dos campos (NOME). Como o campo NOME preenche a primeira linha da coluna por completo, o campo ENDEREÇO começa no início da segunda linha. Entretanto, ENDEREÇO tem apenas 25 caracteres. Inserimos o sinal de ponto e vírgula nesse ponto para que CIDADE comece na terceira linha. TRIM(CIDADE) foi usado para eliminar os espaços em branco que vêm depois do campo CIDADE, possibilitando a inserção de uma vírgula e do ESTADO em suas posições habituais. O ponto e vírgula após ESTADO ocasiona a separação de mais uma linha. Os dois espaços em branco produzem o deslocamento, fazendo com que o CEP seja impresso a dois espaços para a direita da extremidade da coluna.

A coluna que contém os títulos no sentido horizontal "Resid." e "Comerc." foi criada simplesmente com:

```
4,'Resid Comerc'
```



```
PAGINA NO. 00001

15/10/84

                RELATORIO DO BANCO DE DADOS EXEMPLO

NOME DAS PECAS                  QTIDADE  ESTOQ MIN   PRECO UNIT  VALORESTOQ
ALICATES, DE FIO 3/4"               11        5        3.890       42.290
CHAVE DE PARAFUSO, DE CAIXA 3/4"     8        5        6.190       49.520
CHAVE DE PARAFUSO, DE CAIXA 5/8"     4        6        4.290       17.160
MARTELO, DE BOLA 1.5"                5        3        8.150       40.750
MARTELO, DE ORELHA 3/4"              6        6       10.888       65.280

** TOTAL **                        34                             215.500
```

Figura 2. Relatório do banco de dados Exemplo, revisado

Não foi necessário o sinal de ponto e vírgula porque os quatro espaços de tamanho da coluna forçam a linha a se dividir entre "Resid" e "Comerc". O texto está alojado entre delimitadores porque não há entrada de campo para esse item.

Finalmente, a coluna com os números de telefone é criada com:

```
12,TELRESID + TELCOMERC
```

Novamente, não precisamos de uma interrupção de linha por meio do ponto e vírgula, embora, se quiséssemos os dois números telefônicos separados por uma linha em branco, poderíamos inserir um ponto e vírgula (entre aspas) entre os dois títulos de campo.

Observe que a tela mostra uma linha em branco separando os dados dos dois registros. Isso é obtido dispondo o relatório em espaços duplos. Os espaços duplos inserem uma linha em branco entre os registros do banco de dados; as linhas em branco no interior de um registro são controladas pelo uso dos sinais de ponto e vírgula.

Agrupamento de Dados

Suponhamos agora que queremos imprimir as listas de chamada de uma escola. Uma chamada conterá os nomes dos estudantes em uma classe e sala (cada sala pode ter alunos de classes diferentes). As listas conterão somente os nomes dos estudantes. Podemos utilizar o comando REPORT para prepará-las. Suponhamos que o banco de dados que estamos usando esteja indexado em CLASSE+SALA+NOME.

Para fazer com que o comando REPORT prepare as listas, temos que "trapacear" um pouco. Ao receber a pergunta: SÃO NECESSÁRIOS OS TOTAIS?, responda sim – mesmo que não pretenda utilizar os campos numéricos. Da mesma forma, ao receber a pergunta: SUBTOTAIS NO RELATÓRIO?, também responda sim. Você então receberá a indicação para:

ENTRAR O CAMPO DE SUBTOTAIS:

O campo de "subtotais" pode ser uma expressão. Isso permite:

```
ENTRAR O CAMPO DE SUBTOTAIS: 'CLASSE:
■ ' + CLASSE + '          SALA: ' + SALA
```

Isso resolve o primeiro passo, a combinação entre a classe e a sala. Duas questões que o comando REPORT ainda fará são:

EJETAR UMA PÁGINA APÓS OS SUBTOTAIS? (Y/N):

A resposta afirmativa (Y) fará com que cada lista de chamada se inicie em uma folha de papel separada. A coluna começará com:

```
* CLASSE: 6        SALA 11
```

seguida pela lista de nomes. A pergunta seguinte refere-se ao título de subtotais, tendo a finalidade de inserir texto entre o asterisco e qualquer texto criado pela expressão de subtotais. De modo geral, você não precisará digitar qualquer informação em resposta a essa pergunta.

Normalmente, pode-se usar o sinal de ponto e vírgula para controlar o espacejamento de linha, porém, aqui, isso não funciona. O único modo de se obter uma linha em branco entre a impressão de subtotais e o primeiro nome em uma nova chamada será acrescentar um longo conjunto de espaços em branco, no final da expressão subtotais.

A abordagem acima também funciona quando você realmente deseja calcular o subtotal.

Substituição dos Conteúdos do Campo por Texto

Há ocasiões em que será necessário substituir o conteúdo de um campo por texto. Por exemplo, suponhamos que você tenha um campo denominado DATA, que contém data fornecida da seguinte forma: DD/MM/AA. Uma entrada típica seria 15/10/84, que você iria querer impressa como 15, Out. 1984. Pode-se conseguir isso sob certas condições:

Primeiramente temos de criar duas variáveis de memória:

```
STORE 'janfevmarabrmaijun
■ julagosetoutnovdez ' TO mês
STORE '01 02 03 04 05 06 07 08 09 10 11 12'
■ TO código
```

Criamos duas variáveis de memória em que o início do código numérico para *mês* se alinha com a abreviação em Português. Isso nos permite usar "@", o operador de *substring*, que fornece um valor inteiro, indicando onde uma *substring* aparece na outra. Por exemplo:

```
@ ( '05' ,código)
```


equivale ao número da posição de início da seqüência 05, na *string* de caracteres *código*; neste caso, 13.

Então utilizamos o operador de *substring* "$" para obter a parte especificada da *string* da posição de início dada,

```
$(mês,13,3)
```

que naturalmente corresponde a maio. Agora podemos unir as duas da seguinte forma:

```
$(mês,@("05",código),3)
```

Para relacionar esse dado ao conteúdo do campo DATA, lembre-se de que o mês corresponde ao primeiro e segundo caracteres do campo. Isso nos leva à substituição seguinte:

```
$(mês,@($(DATA,1,2),código),3)
```

que, se for fornecida como a expressão para o conteúdo do campo, fará com que REPORT imprima o mês correspondente em Português, em vez do conteúdo efetivo do campo. Para obter o resultado desejado, teríamos de entrar o tamanho de coluna e o conteúdo como:

```
12,$(mês,@($ (DATA,1,2,)código),3 + '  '
@ + $(DATA,4,2) + ',19' + $(DATA,7,2))
```

Essa expressão transforma a entrada 15/10/84 no resultado impresso: 15 out, 1984.

Você pode usar esse esquema para qualquer número de substituições que desejar fazer, mas ele também pode ocasionar alguns problemas. Se houver uma entrada para a qual não há código, você receberá uma mensagem de ERRO DE SINTAXE e toda a operação se interromperá. Entretanto, sendo cuidadoso, será possível utilizar esse artifício de modo eficiente.

Um outro emprego desse artifício serão as entradas verdadeiro/falso. Se puder usar um campo de caracteres com T e F, você poderá obter uma disposição com mais de uma coluna de .T, e .F. Com o nosso exemplo da escola, se quisermos uma disposição que apresente REPROVADO se o aluno foi reprovado e nada se ele foi aprovado, poderíamos usar:

```
STORE"REPROVADO          "TO texto
STORE"F          T       "TO código
```

A entrada no relatório seria:

```
4,$(texto,@(REPROVADO,código),4)
```

Controle da Margem

A primeira solicitação que você encontra quando insere o comando REPORT é:

```
ENTRAR OPÇÕES, M = MARGEM ESQUERDA,
■ L = LINHAS/PÁGINA, T = TAMANHO DA PÁGINA
```

A disposição padronizada para a margem esquerda é de oito caracteres de espaço. Isso corresponde a 8/10 de polegada, usando-se uma impressora que imprima a dez caracteres por polegada. Esse tamanho de margem mal dará para colocar os relatórios em uma pasta/arquivo; é completamente inadequado se você estiver usando uma impressora de 12 ou de 16 cpi. Para adaptar a margem esquerda para um número maior, 15, por exemplo, digite M = 15 em resposta à pergunta. Isso fará com que o relatório se inicie a 15 espaços da extremidade esquerda do papel.

Você também poderá adaptar a margem esquerda por meio do teclado, sem modificar a disposição de margem do comando REPORT. O comando SET MARGIN TO 10, por exemplo, fará com que todo o texto se desloque em dez espaços. Isso significa que o relatório ficará com uma margem esquerda de 18 caracteres (dez espaços devido ao uso do comando SET MARGIN TO e oito, devido à disposição padronizada). O comando SET MARGIN TO afeta somente a impressora, não a tela. Ele também produz uma margem para outros comandos de impressão, como LIST.

De modo semelhante, a largura da página será o número de espaços de caracteres a partir da margem esquerda, antes da mudança de linha. A largura padrão é de 80 (isso protege o cilindro da impressora). Se estiver usando uma impressora de 12 ou de 16 cpi ou papel de 15 polegadas, você provavelmente desejará aumentar a largura para um número adequado à largura usada em sua impressora, bem como do papel utilizado. A Figura 3 apresenta algumas comparações em termos de número de caracteres por linha para diversas larguras de páginas e tamanhos de tipos.

O tamanho da página é o número de linhas em uma página. O tamanho padrão é de 57 linhas. Isso é adequado para papel de 11 polegadas de altura, sem espacejamento duplo. Se quiser que a margem inferior fique maior, tente adaptar o número de linhas para menos de 57.

Cabeçalhos de Páginas

O comando REPORT permite definir cabeçalhos que não o padronizado. O cabeçalho adicional é impresso na linha que contém o número de página, permitindo a inclusão de texto específico relativo ao dia em que o relatório é executado, inserindo-o na parte superior da página, sem necessidade de modificar o relatório. Por exemplo:

```
SET HEADING TO RELATÓRIO ESPECIAL
■ DO DIA ANTERIOR AO INVENTÁRIO
```

Você pode fazer um uso especial dessa característica, de dentro do programa. Por exemplo, você talvez queira a mesma estrutura de relatório para vários clientes. O programa poderia fazer o seguinte:

```
STORE 'INVENTÁRIO PARA ABERCROMBIE
■ E CZYNCOWICZ' TO título
```



```
SET HEADING TO &título

REPORT FOR INVENTÁR = 'A&C'
```

Você teria de entrar o comando STORE para cada cliente e então deixar o programa fazer as listagens de inventário para cada um deles.

Relatórios Utilizando Dois Bancos de Dados

Você pode utilizar dois bancos de dados para criar um relatório único sob certas condições. O segundo banco de dados deve ser uma extensão do primeiro. Isto é, os registros devem corresponder em termos de números de registros. Quando assim, a disposição do relatório será:

```
.SELECT SECONDARY
.USE FILE2
.SELECT PRIMARY
.USE FILE1
.SET LINKAGE ON
.REPORT
```

Extensão	10 cpi	12 cpi	16cpi
8 1/2	85	102	136
15	150	180	240

Figura 3. Número de caracteres por linha

O comando REPORT pode selecionar campos de qualquer um dos dois bancos de dados. Se dois campos possuem o mesmo nome, você deverá distingui-los com o uso de "S." e "P." imediatamente antes do nome de campo (isto é, S.ENDEREÇO). Caso se decida preparar um relatório usando-se dois arquivos, é preciso lembrar de selecionar ambos os arquivos e de posicioná-los no Registro 1.

Relatórios Seletivos

O comando REPORT pode ser usado para selecionar apenas determinados itens para um relatório. Há dois mecanismos para selecionar registros – FOR e WHILE.

Se os registros desejados estiverem distribuídos por todo o banco de dados, a condição FOR será a melhor técnica. Por exemplo, suponhamos que temos um banco de dados de

estudantes que foi indexado pelo nome do estudante, porém queremos que o comando REPORT prepare uma lista de chamada para uma única sala e classe.

```
.REPORT FOR SALA = '11' .AND.CLASSE = '6'
```

Aqui está um exemplo perfeito do uso do comando HEADING. Antes de usar o último comando, digite:

```
.SET HEADING TO SALA 11 CLASSE 6
```

Isso registrará na primeira linha do relatório: SALA 11 CLASSE 6.

Se seu banco de dados estiver indexado tanto por CLASSE como por SALA, você também poderá preparar o mesmo relatório com o uso de WHILE. Primeiramente, você deve posicionar o banco de dados no registro desejado. O comando

```
.FIND 611
```

posicionará o banco de dados no primeiro registro para a classe 6 e sala 11. O relatório pode ser preparado por:

```
.SET HEADING TO SALA 11 CLASSE 6
.REPORT WHILE SALA = '11' .AND.
■ CLASSE = '6'
```

O comando REPORT é extremamente potente. O melhor modo de experimentá-lo consiste em usar a tela com um pequeno arquivo de banco de dados. Como é um comando apenas de leitura, não haverá como danificar o conteúdo do banco de dados. O pior que poderá acontecer ao se cometer um erro ao usá-lo será receber uma mensagem de ERRO DE SINTAXE; então com isso, você fica sabendo que deve tentar uma outra abordagem.

Capítulo Doze

A DATA

A "data" merece atenção especial. É um item que usamos freqüentemente em aplicações de todos os tipos.

A Data do Sistema

A data do sistema é armazenada na memória em uma posição chamada DATE ( ). Se você tem um computador do tipo PC (16 bits), o clock do sistema é lido quando o dBASE II é carregado e a data é automaticamente estabelecida. Com computadores de 8 bits, recebe-se uma mensagem solicitando a entrada da data quando o dBASE II é carregado.

A data do sistema é escrita no cabeçalho do banco de dados sempre que se faz uma modificação nele. Os relatórios preparados por meio do comando REPORT utilizam a data do sistema e ela também pode ser usada em uma série de programas aplicativos.

Por vários motivos, a data do sistema pode não ser estabelecida apropriadamente. Muitos têm a tendência de teclar <RETURN> quando o sistema operacional ou o dBASE II solicitam a data. Nos equipamentos exceto os do tipo PC, a operação de estabelecimento da data é ignorada completamente quando um programa é carregado juntamente com o dBASE II (você pode carregar o dBASE II e um outro programa, usando um único comando – digitando DBASE seguido do nome do programa).

Teste e Estabelecimento da Data do Sistema

Podemos facilmente verificar a data do sistema para saber se foi ou não estabelecida. Em computadores em que o clock do sistema é lido automaticamente, a data será estabelecida em alguma de maneira arbitrária (como 01/01/80) se não ajustarmos o clock ao comprar o

computador. Em computadores de 8 bits, nos quais a data *não* é lida, ela é ajustada para 00/00/00. Podemos testar a data examinando DATE ( ).

A data pode ser estabelecida diretamente com o uso de SET DATE TO. Um programa simples para testar e estabelecer a data é mostrado pelo Programa 1.

```
SET TALK OFF
IF $(DATE(),7,2)+$(DATE(),1,5) < "8501/01"
   STORE DATE() TO mdata
   ERASE
   @ 1,1 SAY "ENTRAR A DATA COMO DD/MM/AA" GET mdata PICTURE "99/99/99"
   READ
   SET DATE TO &mdata
ENDIF
RETURN
```

Programa 1. Teste e estabelecimento da data do sistema

Observe a sintaxe do comando SET DATE TO *&mdata*. Sem o sinal &, que permite o uso do valor armazenado em uma variável de memória em uma linha de comando, os valores 00/00/00 seriam apresentados como a data do sistema.

Validação da Data

O Programa 1 não valida a data – poderíamos digitar, e o sistema aceitaria, 32/15/85 como data válida. Há dois modos de testar dados ilegítimos, embora seja extremamente difícil testar quanto a dados errôneos "legítimos".

O primeiro deles estabelece um pequeno banco de dados com um único campo DIAS, um campo numérico, com um tamanho de campo de valor dois, para o número de dias do mês. O número de registro corresponde ao mês (o Registro 1 possui a entrada 31 etc.). Chamamos esse banco de dados de DATAS. O programa que verifica a data (mês e dia somente) é apresentado como o Programa 2.

O Programa 2 ilustra o uso de um loop DO para testar erros, o uso da função VAL e o uso de DO CASE.

No Programa 2, primeiramente estabelecemos um flag do sistema, *datarvim* para Verdadeiro (True) e criamos a variável de memória *mdata*. Então, criamos um loop DO que se manterá em execução enquanto o flag *datarvim* for True. Por que usar um loop DO? Sem ele, temos que supor que o usuário irá digitar a data correta na primeira tentativa. Mas, se for esse o caso, sequer precisamos do teste dos dados. O comando PICTURE fornece um primeiro nível de depuração de dados inadequados; só será possível inserir espaços em branco ou dados numéricos nessa variável.

Nosso passo seguinte será testar a validade do mês e do dia. Para isso, criamos duas variáveis numéricas, *mês* e *dia*, com as partes apropriadas da *string* de caracteres *mdata*. Isso é obtido por meio da função VAL. Uma vez estabelecidas essas variáveis, podemos executar nosso teste, o que fazemos em dois estágios. Primeiro, verificamos quanto à validade do mês. Se o mês não for válido, informamos ao usuário imprimindo MÊS NÃO VÁLIDO na tela. Se

o teste do mês resultar válido, verificamos o dia do mês. GO *mês* posiciona o banco de dados no registro correspondente ao mês. Nós optamos por usar DO CASE em nosso teste.

Esse é um bom exemplo do uso de DO CASE. Os "CASES" (CONDIÇÕES) são testadas pela ordem, até que uma (ou nenhuma delas) seja atendida. A ordenação das condições é importante. O primeiro teste verifica se o operador realmente forneceu uma data positiva (o comando PICTURE não garante contra sinais negativos). Se tivéssemos colocado a terceira condição em primeiro lugar, a –6, por exemplo, teria sido considerado válido. A segunda condição é um teste para ano bissexto. Se ela ou uma terceira condição forem atendidas, o flag do loop receberá a atribuição FALSE para encerrar o loop.

Observe que há mensagens de erro especiais para cada um dos erros possíveis. A cada passagem pelo loop, apagamos a Linha 3 na tela usando: @3,0. Isso evita qualquer ambigüidade na mensagem de erro, caso o usuário cometa os dois erros possíveis em tentativas sucessivas.

O Programa 3 é uma adaptação alternativa de um programa de validação de data. Ele exemplifica a utilização da função para escolher um determinado item de uma segunda *string* de caracteres. Ele também ilustra o uso de comparações de *substrings* de representações de caracteres dos dados "numéricos" (as datas).

Esse programa é muito mais "engenhoso" do que o Programa 2, embora não tão fácil de se acompanhar. Em muitos dos sistemas ele "funcionará" muito mais rapidamente que o Programa 2, porque não estamos "abrindo" um arquivo de banco de dados.

```
SET TALK OFF
IF $(DATE(),7,2)+$(DATE(),1,5) < "8301/01"
   USE Datas
   STORE T TO dataruim
   STORE DATE() TO mdata
   ERASE
   DO WHILE dataruim
     @ 1,1 SAY "ENTRAR A DATA COMO DD/MM/AA" GET mdata PICTURE "99/99/99"
     READ
     @ 3,0
     STORE VAL($(MDATA,1,2)) TO mes
     STORE VAL($(MDATA,4,2)) TO dia
     IF mes >= 1 .AND. MES <= 12
        GO Mes
        DO CASE
           CASE dia <= 0
                @ 3,20 SAY "DIA INVALIDO"
           CASE mes=2 .AND. $(mdata,7,2)$'84,88' .AND. dia <= DIAS+1
                STORE F TO dataruim
           CASE dia <= DIAS
                STORE F TO dataruim
           OTHERWISE
                @ 3,20 SAY "DIA INVALIDO"
        ENDCASE
     ELSE
        @ 3,1 SAY "MES INVALIDO"
     ENDIF
   ENDDO
   SET DATE TO &mdata
   RELEASE mdata, mes, dia, dataruim
ENDIF
RETURN
```

Programa 2. Testa, valida e estabelece a data do sistema

As variáveis *mascmês* e *mascdia* devem ser usadas para determinar o número de dias em um determinado mês. Observe que o número de dias (*mascdia*) está em alinhamento exato com *mascmês*. Observe também que os meses de numeração menor que "10" são representados precedidos por um "0". A técnica que vamos utilizar é bastante rápida, porém, se você não for cuidadoso, ela será potencialmente desastrosa.

```
SET TALK OFF
IF $(DATE(),7,2)+$(DATE(),1,5) < "3301/01"
   STORE DATE() TO mdata
   STORE "01 02 03 04 05 06 07 08 09 10 11 12" TO mascmes
   STORE "31 28 31 30 31 30 31 31 30 31 30 31" TO mascdia
   STORE T TO dataruim
   ERASE
   DO WHILE dataruim
      @ 1,1 SAY "ENTRAR A DATA COM DD/MM/AA" GET mdata PICTURE "99/99/99"
      READ
      @ 3,0
      SET DATE TO &mdata
      STORE DATE() TO mdata
      IF $(mdata,1,2)$mascmes
         STORE $(mascdia,@($(mdata,1,2),mascmes),2) TO dias
         IF $(mdata,1,2) = "02" .AND. $(mdata,7,2)$"84, 88"
            STORE "29" TO dias
         ENDIF
         IF $(mdata,4,2) >= "01" .AND. $(mdata,4,2) <= dias
            STORE F TO dataruim
         ELSE
            @ 3,20  SAY "DIA INVALIDO"
         ENDIF
      ELSE
         @ 3,1 SAY "MES INVALIDO"
      ENDIF
   ENDDO
   SET DATE TO &mdata
   RELEASE mdata,dias,dataruim,mascmes,mascdia
ENDIF
RETURN
```

Programa 3. Adaptação alternativa do Programa 2

Após a "leitura" da variável *mdata*, atribuímo-la a DATE( ), usando SET DATE TO *&mdata* e então cancelamos essa atribuição. Isso nos permite aproveitar o fato de que DATE sempre armazena dígitos numéricos, de 0 a 9, em cada uma das posições para os dados. As posições de caracteres 3 e 6 são sempre um sinal "/". Precisamos desse recurso para evitar problemas na linha seguinte.

O primeiro IF verifica se os dois primeiros caracteres de *mdata* podem ser encontrados em *mascmês*. Caso não, recebemos uma mensagem de erro MÊS NÃO VÁLIDO. Se tivéssemos admitido a possibilidade de um espaço em branco no primeiro caractere de *mdata*, poderíamos localizar uma data, como janeiro, iniciando um espaço de caractere antes do "10" em *mascmês*. Isso nos levaria a um alinhamento incorreto para a verificação do número de dias.

A linha seguinte, onde criamos a variável *dias*, é especialmente instrutiva. Ela utiliza os dois primeiros caracteres de *mdata* para armazenar o número máximo de dias em um mês diretamente em *dias*. Observe que *dias* é uma *string* de caracteres. Agora vamos explicar o que acontece nessa linha.

Suponhamos que os dois primeiros caracteres de *mdata* sejam 09. A função @($(*mdata*1,2), *mascmês*) fornece um número que é a posição de início de 09 na *string mascmês*. A função de *substring* sempre tem a forma:

$(<nome da variável >, posição de início, tamanho)

Tudo que estamos fazendo é usar @, a função de busca de *substring*, para nos colocar na posição de início. Isso acaba por armazenar os dois caracteres diretamente abaixo de 09 em *dias*, uma *string* de caracteres.

A seguir, testamos o ano bissexto. Se estivermos em fevereiro e for um ano bissexto, armazenamos os caracteres "29" em dias.

Agora, comparamos o quarto e o quinto caracteres com nossos limites. Observe que podemos comparar *strings* de caracteres exatamente como se fossem números. Isso porque, de fato, estamos comparando valores ASCII. Além disso, tomamos a precaução de fazer com que cada uma das posições de caractere seja um caractere numérico. Se passarmos no teste, colocamos o flag *datarvim* em Falso e saímos; caso não, apresentamos a mensagem de erro MÊS NÃO VÁLIDO e recomeçamos todo o processo.

A Data como um Item de Menu

Em todos os sistemas você poderá inserir um programa diretamente a partir do sistema operacional, digitando DBASE seguido pelo nome do programa. Quando usado com o sistema CP/M, isso faz ignorar o processo pelo qual o dBASE II solicita ao usuário que digite a data do sistema.

Ao ser ignorado o processo de entrada da data, o dBASE II impede o uso da data do sistema como fonte da entrada de dados. Para se superar isso faz-se com que o programa menu apresente a data e com que uma das seleções de menu possibilite a opção de modificar a data.

Um modo de obter uma fonte para a data é manter a última data armazenada no disco em um arquivo de memória (.MEM). Então, quando o programa é carregado, um dos comandos do programa lê o arquivo de disco e usa-o para a data do sistema.

O Programa 4 demonstra a restauração (leitura) de variáveis de memória de um arquivo em disco e a incorporação de uma data e de um programa de mudança de data do menu. O primeiro processo é obtido por meio do comando RESTORE FROM DATA ADDITIVE. A palavra ADDITIVE é opcional. Porém, sem ela, todas as variáveis de memória que estiverem sendo utilizadas quando o comando RESTORE é usado serão apagadas.

Não fica explícito o fato de que restauramos duas variáveis de memória: *mdata* e *cdata*. A variável *mdata* apresenta a data no formato convencional DD/MM/AA, *cdata* é a versão "por extenso" da mesma data. Se *mdata* contiver 11/10/84, *cdata* conterá 11 out 1984. Usamos *cdata* como a apresentação da data porque todos nós, uma ou outra vez, fazemos a transposição do dia e do mês quando eles são apresentados como números. O Programa 4 possibilita a validação visual positiva da data.

```
SET TALK OFF
RESTORE FROM Data ADDITIVE
SET DATE TO &mdata
DO WHILE T
   ERASE
   STORE " " TO escolha
   @  5,30 SAY "DISTRIBUIDORA FERREIRA"
   @  7,35 SAY cdata
   @  9,30 SAY "1 - MODIFICAR A DATA"
   @ 10,30 SAY "2 - MENU DE CLIENTES"
   @ 11,30 SAY "3 - MENU DE INVENTARIO"
   @ 12,30 SAY "4 - ENTRADA DE PEDIDOS DE VENDA"
   @ 13,30 SAY "5 - MENU DE LISTAS DE ENDERECAMENTO"
   @ 14,30 SAY "6 - MENU DE FATURAS"
   @ 15,30 SAY "0 - ABANDONAR"
   @ 18,32 SAY "  - ESCOLHER UMA"
   @ 18,37 GET ESCOLHA
   READ
   DO CASE
      CASE escolha = "1"
           DO Data
      CASE escolha = "2"

           ETC

   ENDCASE
ENDDO
```

Programa 4. Programa menu que usa a data apresentada

O Programa 5, chamado para modificar a data, inicia com o comando CLEAR GETS. Nós não apagamos a tela quando o inserimos; o menu ainda está sendo exibido. O comando CLEAR GETS é usado para evitar que o cursor se mova para a área da tela reservada para ESCOLHA.

Então criamos as três variáveis de memória para verificar a data e passá-la para a apresentação "por extenso". Essas variáveis são mantidas fora do loop DO pois não há necessidade de recriá-las a cada passagem pelo loop (mantenha-se o máximo possível no interior do loop).

O resultado deste programa é basicamente o mesmo que o apresentado no Programa 2, exceto que foi modificada a posição que estamos usando na tela. Observe que apagamos a Linha 7 antes de inserir o loop DO. A Linha 7 é o ponto em que a data estava sendo apresentada. Se nossa nova Linha 7 não sobrepuser completamente o texto anterior, parte de ambos será apresentada, ocasionando confusão. Será melhor garantir, apagando a linha primeiramente.

Após termos saído do loop, tendo digitado uma data válida, passamos a data para a apresentação "por extenso", com o uso das duas linhas de comando que envolvem *cdata*. Embora essas duas linhas possam ser incorporadas em um único comando, usamos dois estágios devido ao tamanho do comando na página.

Para armazenar nossa data com segurança no disco, usamos o comando SAVE ALL LIKE *?data* TO DATA. Esse comando permite gravar somente as variáveis que possuem nomes com cinco caracteres e que se encerram em DATA no arquivo em disco. O sinal de ? é um caractere-chave – significa qualquer caractere ou um espaço. A expressão ?DATA é o "esqueleto" do nome da variável.

A Data como um Campo

As datas são campos muito usados: datas de nascimento, de matrícula, de transações, de promoções etc. Ainda assim, a data coloca um problema específico para o programador do dBASE II. Nos Estados Unidos, prefere-se escrevê-la como MM/DD/AA. Os europeus, DD/MM/AA, enquanto os japoneses usam AA/MM/DD. Somente os japoneses usam datas sob uma forma que é natural para o computador.

Se quisermos testar se uma determinada data recai entre duas outras datas (data mais antiga <data testada <data mais recente), as datas deverão ser dispostas como AA/MM/DD. Isso porque a comparação padrão começa pelo caractere mais significativo (ou dígito) que é o da esquerda. Tomemos três datas: A, B e C.

A. 02/01/84
B. 11/11/85
C. 01/01/86

Queremos colocar essas datas na ordem A, B, C – da mais antiga para a mais recente. Sem qualquer manipulação, o computador as ordenaria como C, A, B.

As datas são mais freqüentemente inseridas como campos de caracteres, geralmente com um tamanho de oito. Isso pode dificultar mais ainda o problema. Temos aqui datas fornecidas de várias formas:

D. 01/01/84
E. 2/ 1/84
F. 1 /1 /84

Para o computador, a data E é menor que a data D. A comparação é feita da esquerda para a direita, de acordo com o valor ASCII de cada caractere. Os valores ASCII para caracteres são apresentados no Apêndice D.

A menos que pretendamos nunca fazer nada com as datas, devemos estabelecer algum método seguro de lidar com elas. No mínimo, precisamos excluir a possibilidade de armazená-las na forma em que se encontram os exemplos E e F. Enquanto não o fizermos, não poderemos executar com segurança qualquer operação que envolva datas.

É fácil garantir que os dados estejam coerentemente armazenados sob a forma do exemplo D. Podemos utilizar o comando SET DATE TO para garantir a estrutura do formato da data. O Programa 6 é um exemplo de sub-rotina que pode realizar isso. Observe que usamos o comando STORE para armazenar a data do sistema em uma posição segura e então a restauramos, com o comando RESTORE, antes de usar o comando REPLACE. Na verdade, precisamos restaurar a data do sistema somente antes de fechar o arquivo. A data do cabeçalho é gravada novamente com a data do sistema quando o arquivo é fechado. Observe que armazenamos cada campo em uma variável antes de usar SET DATE TO (temos de usar o sinal & antes do nome da variável; não se pode usá-lo com um nome de campo).

A técnica de se usar a data do sistema para formatar a data, e depois usar a data do sistema para substituir o conteúdo de um campo, pode ser usada eficazmente sempre que um

```
* Programa -------: Menu1.CMD (ou MENU1.prg)
* Obs.------------: Este programa armazena a data em um arquivo de memoria.

CLEAR GETS
STORE "01 02 03 04 05 06 07 08 09 10 11 12" TO mascmes
STORE "31 28 31 30 31 30 31 31 30 31 30 31" TO mascdia
STORE "JANFEBMARABRMAIJUNJULAGOSETOUTNOVDEZ" TO calendario
STORE T TO dataruim
ERASE
@ 7,0
DO WHILE dataruim
   @ 7,1 SAY "ENTRAR A DATA COMO DD/MM/AA" GET mdata PICTURE "99/99/99"
   READ
   @ 8,0
   SET DATE TO &mdata
   STORE DATE() TO mdata
   IF $(mdata,1,2)$mascmes
      STORE $(mascdia,@($(mdata,1,2), mascmes), 2) TO dias
      IF $(mdata,1,2) = "02" .AND. $(mdata,7,2)$"84,88"
         STORE "29" TO dias
      ENDIF
      IF $(mdata,4,2) >= "01" .AND. $(mdata,4,2) <= dias
         STORE F TO dataruim
      ELSE
         @ 8,20 SAY "DIA INVALIDO"
      ENDIF
   ELSE
      @ 8,1 SAY "MES INVALIDO"
   ENDIF
ENDDO
@ 8,0
SET DATE TO &mdata
STORE DATE() TO mdata
STORE $(calendario,@($mdata,1,2),mascmes),3) TO cdata
STORE $(mdata,4,2)+" "+cdata+" 19"+$(mdata,7,2) TO cdata
SAVE ALL LIKE ?data TO Data
RELEASE mes,dia,mascmes,mascdia,calendario,dias,dataruim
RETURN
```

Programa 5. MENU1.CMD/.PRG Programa para modificar e verificar a data do sistema

campo tiver de ser substituído pela data atual. Isso também garante a uniformidade na estrutura dos conteúdos do campo da data.

Mesmo fornecendo uma estrutura sólida, podemos ficar sem alternativas. Não podemos obter uma classificação significativa de um campo de data. Podemos indexar com o uso do artifício:

```
INDEX ON $(CAMPODATA,7,2)
■ + $(CAMPODATA,1,6,) TO datas
```

Isso organizará o banco de dados pela ordem da data. Entretanto, o comando FIND deverá ser usado como FIND <aa/mm/dd>.

O uso de qualquer operação lógica, exceto DAT = "<data>", pode se tornar confuso. Como exemplo, se quisermos apresentar os registros onde uma data é anterior a uma data de teste, usamos:

```
DISPLAY FOR $(DATA,7,2) + $(DATA,1,6)
■ < $(mdata,7,2) + $(mdata,1,6)
```

Podemos evitar uma série de problemas adotando o estilo de data usado pelos japoneses – AA/MM/DD. Isso nos permite usar o comando SORT para a classificação e utilizar a indexação direta, podendo simplificar os procedimentos lógicos usados.

Outros Métodos de Armazenar Datas

Um outro método de armazenamento da data é obtido pelo dia juliano. O dia juliano é um esquema usado pelos astrônomos, pelo qual os dias recebem um número em série. Os astrônomos começam por 1 de janeiro de 4713 AC. O dia 1 de janeiro de 1965 é o dia juliano 2.438.762. Variações dessa idéia (propostas por Joseph Scalizer, em 1938) são freqüentemente usadas em relógios digitais e em calculadoras. Um contador com seis dígitos pode funcionar como um calendário com um ciclo de 2737 + ano (divida 999.999 por 365,25). O dia juliano também nos permite determinar o dia da semana para qualquer data no limite do calendário. Isso será útil se estivermos tentando desenvolver um programa para algum sistema de programação de horário (o uso dos dígitos de seis dados de uma representação normal da data permite somente um calendário de 99 anos sem ambigüidades).

O desenvolvimento do sistema do dia juliano se torna um pouco complexo devido a nosso sistema específico de calendário, o calendário gregoriano, que é derivado do calendário juliano. O calendário juliano não tem relação com o dia juliano; foi estabelecido por Júlio César em 46 AC. Ele fixava o ano em 365 dias com um ano bissexto a cada quatro anos.

Infelizmente, o estabelecimento de um calendário não é uma tarefa simples: o ano tem na verdade um pouco menos de 365 1/4 dias. Quando foi determinado em 1582 que a data correspondente ao Equinócio de Primavera estava atrasado em dez dias, o Papa Gregório XIII eliminou dez dias do calendário juliano para corrigir a data e estabeleceu o calendário gregoriano que usamos até hoje.

```
-----------
@ 7,1 SAY "ENTRAR A DATA DE NASCIMENTO" GET datan PICTURE "99/99/99"
@ 8,1 SAY "ENTRAR A DATA INICIAL" GET datai PICTURE "99/99/99"
READ
STORE DATE() TO seguro
STORE datan TO mdatan
SET DATE TO &mdatan
STORE DATE() TO mdatan
STORE datai TO mdatai
SET DATE TO &mdatai
STORE DATE() TO mdatai
SET DATE TO &seguro
REPLACE datan WITH mdatan, datai WITH mdatai
-----------
```

Programa 6. Sub-rotina para garantir a uniformidade na entrada das datas

```
* Programa para calcular a "data juliana" a partir do calendario
SET TALK OFF
ERASE
STORE "  /  /  " TO data
@ 1,1 SAY "ENTRAR A DATA" GET data PICTURE "99/99/9999"
READ
SET DATE TO &data
STORE VAL($(data,7,4)) TO ano
STORE 0 TO baseano
STORE ano-baseano TO ano

* primeiramente testamos se o ano e bissexto

DO CASE
   CASE ano-INT (ano/400)*400=0
        STORE T TO anobissex
   CASE ano-INT (ano/100)*100=0
        STORE F TO anobissex
   CASE ANO-INT (ANO/4)*4=0
        STORE T TO anobissex
   OTHERWISE
        STORE F TO anobissex
ENDCASE

* calcular o dia do ano

STORE "01 02 03 04  05  06  07  08  09  10  11  12" TO meses
IF anobissex
   STORE "  31  60  91 121 152 182 213 244 274 305 335  366" TO basedia
   STORE 0 TO ajustar
ELSE
   STORE 1 TO ajustar
   STORE "  31 59  90 120 151 181 212 243 273 304  334  365" TO basedia
ENDIF
STORE VAL($(DATE(),4,2)) TO dia
STORE VAL($(basedia,@($(DATE(),1,2),meses),3)) + dia TO diadoano

* determinar o numero de anos bissextos desde o ano base

STORE INT(ano/4) - INT(ano/100) + INT(ano/400) + ajustar TO anosbissex

* agora calcular a data juliana

STORE 365* (ano) + anosbissex + diadoano TO datajulian
? "DATA JULIANO = ", datajulian
RETURN
```

Programa 7. Programa para converter uma data em um dia juliano

Embora o calendário gregoriano fosse quase imediatamente adotado por países de religião Católica Romana, a Europa protestante, especialmente a Inglaterra, estava menos propensa a aceitá-lo. De fato, a Inglaterra e as colônias americanas só adotaram o calendário gregoriano após 1752, e mesmo então, em meio a enorme protesto. Afinal de contas, as pessoas estavam sendo "roubadas" em onze dias de suas vidas (o espaço de tempo entre 1582 e 1752 foi suficiente para acrescentar um outro dia). A estação de crescimento seria mais curta e as pessoas iriam ganhar menos dinheiro nesse ano.

O calendário gregoriano tem um ano bissexto a cada quatro anos, exceto nos anos centenários. E cada quarto ano centenário é também um ano bissexto. Os anos centenários que são exatamente divisíveis por 400 são anos bissextos. Assim, 1600 e 2000 são anos bissextos,

mas não 1700, 1800 e 1900. Isso é bastante complexo para nos fornecer alguns exemplos interessantes de programação (os Programas 7, 8 e 9).

O Programa 7 toma uma data do calendário padrão e a converte em uma "data juliana" baseada em 1 de janeiro, 0. Não existe realmente essa data, mas ela fornece um número serial para cada dia da Era Cristã. Nesse programa, nós não colocamos limites na data, nem verificamos quanto à validade da data fornecida.

O Programa 7 é também um exemplo da programação com o dBASE II sem o uso de um banco de dados. O programa utiliza uma combinação de buscas de *string* e de cálculos aritméticos para chegar ao resultado – um número em série para uma data.

Observe que a variável de entrada de data reserva quatro dígitos para o ano. Uma vez fornecida a data, nós nos asseguramos de que os dígitos estejam alinhados convenientemente, armazenando-a na data do sistema. A seguir, armazenamos o "ano" em uma variável numérica. Vamos utilizar vários cálculos aritméticos ao usar o ano. O *anobase* é fornecido com o subprograma incluído, para maior facilidade do leitor. Ele funcionará com qualquer anobase que seja exatamente divisível por 400.

Novamente, observe que "comentamos" o programa. Embora as linhas que começam por "*" sejam ignoradas pelo dBASE II, os comentários efetivamente diminuem um pouco a velocidade de execução do programa. Os comentários podem ser muito úteis quando posteriormente se lê um programa que recorre a procedimentos artificiais. Se usar algum artifício em seu programa, você deverá realmente comentá-lo (você sempre poderá fazer uma cópia "operativa" sem os comentários, se a velocidade for importante para você).

Para calcular o dia juliano, precisamos determinar se o ano é um ano bissexto. Essa é uma operação em que o DO CASE pode ser muito útil. Esse comando executa somente a primeira condição (CASE) encontrada. Em nosso programa, a ordem na qual inserimos as condições é importante. Por exemplo se tivéssemos colocado a Condição 3 em primeiro lugar, os anos centenários seriam declarados como anos bissextos. Se tivéssemos colocado a Condição 2 primeiro, os anos centenários bissextos seriam declarados como anos não-bissextos. A função INT elimina as casas decimais de números ou expressões. Quando temos a oportunidade de utilizar o comando CASE com sua finalidade própria – em vez de como um substituto abreviado de IF – freqüentemente precisamos fazer um planejamento. Usado com CASE, OTHERWISE significa "tudo mais" e se assemelha a ELSE em IF/ELSE.

Calculamos o dia do ano somando o dia ao número acumulado de dias até o final do mês anterior. Os dias acumulados estão contidos por *basedia*. Fazemos a escolha entre duas possíveis *basedias*, utilizando o ano bissexto.

A seguir, temos de calcular o número de anos bissextos desde a data base. Novamente utilizamos a função INT, como no DO CASE.

Finalmente chegamos ao cálculo efetivo. Todo o problema consiste em determinar o número de anos bissextos que deve ser usado no cálculo final. Esse é um bom exemplo de programa no sentido de que força a dar atenção a detalhes e resolve um problema simples.

O Programa 8 converte o calendário novamente em data; é o mesmo problema, ao reverso. Para acelerar o processo, calculamos o ano dividindo o dia juliano por 365. Isso nos dá o cálculo de um ano que é maior ou igual ao ano efetivo. O erro aumenta em proporção ao tamanho do

ano. Isso porque o ano calculado não leva em conta os anos bissextos. A lógica exige que calculemos o ano e então calculemos as correções necessárias.

Observe que o código é quase idêntico ao usado no Programa 7. Utilizamos exatamente os mesmos testes para o ano bissexto e o mesmo cálculo para determinar o número de anos bissextos.

Uma vez calculada a data, nós a aprimoramos pelo loop DO. Essas datas só precisam ser processadas quando o resto for menor ou igual a zero. Essas poucas datas positivas foram operadas direta e completamente durante nossa primeira passagem por um resultado.

Observe o modo pelo qual calculamos o dia e o mês. Essa é uma técnica semelhante à usada para gerenciar dados com o uso de linguagens convencionais de programação. Aqui, indexamos nossa passagem por um conjunto de dados, um item por vez, até alcançarmos o resultado desejado.

O tempo de execução desse programa é respeitável. Entretanto, se for necessário freqüentemente esse tipo de cálculo (várias vezes em um registro de banco de dados ou em uma busca do banco de dados), esse é o tipo de programa que deve ser executado em linguagem assembly e usado com o dBASE II por meio dos comandos LOAD e CALL do dBASE II.

O Programa 9 ilustra a facilidade com que podemos determinar o dia da semana para qualquer data a partir do dia juliano. O programa fornecerá o resultado correto para nosso ano base de zero. Se escolhermos um outro ano base será necessário adaptar o *codigosem* para obter o resultado correto.

Tudo que precisamos fazer é determinar o resto quando dividimos o dia juliano por sete. É isso que fazemos no único cálculo que envolve o dia juliano.

```
* Programa para converter da "data juliana" para a data do calendario

SET TALK OFF
INPUT "ENTRAR A DATA JULIANA" TO datajulian
STORE 0 TO anobase
STORE anobase TO ano

* primeiramente temos que calcular o ano ( o calculo e >= que o ano real)

STORE INT(datajulian/365) TO anocalc
STORE INT(anocalc/4) - INT(anocalc/100) + INT(anocalc/400) TO anosbissex
DO CASE
   CASE anocalc-INT(anocalc/400)*400=0
        STORE T TO anobissex
   CASE anocalc-INT(anocalc/100)*100=0
        STORE F TO anobisssex
   CASE anocalc-INT(anocalc/4) *4=0
        STORE T TO anobissex
   OTHERWISE
        STORE F TO anobissex
ENDCASE
IF anobissex
   STORE datajulian - anosbissex - anocalc*365 TO diasrest
ELSE
   STORE datajulian - anosbissex - 1 - anocalc*365 TO diasrest
ENDIF
```

Programa 8. Conversão do dia juliano para uma data do calendário


```
* os dias restantes podem ser +,-, ou 0

DO WHILE diasrest < 0
   STORE anocalc-1 TO anocalc
   STORE INT(anocalc/4) - INT(anocalc/100) + INT(anocalc/400) TO anosbissex
   DO CASE
      CASE anocalc-INT(anocalc/400)*400=0
           STORE T TO anobissex
      CASE anocalc-INT(anocalc/100)*100=0
           STORE F TO anobissex
      CASE anocalc-INT(anocalc/4)*4=0
           STORE T TO anobissex
      OTHERWISE
           STORE F TO anobissex
   ENDCASE
   IF anobissex
      STORE datajulian - anosbissex - anocalc*365 TO diasrest
   ELSE
      STORE datajulian - anosbissex - 1 - anocalc*365 TO diasrest
   ENDIF
ENDDO
STORE "01 02 03 04 05 06 07 08 09 10 11 12" TO meses
IF anobissex
   STORE "31  29  31  30  31  30  31  31  30  31  30  31" TO diasnomes
   STORE "31  60  91  121 152 182 213 244 274 305 335 366" TO basedia
ELSE
   STORE "31  28  31  30  31  30  31  31  30  31  30  31" TO diasnomes
   STORE "31  59  90 120 151 181 212 243 273  304 334 365" TO basedia
ENDIF
STORE 1 TO indice, mes
DO WHILE indice <= 45
   IF diasrest > VAL(SUBSTR(diasnomes,indice,3))
      STORE mes + 1 TO mes
      STORE diasrest-VAL(SUBSTR(diasnomes,indice,2)) TO diasrest
   ENDIF
   STORE indice + 4 TO indice
ENDDO
STORE STR(mes,2) + "/" + STR(diasrest,2) TO data
SET DATE TO &data
STORE SUBSTR(DATE(),1,6) + STR(anocalc,4) TO data
? data
RETURN
```

Programa 8 (Continuação)

```
SET TALK OFF
INPUT "ENTRAR A DATA JULIANA" TO datajulian
STORE "0   1   2   3   4   5   6 " TO codigosem
STORE "Sex Sab Dom Seg Ter Qua Qui" TO diasdasem
STORE STR(d.tajulian - (INT(datajulian/7))*7,1) TO testedia
? $(diasdasem,@(testedia,codigosem),3)
RETURN
```

Programa 9. Calcula o dia da semana a partir do dia juliano

Capítulo Treze

TÉCNICAS DE INSERÇÃO DE DADOS

A inserção (entrada) de dados é a tarefa que consome mais tempo quando se opera um banco de dados, pois é primordialmente um processo manual. Felizmente o dBASE II oferece vários instrumentos que auxiliam nesta tarefa.

Os comandos básicos da linguagem de solicitação de dados para inserção de dados a partir do teclado são:

APPEND
INSERT
BROWSE
EDIT

Os comandos APPEND e INSERT criam novos registros de dados e permitem inserir dados nesses registros. Os comandos BROWSE e EDIT permitem modificar o conteúdo de registros já existentes.

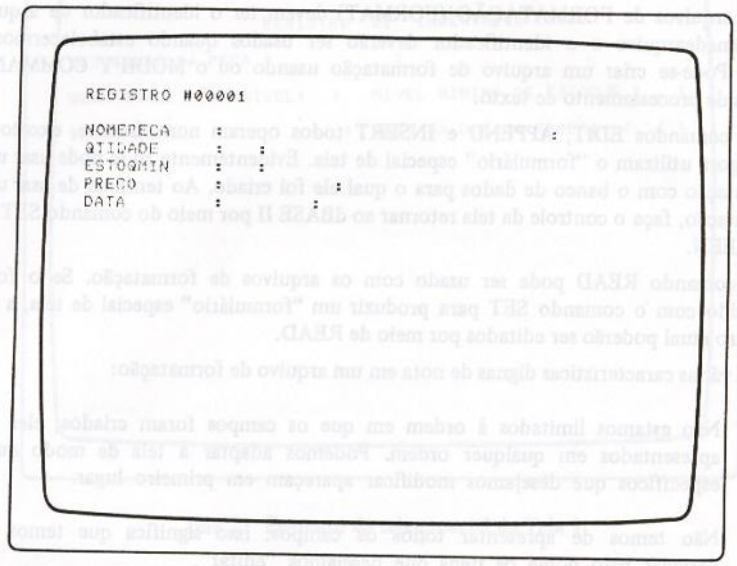
Esses comandos são extremamente potentes. Cada um deles utiliza um "formulário de tela cheia" em que o cursor fica sob controle do usuário, dando a este a mesma liberdade de inserir dados, retroceder e modificar entradas, como, por exemplo, formulários em papel usados pelas empresas.

Por mais potentes que sejam esses comandos, há várias coisas que podem ser feitas pelo programa dBASE II para melhorar esses recursos padrão. A inserção de dados pode ser melhorada por meio de:

Formulários de tela especiais
Inserção indireta de dados
Inserção em dois bancos de dados ao mesmo tempo
Verificação de erros
Edição através de um menu

Os comandos básicos do dBASE II usados para a programação de inserção de dados são:

APPEND BLANK	SET FORMAT TO
@ x,y SAY ... USING	ACCEPT
@ x,y GET ... PICTURE	INPUT
CLEAR GETS	REPLACE
READ	



Tela 1. Exemplo de uma tela padrão de inserções com os comandos APPEND e EDIT

Formulários de Tela Especiais

A Tela 1 mostra uma típica tela de inserção de dados, obtida com os comandos APPEND e EDIT. Essa tela utiliza o nomecampo como indicador para cada "espaço em branco" do campo. Os nomes de campo e o "espaço em branco" para o conteúdo de cada campo estão em uma coluna. Se você tiver mais de 22 campos, eles não serão apresentados todos ao mesmo tempo. O texto da tela subirá após ser fornecido o vigésimo segundo campo.

Podemos facilmente criar formulários especiais em tela cheia, usando os comandos @ x,y SAY e @ x,y GET. O comando @ x,y simplesmente significa: "na linha x, a partir da coluna y". A Tela 2 mostra uma versão "especial" de uma tela padrão; ela é criada pelo arquivo de formatação apresentado na Tela 3.

Nosso formulário especial será usado pelo dBASE II com os comandos EDIT, APPEND e INSERT.

Ao montar essa tela especial, primeiramente usamos um dos comandos SET para estabelecer o formato. Se o arquivo de formatação for denominado EXEMPLO.FMT, temos de estabelecer:

```
SET FORMAT TO EXEMPLO
APPEND
```

Os arquivos de FORMATAÇÃO (FORMAT) devem ter o identificador de arquivo .FMT, ou o nomedearquivo e o identificador deverão ser usados quando estabelecermos (SET) o formato. Pode-se criar um arquivo de formatação usando ou o MODIFY COMMAND ou um programa de processamento de texto.

Os comandos EDIT, APPEND e INSERT todos operam normalmente, exceto pelo fato de que agora utilizam o "formulário" especial de tela. Evidentemente só se pode usar um arquivo de formatação com o banco de dados para o qual ele foi criado. Ao terminar de usar um arquivo de formatação, faça o controle da tela retornar ao dBASE II por meio do comando SET FORMAT TO SCREEN.

O comando READ pode ser usado com os arquivos de formatação. Se o formato for estabelecido com o comando SET para produzir um "formulário" especial de tela, a tela criada e o registro atual poderão ser editados por meio de READ.

Há várias características dignas de nota em um arquivo de formatação:

1. Não estamos limitados à ordem em que os campos foram criados; eles podem ser apresentados em qualquer ordem. Podemos adaptar a tela de modo que os itens específicos que desejamos modificar apareçam em primeiro lugar.

2. Não temos de apresentar todos os campos. Isso significa que temos apenas de designar pelo nome os itens que desejamos "editar".

3. Podemos "editar" qualquer variável designada por meio do comando GET. Podemos digitar e editar variáveis de memória bem como campos.

4. Podemos colocar na tela qualquer texto descritivo que desejarmos.

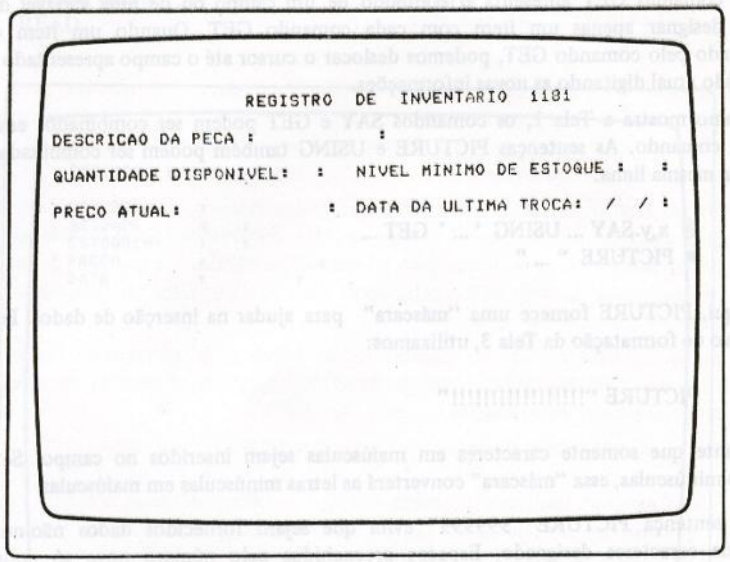
Usamos somente dois comandos do dBASE II em um arquivo FORMAT: @ x,y SAY e @ x,y GET. O comando GET pode ser ampliado por meio de PICTURE e o comando SAY por meio de USING. PICTURE E USING são variáveis e campos de caracteres de "máscara". Deve-se tomar cuidado quando são usados com variáveis ou campos numéricos.

A função de SAY é apresentar dados de caracteres do tipo texto descritivo, conteúdo de campo, variáveis de memória e combinações desses recursos. Quando apresentamos mais de um item com SAY, os itens devem ser concatenados com sinais de + .

```
@ 3,4 SAY 'DESCRIÇÃO DE PEÇA:
■ '+ NOMEPEÇA
```


Você pode apresentar diretamente o conteúdo de uma variável ou campo numérico se esse for o *único* item de um comando SAY.

```
@ 8,1 SAY CUSTO
```



Tela 2. Formato de tela especial da Tela 1

```
* este arquivo de formatacao e usado com o INV. DBF
@ 2,36  SAY "REGISTRO DE INVENTARIO: "+STR(#,5)
@ 5,2   SAY "NOME DA PECA " GET NOMEPECA PICTURE "!!!!!!!!!!"
@ 7,2   SAY "QUANTIDADE DISPONIVEL " GET QTIDADE
@ 7,37  SAY "NIVEL MINIMO DE ESTOQUE" GET ESTOQMIN
@ 9,2   SAY "PRECO ATUAL" GET PRECO
@ 9,37  SAY "DATA DA ULTIMA TROCA " GET DATA PICTURE "99/99/99"
```

Tela 3. Arquivo de formatação para criar o formato da Tela 2

Caso contrário, será necessário usar a função STR para apresentar o item numérico.

```
@ 8,1 SAY "CUSTO POR ATACADO
■ "+ STR(CUSTO,7,2)
```

O comando GET apresenta o conteúdo de um campo ou de uma variável de memória. Pode-se designar apenas um item com cada comando GET. Quando um item de dados é apresentado pelo comando GET, podemos deslocar o cursor até o campo apresentado e modificar o conteúdo atual digitando as novas informações.

Como mostra a Tela 1, os comandos SAY e GET podem ser combinados em uma única linha de comando. As sentenças PICTURE e USING também podem ser combinadas com SAY e GET na mesma linha.

```
@ x,y SAY ... USING ' ... ' GET ...
■ PICTURE " ... "
```

Aqui, PICTURE fornece uma "máscara" para ajudar na inserção de dados. Por exemplo, no arquivo de formatação da Tela 3, utilizamos:

```
PICTURE "!!!!!!!!!!!!!!!!!"
```

Isso garante que somente caracteres em maiúsculas sejam inseridos no campo. Se digitarmos em letras minúsculas, essa "máscara" converterá as letras minúsculas em maiúsculas.

A sentença PICTURE "999999" evita que sejam fornecidos dados não-numéricos ao campo de caracteres designado. Espaços preenchidos pelo número nove só podem receber números. Esse é um bom método de proteger entradas de números telefônicos.

A sentença PICTURE "99/99/99" fornece uma estrutura para datas. Os sinais de "/" são fornecidos para o usuário. Quando a data é fornecida, o dBASE II transpõe as barras.

A sentença USING assemelha-se a PICTURE, exceto que é usada para "mascarar" os itens apresentados pelo comando SAY.

O PC da IBM, assim como muitos outros computadores, exibe os itens designados por um GET em vídeo reverso. Os itens designados por SAY são apresentados normalmente. Alguns outros sistemas exibem GET normalmente e SAY em meia intensidade. Para eliminar o efeito de vídeo reverso ou de meia intensidade do dBASE II podemos usar o comando SET INTENSITY OFF. Para voltar à apresentação normal do vídeo, podemos usar SET INTENSITY ON.

O dBASE II usa sinais de dois pontos ":" para delimitar os itens designados por um GET. Isso permite visualizar um campo, mesmo que ele contenha apenas espaços em branco. Você pode eliminar os dois pontos com SET COLON OFF, bem como criar efeitos especiais. Por exemplo, se uma área de campo estiver sublinhada, provavelmente você preferirá "desligar" os dois pontos. Para restaurá-los basta usar o comando SET COLON ON.

A maioria das telas de CRT tem 24 linhas, cada uma com 80 espaços de caracteres. No dBASE II, essas linhas são numeradas de 0 a 23 e as colunas são numeradas de 0 a 79. Não é

aconselhável utilizar a Linha 0 (a linha superior) nem a Coluna 0 (a posição à extrema esquerda da tela).

Você também pode usar o comando NOTE (ou *) em um arquivo de formatação para documentar o arquivo. Por exemplo, pode-se utilizar esse comando para identificar com que arquivo do banco de dados o arquivo de formatação é usado (na Tela 3, nosso comando identificou INV como o arquivo de banco de dados em uso).

Os arquivos de formatação podem ser chamados de dentro de um programa ou então por meio do teclado. Quando usados juntamente com um programa (como nos programas para somar, editar e excluir registros, no Capítulo 5), são mais úteis durante a fase de desenvolvimento do programa, ou quando o mesmo formato de tela for usado em vários programas. Isso diminui ao máximo o código do programa durante o processo de desenvolvimento, uma vez que é mais fácil ver o que você está fazendo enquanto desenvolve o programa, se ele não estiver cheio de AT's e SAY's.

Os arquivos de FORMATAÇÃO são arquivos separados. A abertura de um arquivo leva uma fração de segundo – e muito mais nos sistemas MS/DOS com unidades de discos flexíveis. Em alguns sistemas, quando se usa o comando SET para estabelecer o formato, haverá uma hesitação perceptível. Nesse caso, você provavelmente irá querer passar o FORMATO para o programa depois que o programa estiver desenvolvido.

Inserção Indireta de Dados

Ao atualizar um inventário, o usuário tem de acrescentar a quantidade recebida à quantidade disponível e então fornecer o resultado da soma ao banco de dados. Queremos que o usuário simplesmente insira no computador a quantidade recebida e que o computador faça a soma e as modificações necessárias no banco de dados.

Utilizamos amplamente essa técnica nos exemplos da Parte II. Estas poucas linhas de código do dBASE II são um exemplo simples dessa técnica:

```
ACCEPT "ENTRAR O NÚMERO DA PEÇA" TO np
LOCATE FOR NÚMEROPEÇA = np
INPUT "QUANTIDADE RECEBIDA" TO qtidrec
REPLACE QTID WITH QTID + qtidrec
```

A entrada indireta reduz o esforço exigido do usuário e com isso as possibilidades de erro.

Inserção de Dados em Dois Bancos de Dados por Vez

Os dados podem ser fornecidos a dois bancos de dados diferentes ao mesmo tempo. Para isso, temos de estar usando um arquivo primário e um secundário, com cada um deles posicionado no registro que desejamos editar. A partir de um programa ou arquivo de formatação, os campos que serão editados em ambos os arquivos são exibidos com o uso do comando GET. Tecnicamente,

somente o arquivo escolhido poderá ser modificado, embora possa parecer que qualquer campo exibido possa ser editado.

Na tela, não haverá indicação de que os registros estão em arquivos diferentes. Como o dBASE II somente admite modificações para os campos selecionados, temos de "trapacear", registrando no arquivo não selecionado. Para isso, escolhemos o outro banco de dados e substituímos *qualquer* item do registro atual no próprio banco de dados. Isso fará com que os dados editados na tela sejam registrados nesse arquivo secundário. Essa técnica é muito mais fácil do que se tentar criar artifícios por meio de telas e ela permite efetivamente editar os dois arquivos em tela cheia. Entretanto, você estará limitado a 64 itens em qualquer edição em tela cheia.

Essa técnica é também diretamente aplicável aos bancos de dados em que um registro fica em dois arquivos contínuos por ser grande demais para um único. Suponhamos que temos uma escola primária. Cada registro de estudante requer 60 campos. Para acomodar o registro, utilizamos dois arquivos, ESCOLA1 e ESCOLA2. Os registros são acrescentados regularmente, de modo que o Registro 100 em ESCOLA2 seja continuação do Registro 100 em ESCOLA1. Os 60 campos e seus textos descritivos correspondentes se ajustam bem à tela. A tela é criada pelo arquivo de formatação ESCOLA.FMT.

Abaixo, um segmento de programa que ilustra essa operação:

```
SELECT PRIMARY
GO 100
SELECT SECONDARY
GO 100
SET FORMAT TO ESCOLA
READ
SELECT PRIMARY
REPLACE NOME WITH NOME
```

Verificação de Erros

A inserção de dados é a tarefa do banco de dados na qual mais se comete erros. Em grande parte dos casos, pode-se oferecer uma proteção contra erros na inserção. Entretanto, *não podemos* proteger contra a inserção de um valor legítimo – porém errado.

Pode-se usar a sentença PICTURE para se obter uma certa proteção, embora limitada, contra erros. Como descrevemos acima, PICTURE pode ser usada para verificar cada caractere digitado e para fazer com que um caractere esteja em maiúscula ou que seja um número. Esse recurso pode ser usado para evitar que se digite um número telefônico desta forma: 555-12A1. A característica de máscara de PICTURE pode ser usada para "preencher" partes de uma entrada – como sinais de barras (/) em uma entrada de data. A sentença PICTURE pode ser usada como uma "defesa de primeira linha" contra erros.

Para ilustrar a verificação de erros em um programa, examine a entrada dos nomes de estudantes em uma lista de chamada de escola. Os itens a serem fornecidos são NOME, SALA, CLASSE e REPROVADO. Os quatro são campos de caracteres. Podemos garantir que os três

últimos itens sejam, pelo menos, valores legítimos. O Programa 1 mostra como é feita a validação dessas entradas. Uma alternativa preferível é a apresentada pelo Programa 2.

Embora utilizável, o Programa 1 elimina a maior vantagem da entrada em tela cheia: a liberdade de mudar de idéia, de retroceder um item e modificar a entrada. O comando CLEAR GETS permite isso dentro de cada loop de verificação. Você precisa desse comando, pois só pode usar 64 chamadas por meio de GET antes de ter de usar ou um ERASE ou um CLEAR GETS. Com mais de 64 chamadas pelo comando GET, o dBASE II "entra em greve".

O método mais fácil de lidar com a verificação de erro para esse tipo de problema de entrada é apresentado pelo Programa 2. Observe que não há realmente muita diferença, exceto que, após a entrada inicial para "testar o lote" de dados fornecidos, inserimos os dados sob a forma de tela cheia verdadeira.

Embora tenha mais linhas de código, o Programa 2 é preferível ao programa original.

Telas de Entrada Dinâmicas

A entrada preparada dos dados, como a apresentada no Programa 1, deverá ser limitada a situações em que o "formulário" apresentado ao usuário varia de acordo com a resposta a uma pergunta específica. Neste caso, não estamos verificando a validade de uma resposta, mas buscando uma indicação quanto ao que fazer a seguir.

Para ilustrar esse ponto, examinamos um problema que muitas escolas americanas vêm enfrentando atualmente. Um número crescente de alunos provem de famílias que não falam inglês. As escolas têm de classificar esses alunos pela sua habilidade em utilizar o idioma inglês. Se a língua falada pela família do aluno é uma parte do registro do banco de dados do estudante, podemos ignorar as perguntas sobre a habilidade no uso do idioma cuja família fala inglês. O segmento de programa abaixo ilustra como a entrada de dados pode ser facilitada.

```
@ 3,1 SAY "NOME DO ALUNO" GET NOME
@ 5,1 SAY "SALA" GET SALA
@ 5,20 SAY "CLASSE" GET CLASSE
@ 5,40 SAY "REPROVADO (Y/N)? 'GET reprovado
@ 7,1 say 'A LÍNGUA FALADA PELA FAMÍLIA É O
■ INGLÊS (Y/n)?' GET pergunta
READ
IF pergunta $ 'Nn'
@ 7,0
@ 7,1 say 'LÍNGUA FALADA PELA FAMÍLIA'
■ GET LÍNGUA
@ 9,1 SAY 'NÍVEL DA HABILIDADE NO USO
■ DO 'INGLÊS  'GET HABILIDADE
ELSE
   REPLACE LÍNGUA WITH 'INGLÊS'
ENDIF
```

Neste exemplo, o usuário tem a oportunidade de inserir a informação sobre a língua falada pela família e a habilidade no uso do inglês somente se a resposta à pergunta "A língua falada pela família é o inglês?" for negativa.

```
@ 3,1 SAY "NOME DO ALUNO" GET NOME
READ
DO WHILE .NOT. SALA $"1,3,15,22,31,4A"
   @ 5,1  SAY "SALA " GET SALA
   READ
   CLEAR GETS
ENDDO
DO WHILE .NOT. CLASSE $"1,2,3,4,5,6,K"
   @ 5,20  SAY "CLASSE" GET CLASSE
   READ
   CLEAR GETS
ENDDO
DO WHILE .NOT. REPROVADO $"YnNn"
   @ 5,40  SAY "REPROVADO (Y/N)?" GET REPROVADO
   READ
   CLEAR GETS
ENDDO
```

Programa 1. Programa amostra com verificação de erro

```
@ 3,1  SAY "NOME DO ALUNO" GET NOME
@ 5,1  SAY "SALA" GET SALA
@ 5,20 SAY "CLASSE" GET CLASSE
@ 5,40 SAY "REPROVADO (Y/N)?" GET REPROVADO
READ
DO WHILE .NOT. SALA $"1,3,15,22,31,4A"
   @ 5,1  SAY "SALA" GET SALA
   READ
   CLEAR GETS
ENDDO
DO WHILE . NOT. CLASSE $"1,2,3,4,5,6,K"
   @ 5,20  SAY "CLASSE" GET CLASSE
   READ
   CLEAR GETS
ENDDO
DO WHILE .NOT. REPROVADO $"YnNn"
   @ 5,40  SAY "REPROVADO (Y/N)?" GET REPROVADO
   READ
   CLEAR GETS
ENDDO
```

Programa 2. Programa aperfeiçoado pela verificação de erros

Há vários casos em que a resposta a uma inserção de tela pode alterar um "formulário" para o usuário. Outro exemplo de se adaptar a inserção de tela de acordo com os dados que interessam seria com o objetivo de evitar solicitar a pessoas solteiras o nome de seus cônjuges.

Edição por Meio do Menu

Podemos fazer um menu dos itens apresentados para edição, o que pode ser conveniente quando a apresentação consiste em muitos itens e os itens a serem modificados variam consideravelmente de um registro para outro. A técnica também nos permite fazer maior uso dos códigos uma vez que trabalhamos com apenas um único item com cada READ (observe que essa técnica é acentuadamente diferente da colocação de um READ a cada entrada "preparada").

O Programa 3, baseado em nosso exemplo da escola, ilustra o uso de um menu de edição. Uma das vantagens de uma abordagem de seleção de menu para a edição está no fato de que o software de "edição" para cada entrada pode ser muito sofisticado. Podemos efetivamente buscar outros arquivos e fazer usos muito amplos da "codificação" por meio dessa abordagem. Essa técnica é extremamente eficiente, embora pouco utilizada.

O Uso de BROWSE

BROWSE não é um bom comando para se usar quando se desenvolve um programa para alguma outra pessoa. Para utilizá-lo eficientemente, é necessário estar familiarizado com as teclas de controle. Em todas as demais situações, o uso das teclas de controle na tela pode ser descrito sempre que necessário. Você não poderá fazer isso com BROWSE, a menos que o terminal do computador possua uma linha de status que possa ser substituída por uma linha do usuário (ver Capítulo 14).

Outro inconveniente do comando BROWSE é que a extensão do banco de dados que está sendo modificado por ele não pode ser facilmente controlada, como acontece com outras técnicas de edição. Igualmente, os títulos de campo usados com BROWSE não são adequados para campos pequenos.

Entretanto, BROWSE pode eliminar a necessidade de se escrever instruções detalhadas no dBASE II e muitas linhas de código. O Programa 4 ilustra a atualização das notas de leitura para grupos escolhidos em um banco de dados ESCOLA. Esse banco de dados está indexado por grupo (CLASSE+SALA+NOME). A criação de um recurso de entrada equivalente, com instruções detalhadas em dBASE II, iria exigir um número muito maior de linhas de código.

Todos os tópicos apresentados nesse capítulo podem ser complementos úteis para os comandos básicos de inserção de dados APPEND, INSERT, BROWSE e EDIT. Todos eles merecem ser experimentados. Entretanto, o sucesso máximo de um programa está na facilidade de uso para a pessoa que efetivamente fornecerá os dados. Deve-se examinar com cuidado as exigências de quem vai efetivamente digitar os dados.

```
STORE T TO loopentr
DO WHILE loopentr
   STORE " " TO item
   @ 3,1  SAY " 1 - NOME DO ALUNO " GET NOME
   @ 5,1  SAY " 2 - SALA " GET SALA
   @ 5,20 SAY " 3 - CLASSE " GET CLASSE
   @ 5,40 SAY " 4 - REPROVADO (Y/N)?" GET REPROVADO
   CLEAR GETS
   @ 20,1 SAY "Entrar o Numero do Item a se editado" GET ITEM
   READ
   DO CASE
      CASE item=" "
           STORE F TO loopentr
      CASE item="1"
           @ 3,6  SAY "NOME DO ALUNO " GET NOME
           READ
      CASE item="2"
           @ 5,6  SAY "SALA " GET SALA
           READ
           DO WHILE .NOT. SALA $'1,3,15,22,31,4A'
              @ 18,1  SAY "Numero de sala nao-existente"
              @ 5,6   SAY "SALA " GET SALA
              READ
              CLEAR GETS
           ENDDO
      CASE item="3"
           @ 5,25  SAY "CLASSE " GET CLASSE
           READ
           DO WHILE .NOT. CLASSE $'1,2,3,4,5,6,K'
              @ 18,1  SAY "Numero da classe invalido"
              @ 5,25  SAY "CLASSE " GET CLASSE
              READ
              CLEAR GETS
           ENDDO
      CASE item="4"
           @ 5,45  SAY "REPROVADO (Y/N)?" GET REPROVADO
           READ
           DO WHILE .NOT. REPROVADO $'YyNn'
              @ 18,1 "Responde sim (Y) ou Nao (N)"
              @ 5,45  SAY "REPROVADO (Y/N)?" GET REPROVADO
              READ
              CLEAR GETS
           ENDDO
      ENDCASE
      @ 18,0
      CLEAR GETS
ENDDO
```

Programa 3. "Edição de menu"


```
USE Escola INDEX Classe
STORE T TO loopentr
DO WHILE loopentr
   ERASE
   STORE "    " TO sala
   STORE "    " TO classe
   @ 1,1  SAY "ENTRAR O NUMERO DA SALA" GET m.sala
   READ NOUPDATE
   IF m.sala="    "
      STORE F TO loopentr
      LOOP
   ELSE
      @ 2,1  SAY "ENTRAR A CLASSE" GET m.classe
      READ NOUPDATE
   ENDIF
   STORE m.classe+m.sala TO grupo
   FIND &grupo
   COPY FIELDS NOME,SALA,CLASSE,NOTALEIT TO Temp WHILE classe+sala=grupo
   SELECT SECONDARY
   USE Temp
   BROWSE FIELDS NOEM,NOTALEIT
   USE
   SELECT PRIMARY
   UPDATE FROM Temp ON CLASSE+SALA+NOME RANDOM;
   REPLACE NOTALEIT WITH NOTALEIT
   DELETE FILE Temp
ENDDO
```

Programa 4. Uso do comando BROWSE

Capítulo Catorze

DEPURAÇÃO DE ERROS E EFEITOS ESPECIAIS

A depuração de erros é o processo de fazer o programa funcionar da forma que você deseja. Qualquer que seja seu nível de habilidade de programação, a depuração varia do trivial até o impossível. Ela é sempre frustrante.

Há dois tipos básicos de erros que se pode cometer ao criar um programa: *erros mecânicos* e *erros no desenvolvimento do programa*. Entre esses dois, os erros mecânicos são preferíveis. Geralmente o programa não "rodará" até que se tenha encontrado e corrigido todos os erros mecânicos. Os erros de desenvolvimento são mais traiçoeiros: o programa poderá "rodar", mas serão obtidos resultados incorretos. Pior que isso será quando se obtiver resultados incorretos somente ocasionalmente.

Erros Mecânicos

Exemplos de erros mecânicos são comandos escritos de modo errado, o erro na estrutura do comando e uma operação não-admissível. Os dois primeiros tipos são chamados erros de sintaxe.

O dBASE II detectará e diagnosticará, em parte, a maioria dos erros mecânicos possíveis. Você deverá completar o diagnóstico e corrigir o erro. Esse é o trabalho de depuração.

Um exemplo de erro mecânico será digitar DISPLAY como DISPAY. A reação do dBASE será interromper a execução do programa a exibir na tela:

```
.DISPAY
****VERBO DO COMANDO IRRECONHECÍVEL
DISPAY
CHAMADO DE B: EXEMPLO.PRG
CORRIGIR E TENTAR NOVAMENTE (Y/N)?
```


O dBASE II não elimina o programa nem tenta dar uma sugestão para correção do erro. Ele simplesmente interrompe a execução e dá ao usuário a oportunidade de corrigir o erro através do teclado. Isso lhe permite continuar até o erro seguinte. As correções por meio do teclado não são permanentes; você deverá retornar e corrigir o próprio programa.

A depuração da maioria desses tipos de erros é uma tarefa aborrecida – uma "multa cobrada em termos de tempo" que pagamos por nossa falta de cuidado. Ocasionalmente, teremos um pouco mais de trabalho do que simplesmente depurar um erro.

```
*** ERRO DE SINTAXE ***
                ?
DISPLAY FOR SALA = "6" .AND.
■ CLASSE = 5
CHAMADO DE B: EXEMPLO1.PRG
CHAMADO DE B: EXEMPLO.PRG
CORRIGIR E TENTAR NOVAMENTE Y/N?
```

No exemplo de erro acima, não fica claro o que está errado. O dBASE II nos informou em que programa está o erro. Ele também nos informou que o erro está à direita de ? colocado acima de S de SALA. Qual é o problema? Há campos ou variáveis de memória chamados SALA e CLASSE? Nós escrevemos um deles incorretamente? Os tipos de campo correspondem às comparações? No exemplo, podemos detectar o problema com o comando ? e a função TYPE.

A função TYPE fornece uma letra que indica o tipo de campo ou variável (C, N ou L) ou um "U", se a variável não existir.

```
? TYPE(SALA), TYPE(CLASSE)
C C
```

Nosso teste informa que os dois campos existem e que são ambos campos de caracteres. A sintaxe usada no comando para o conteúdo do campo classe indica um campo numérico – não há delimitadores. A falta de delimitadores (ou seu uso em combinação inadequada) é uma causa freqüente de erros de sintaxe. Podemos corrigir esse problema colocando os delimitadores no "5" de CLASSE.

Nem todas as mensagens de erro do dBASE II auxiliam tanto quanto os dois exemplos acima. Por exemplo, se tentarmos executar o seguinte fragmento de código do dBASE II:

```
SELECT PRIMARY
USE EXEMPLO
SELECT SECONDARY
USE EXEMPLO
```

O dBASE II não admite que se abra um arquivo como primário e ao mesmo tempo como secundário (isso é uma pena, pois infelizmente há também vários efeitos colaterais sérios). Se tentar a rotina acima a partir de um programa, você receberá a mensagem:

O ARQUIVO JÁ ESTÁ ABERTO

e será "expulso" do programa. Observe que você não recebe informação quanto aos nomes dos programas que estão sendo usados nem tem chance de corrigir o erro. Este erro, em especial, é o motivo por que você deve ficar atento e lembrar-se de encerrar todos os arquivos secundários – e às vezes até mesmo os primários – ao final de cada programa. Pelo menos é relativamente fácil encontrar o erro: você sabe o que deve procurar.

Há outros erros mecânicos que podem resultar numa expulsão do usuário para fora do programa, retornando ao teclado ou mesmo ao sistema operacional. Alguns desses erros podem – pelo menos, não coerentemente – não dar qualquer indicação quanto ao que aconteceu. Tudo o que se fica sabendo é que o programa não foi completado, e às vezes pode-se nem sequer ficar sabendo *disso*, se o programa for projetado para voltar ao teclado sem fornecer uma apresentação da situação.

Uma das causas mais comuns de comportamento estranho do computador ocorre quando se esquece a segunda parte de um par de comando: um ENDDO, ENDIF ou ENDCASE. Quando isso acontece, a estrutura lógica do programa pode ficar "torcida". Este tipo de erro raramente resultará em uma mensagem de erro que ajude a corrigi-lo.

Outra situação que pode fazer com que o dBASE II se descontrole sem alertar o usuário ocorre quando se utiliza variáveis de memória de modo incorreto. O dBASE II possui mensagens de erro para o caso de variáveis de memória *em demasia*, e de *fora de memória*, no caso das variáveis de memória.

Contudo, há situações em que o uso de um excesso de bytes de variáveis de memória ultrapassará o limite do sistema, especialmente se a criação e o abandono das variáveis não foram estruturados, de modo que todo o espaço de memória disponível ficará em um bloco. Embora o dBASE II possa nos indicar que há espaço suficiente, pode não haver espaço contíguo suficiente para a variável que se está tentando armazenar.

A primeira coisa a fazer quando se é sumariamente expulso de um programa será:

- Verificar o estado das variáveis de memória.
- Verificar se cada condição IF tem seu ENDIF.
- Verificar se cada DO WHILE... tem seu ENDDO.
- Verificar se cada DO CASE tem seu ENDCASE.

Um fenômeno interessante poderá ocorrer quando se é lançado para fora de um programa enquanto se está em um DO CASE. Poderá parecer que você perdeu o controle do teclado – o dBASE II não responderá a comandos provenientes dele. Quando isso acontece, tente o comando ENDCASE.

Erros no Desenvolvimento do Programa

Erros de desenvolvimento em seu programa ocorrem quando você fornece ao dBASE II uma série de comandos perfeitamente válidos sem erros mecânicos envolvidos. Mesmo assim, o programa não produz o resultado desejado.

Examinemos o programa simples, abaixo. É o esquema de um programa para imprimir listas de chamada de um banco de dados de ALUNOS, CLASSE, SALA etc. de uma escola.

```
USE ESCOLA
INDEX ON CLASSE + SALA + NOME TO
■ LISTA
DO WHILE .NOT. EOF
   DO CABEÇALH
   STORE CLASSE + SALA TO grupo
   SET MARGIN TO 30
   DISPLAY OFF NOME WHILE
      CLASSE + SALA = grupo
   SET MARGIN TO 0
ENDDO
```

Quando executamos esse programa não conseguimos qualquer resultado. Apenas um indicador de ponto. Não obtemos as listas de chamada. *Não* há erro mecânico. O dBASE II fez exatamente o que lhe dissemos que fizesse. Quando o dBASE II acaba de usar o comando INDEX, o banco de dados está no último registro *e* o flag de fim-de-arquivo está estabelecido. A instrução DO WHILE .NOT. EOF fez exatamente o que pedimos que fizesse, e não conseguimos as listas impressas. Neste caso, precisamos usar o comando GO TOP antes de entrar o loop DO.

Na maioria dos casos, a depuração consistirá em observar cuidadosamente o que acontece a seu programa e em usar um pouco de bom senso. A maioria dos erros é simples e direta. Quando as coisas se complicam, há uma série de soluções que podemos tentar.

O Que se Poderá Tentar

Fazer as próprias mensagens de diagnóstico. Neste caso, pode-se apresentar o conteúdo de uma variável de memória ou uma série de *strings* de texto na tela. Aqui modificamos temporariamente nosso programa para providenciar esse tipo de saída em tela – o que será especialmente útil se formos expulsos sem aviso.

Isso pode ser usado para controlar o conteúdo de um acumulador em um loop DO. Suponhamos que você não esteja obtendo os subtotais corretos durante a execução de um determinado processo. Supostamente, o subtotal deveria ser impresso no final de um loop. Você pode controlar o processo à medida que ele passa pelo loop e tentar ver onde ele não dá certo pela apresentação do acumulador a cada estágio.

Acrescentar um RETURN. O comando RETURN é usado para encerrar um programa. Podemos colocar quanto texto desejar após o RETURN. O comando RETURN permitirá parar o programa no ponto que escolhermos. Agora podemos examinar o estado dos bancos de dados e das variáveis de memória. Podemos acrescentar um RETURN, executar o programa, examinar o resultado, e MOVER o RETURN até conseguir localizar o problema.

Usar os instrumentos padrão de depuração. O dBASE II fornece um conjunto de instrumentos (além das mensagens de erro) que se podem utilizar para ajudar a detectar o problema. São eles:

```
SET STEP ON
SET ECHO ON
SET TALK ON
SET DEBUG ON
```

O comando SET STEP ON/OFF permite acompanhar o programa passo a passo, sob controle manual, uma linha por vez. Pode-se interpor comandos por meio do teclado a cada passo e encerrar (SET STEP OFF) em qualquer estágio.

O comando SET ECHO ON/OFF apresenta a linha de comando no terminal CRT. SET TALK ON/OFF apresenta a resposta do dBASE II à maioria dos comandos. Esses comandos são mais eficazes na depuração quando usados juntos. Eles permitem passar pelo programa passo a passo e examinar cada comando e sua resposta, um por vez. Evidentemente, se você tentar realizar a passagem por um loop contador, esse processo se tornará muito entediante. Lembre-se, você pode inserir esses itens como linhas de comando em seu programa durante um processo de depuração.

O comando SET DEBUG ON/OFF tem a finalidade de ajudar nos processos de depuração que envolvem operações com tela cheia. SET DEBUG ON passa a saída dos comandos ECHO e STEP para a impressora em vez da tela, o que evita que a tela fique abarrotada.

Tudo somado, a depuração é um processo que assusta mais do que deve. A maioria de nossos problemas será mecânico. Entre estes, a maioria é resultado de erros na digitação, escrita incorreta e de falta de delimitadores.

O tempo gasto na depuração pode ser enormemente reduzido pelo desenvolvimento dos programas um pouco por vez, com o uso de pequenos módulos. Tente usar um programa de alto nível com uma estrutura próxima a esta:

```
DO PARTE1
DO PARTE2
DO PARTE3
DO PARTE4
  etc.
```

Se utilizarmos essa abordagem em pequenos módulos e desenvolver seu programa por estágios, estaremos menos propensos a ter problemas sérios de depuração. Também pode ser interessante manter os módulos do programa em uma tela. Isso poderá levar o programa a ser executado mais lentamente a princípio, já que estaremos abrindo e fechando vários arquivos de programa. Mas uma vez que tudo esteja funcionando e todos os módulos estejam unificados, teremos um programa de execução rápida e sem problemas.

Faça o Hardware Ajudá-lo

Todos os sistemas de bancos de dados são baseados necessariamente em discos: os arquivos de dados são armazenados em disco. Quando acessamos dados desses arquivos em disco, a cabeça do disco se move para onde os dados estão localizados no disco.

Os registros de banco de dados do dBASE II são armazenados em registros em discos. O tamanho do registro em disco é determinado pelo sistema operacional. O tamanho do registro do dBASE II é estabelecido com a criação do banco de dados. Pode haver muitos registros de banco de dados por registro em disco para cada registro do banco de dados.

Os dois mais importantes sistemas operacionais usados com o dBASE II, o MS/DOS (PC-DOS) e o CP/M, utilizam esquemas de armazenamento em disco ligeiramente diferentes. Entretanto, cada um deles atribui espaço em disco para um arquivo em quantidades fixas. Essa quantidade é relativamente pequena. Assim, temos no disco uns poucos registros provenientes do banco de dados A seguidos por uns poucos registros do banco de dados B etc. Essa situação é descrita pela Figura 1.

Quando "lemos" o banco de dados para evitar desvio do disco, queremos "agrupar" todos os seus registros em um conjunto de registros contíguos em disco. Isso permite o mínimo possível de movimento da cabeça. Para agrupar esses registros, coloque seu banco de dados mais freqüentemente usado na trilha externa, ao montar um novo sistema de banco de dados. Então, coloque tantos registros em banco do dBASE II quantos serão utilizados no banco de dados completo. Se for utilizar índices com esse banco de dados, crie-os agora. Não se preocupe com o fato de que não terão qualquer significado.

Com isso, há um banco de dados modelo nas trilhas exteriores. Como todos os registros do dBASE II são contíguos, os arquivos de índice modelo estarão em quantidades contíguas movendo-se para dentro, em direção ao centro do disco. Agora utilizamos o comando MODIFY STRUCTURE para "limpar" todos os registros em branco e o comando REINDEX. Embora tenhamos agora um banco de dados, sem registros, e arquivos de índice vazios, o sistema operacional irá considerar como se o dBASE II estivesse efetivamente usando todos esses registros vazios. O dBASE II *não* devolve o espaço em disco ao sistema operacional. Quando começarmos a acrescentar nossos registros verdadeiros ao banco de dados, eles irão para o espaço anteriormente ocupado pelos registros em branco.

Esse esquema limita o âmbito do movimento da cabeça de disco e nos dá uma "melhor" utilização dela. Esse cuidado com a cabeça de disco pode ocasionar uma diferença enorme na velocidade do processamento.

Para exemplificar esse ponto, construímos um esquema em que incluímos um banco de dados nele mesmo. A primeira parte do banco de dados foi posicionada na borda externa do disco. A segunda, foi posicionada na borda mais interna. Então, indexamos o banco de dados (o índice estava em uma área anteriormente bloqueada, no centro do disco). A "leitura" do arquivo – usado com o índice – levou quase uma hora. Um registro sim, um registro não, a cabeça de disco tinha de ser reposicionada de acordo com o tamanho do disco. A leitura do mesmo banco de dados, sem o índice, levou apenas três minutos. A reordenação física do banco de dados como acima descrito nos permitiu ler o banco de dados indexado em um pouco mais de quatro minutos.

Se o banco de dados tiver de ser usado normalmente com um ou mais índices, podemos "acelerar" as leituras de disco primeiramente classificando o arquivo pelo índice mais freqüentemente usado. Isso faz com que as posições físicas e lógicas se combinem. Se combinarmos nossa operação de "agrupamento" de arquivo com a operação de classificação, teremos obtido o máximo possível em desempenho nas operações em disco.

A propósito, o mesmo conceito vale para o próprio dBASE II. O dBASE II (versão 2.4) é composto por três arquivos: o dBASE.COM, o dBASEOVR.COM e o dBASEMSG.TXT. Se possível, coloque o dBASE.COM no interior do disco. Quando usamos o dBASE II, nós o carregamos e é tudo o que fazemos com ele, o que não acontece com os outros dois arquivos. Um grande número de comandos como SUM e DO CASE é "chamado" de dBASEOVR.COM. Isso significa que desejamos que esse arquivo seja acessível tanto quanto possível ao disco do sistema, mas não tão acessível quanto qualquer um dos arquivos de banco de dados.

O dBASE II permite trabalhar sem se ter conhecimento de como opera o sistema em disco. Embora isso seja bom, o conhecimento de como o sistema de disco armazena e recupera os dados pode dar a você a oportunidade de "acelerar" substancialmente o processamento do dBASE II, sem qualquer artifício de codificação e sem gastar um tostão com discos "rápidos" ou discos de RAM.

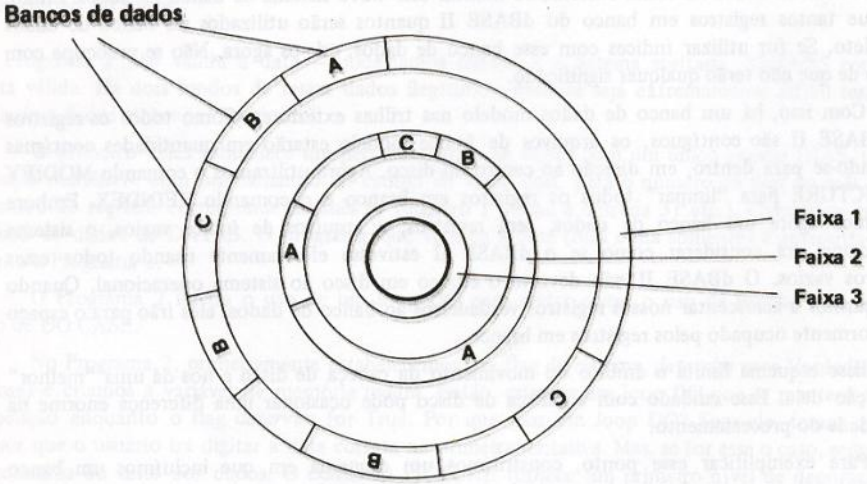


Figura 1. Armazenamento de arquivos em disco

Compreender como funciona um sistema de manipulação de disco exigirá um pouco de tempo – mas valerá a pena.

Efeitos Especiais

As telas exibidas pelos terminais CRT e os relatórios impressos são o que os outros vêem como o resultado de nosso trabalho de codificação. O dBASE II não utiliza quaisquer recursos especiais da impressora e somente alguns dos recursos do CRT. No PC da IBM, ele normalmente utilizará a apresentação com vídeo normal e reverso. Se tiver um monitor em cores, você poderá usar o dBASE II em cores.

Podemos ampliar bastante nosso controle dos dispositivos periféricos e com pouco esforço por meio da função CHR, que é relativamente semelhante à função CHR$ do BASIC. O dBASE II normalmente filtra quaisquer caracteres de controle (os caracteres não imprimíveis ASCII) que poderíamos tentar inserir para os campos ou para variáveis de memória. A função CHR permite incorporar caracteres de controle a variáveis.

O controle da tela é muito diferente no PC da IBM e na maioria dos computadores CP/M de 8 bits. No PC, os atributos de tela podem ser controlados por meio do comando SET COLOR TO; com a maioria dos computadores de 8 bits será preciso utilizar a função CHR. Tanto no PC da IBM como nos computadores com CP/M de 8 bits, poderemos utilizar a função CHR para controlar atributos especiais *da impressora.*

A Impressora

Muitas impressoras possuem um grande número de atributos especiais que podem ser chamados quer por interruptores quer pelo uso de códigos especiais de controle em seu software. São exemplos:

- tabulações horizontais
- tabulações verticais
- espacejamento de linha
- densidade de impressão
- seleção da fonte
- seleção de cores
- sublinhado

O equivalente das tabulações horizontal e vertical pode ser obtido por meio do comando a x,y SAY do dBASE II. Às vezes, ao usar essas "tabulações" em cabeçalhos, pretendemos trocar de fonte ou mudar os tamanhos de impressão. Com isso, poderemos obter uma saída impressa fora do padrão. E podemos dar à saída impressa comum do computador uma aparência muito mais atraente.

Por exemplo, suponhamos que queremos que o nome de nossa empresa seja impresso em um tipo maior que o restante do texto. Muitas impressoras oferecem software de 10, 12 e 16 caracteres por polegadas. Muitas delas oferecem um modo de impressão realçada em que cada caractere é impresso em tamanho duplo: na verdade, 5, 6, 8, 10, 12 e 16 cpi.

Queremos que nossa impressora produza um cabeçalho de página com a seguinte aparência

DISTRIBUIDORA FERREIRA
RUA ITATIAIA, 10150
SÃO PAULO – SP
90230

A maioria das impressoras utiliza seqüências de controle de dois ou três bytes. Estes geralmente se iniciam pela tecla <ESCAPE>. A impressora IDS 560, por exemplo, utiliza <ESCAPE> + Control A para selecionar o modo de impressão realçada e <ESCAPE> + Control B para selecionar o modo de impressão normal. O modo de 16 cpi é selecionado por <ESCAPE> + Control – e o modo de 12 cpi por <ESCAPE> + Control | .

Para imprimir a primeira linha do cabeçalho a oito caracteres por polegada, primeiramente selecionamos a densidade de impressão de 16 cpi. Então, selecionamos o modo de realce. Uma vez impressa a linha, temos que voltar ao modo normal e selecionar a densidade de impressão de 12 cpi. As tabelas de código ASCII para relacionar os símbolos ASCII aos códigos decimal, binário e hexadecimal são encontradas no Apêndice D. Em nosso exemplo, em especial, as funções CHR são mostradas na tabela abaixo. Com relação ao Apêndice D, verificamos que a função ESCAPE em ASCII corresponde ao valor decimal 27. Para emitir <ESCAPE>, a partir do dBASE II, colocamos o valor decimal para um Control A (01) em uma função CHR.

modo de realce	ESCAPE + ^	A	chr(27) + chr(01)
modo normal	ESCAPE + ^	B	chr(27) + chr(02)
16 cpi	ESCAPE + ^	_	chr(27) + chr(31)
12 cpi	ESCAPE + ^	|	chr(27) + chr(30)

Podemos escrever o modo para imprimir nosso cabeçalho da seguinte forma:

```
@ 5,25 SAY CHR(27) + CHR(31) + CHR(27)
■ + CHR(01) + "DISTRIBUIDORA FERREIRA"
@ 6,23 SAY CHR(27) + CHR(30) + CHR(27)
■ + CHR(02) + 'RUA ITATIAIA, 10150'
@ 7,27 SAY 'SÃO PAULO – SP'
@ 8,32 SAY '90230'
```

Cada utilização da função CHR envia um byte para a impressora. Nosso título especial exigia mais bytes que a impressão normal. Talvez sejam necessárias algumas experimentações para se perceber como estabelecer as posições de início de linha.

Uma outra abordagem desse mesmo problema seria:

```
STORE CHR(27) TO esc
STORE esc + CHR(31) TO p16cpi
STORE esc + CHR(30) TO p12cpi
STORE esc + CHR(01) TO realce
STORE esc + CHR(02) TO normal
@ 5,25 SAY p16cpi + realce + "DISTRIBUIDORA
■ FERREIRA"
@ 6,23 SAY p12cpi + NORMAL +  RUA ITATIAIA,
■ 10150'
@ 7,27 SAY 'SÃO PAULO – SP'
@ 8,32 SAY '90230'
```


Esse fragmento de comando utiliza maior quantidade de código, mas é mais claro. Se tivéssemos que usar muita impressão como funções especiais seria melhor usarmos essa abordagem, especialmente porque ela nos permite saber o que está acontecendo sem um manual.

Nossa impressora específica permitirá enviar códigos de controle em BASIC e, assim, ao dBASE II. Por exemplo, para estabelecer as tabulações horizontais (em BASIC) em duas, quatro e seis polegadas, utilizamos:

```
10 PRINT CHR$(27);"F,240,480,720,$"
```

No dBASE II, esse comando do BASIC se torna:

```
? CHR(27) + 'F,240,480,720,$'
```

O "F" é o código de controle da impressora para o estabelecimento das tabulações horizontais. O sinal do $ é um finalizador. A impressora procurará por esse caractere para encerrar a seqüência. O F somente é reconhecido como um caractere de controle quando é o primeiro caractere após um ESCAPE CHR(27).

Nem todas as impressoras permitirão o envio de seqüências de controle em código ASCII como no exemplo acima. As que não o fazem exigirão que se envie toda a seqüência como uma seqüência de funções CHR.

A função CHR é a chave para se obter o máximo de sua impressora. Ela permite adaptar e personalizar a saída impressa. As impressoras mais recentes apresentam uma grande variedade de recursos atraentes que permanecerão em grande parte não utilizados enquanto você não se familiarizar com o uso de CHR.

Telas Criativas

A maioria dos terminais de computadores são códigos de controle semelhantes aos usados pelas impressoras. O PC da IBM utiliza um monitor com um controle de monitor que é parte do PC. Examinaremos primeiramente o terminal e depois o monitor, devido à semelhança entre o terminal e a impressora.

Como acontece com a impressora, CHR é a chave de efeitos especiais em seu terminal. Entre os efeitos especiais disponíveis na maioria dos terminais estão:

- Vídeo normal
- Vídeo de meia intensidade
- Vídeo reverso
- Sublinhado
- Pisca-pisca

- Desenho de linhas
- Gráficos

Você pode chamar esses efeitos especiais por meio da função CHR, exatamente da mesma forma que faz com a impressora. O dBASE II permite utilizar alguns deles de forma limitada, sem necessidade de recorrer à função CHR. Entretanto, a função CHR possibilita liberdade completa de utilização desses atributos, de acordo com a capacidade de seu terminal.

Como acontece com impressoras, os manuais do equipamento que explicam como utilizar esses efeitos são um tanto irregulares. Há uns dois anos, quase não eram utilizados por ninguém. Isso não mais acontece. A maioria dos manuais de equipamentos melhorou acentuadamente. Ainda assim, se você nunca tentou estabelecer atributos, será necessário um pouco de experimentação antes de obter bons resultados.

Para demonstrar a utilização dos atributos de tela (efeitos especiais), utilizaremos o Terminal Televídeo 950. É um terminal muito usado e já existe há algum tempo. Muitos dos atributos de tela do Televídeo exigem um espaço de tela. Alguns dos mais recentes terminais manipulam atributos de tela em uma memória separada no terminal e não exigem espaço na tela.

Para exemplificar, vamos modificar o programa para criar nosso menu principal do sistema de contabilidade – o Programa 1 do Capítulo 4. Queremos que a tela tenha a aparência da Tela 1.

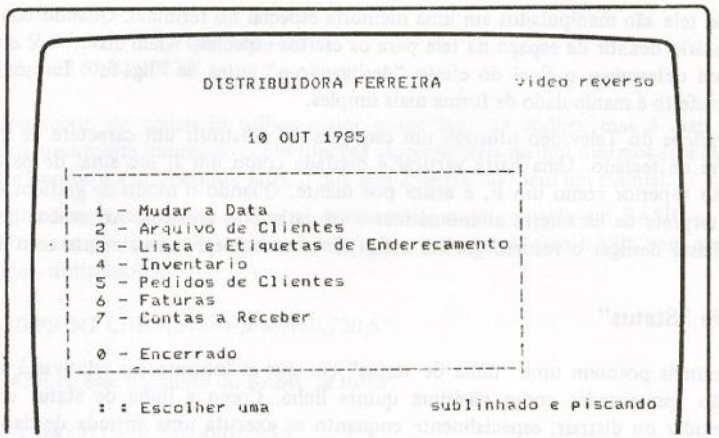
Nesse exemplo, queremos o título DISTRIBUIDORA FERREIRA destacado em vídeo reverso, a data e o "Escolha Uma" sublinhados e a área que contém o menu contida em um quadro. O quadro é criado por meio de linhas regulares; o Televídeo 950 pode executar esse tipo de desenho de linha. Além de sublinhado, queremos que "Escolha Uma" esteja intermitente na tela.

Nosso programa modificado – a parte que interessa – é mostrado como o Programa 1. Para clareza, armazenamos as funções CHR apropriadas nas variáveis de memória *reverso, normal* e *sublinhado*. Podemos usar as funções CHR diretamente como no primeiro exemplo de impressão.

Antes de inserir o loop DO no programa, armazenamos as seqüências apropriadas de controle em variáveis de memória com nomes descritivos. A variável normal é a característica de apresentação padrão do terminal do Televídeo. O termo *grafon* significa modo gráfico ligado (observe que utilizamos o comando SET INTENSITY OFF do dBASE II. Estamos tentando controlar a tela e não queremos entrar em conflito com os próprios comandos do dBASE II. Os gráficos do Televídeo não parecem funcionar como pretendemos quando o comando INTENSITY está ligado (ON). Se tiver problemas com os controles de tela, use o comando SET INTENSITY OFF e tente novamente).

Em muitos dos terminais como o Televídeo 950, o atributo de tela é válido desde a posição atual até o final da tela, ou até o encontro de um outro atributo. Se quiséssemos passar para vídeo reverso, imprimir o título, e então sair de vídeo reverso com *normal*, veríamos um grande brilho enquanto o vídeo reverso se mantivesse até o final da tela antes de ser interrompido pelo *normal* no final da *string* de texto. Para evitar esse brilho, colocamos em normal no final da *string* de texto antes de chamarmos o atributo especial de tela.

Observe que esses atributos ocupam espaço na tela. Eles exigem que "desistamos" de um espaço de caractere no final do atributo especial. Em alguns dos terminais mais recentes, os



Tela 1. A tela do menu principal com efeitos especiais

```
STORE CHR (27) TO esc
STORE esc+"G0" TO normal
STORE esc+"G4" TO reverso
STORE esc+"G8" TO sublinhado
STORE esc+"G:" TO piscapisca
STORE esc+"$"  TO grafon
STORE esc+"%"  TO grafoff

DO WHILE T
   ERASE
   SET INTENSITY OFF
   @ 5,32  SAY "DISTRIBUIDORA FERREIRA"+ normal
   @ 5,29  SAY reverso
   @ 7,35  SAY "10 OUT 1985"+ normal
   @ 7,32  SAY sublinhado
   @ 10,25 SAY "1 - Mudar a Data"
   @ 11,25 SAY "2 - Arquivo de Clientes"
   @ 12,25 SAY "3 - Lista e Etiquetas de Enderecamento"
   @ 13,25 SAY "4 - Inventario"
   @ 14,25 SAY "5 - Pedidos de Venda"
   @ 15,25 SAY "6 - Faturas"
   @ 16,25 SAY "7 - Contas a Receber"
   @ 19,25 SAY "0 - Encerrado"
   @ 21,31 SAY "Escolha uma"+ normal
   @ 21,28 SAY piscapisca
   @ 9,20  SAY grafon+ "FKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKG"
   @ 20,21 SAY "EKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKH"
   STORE 10 TO linha
   DO WHILE linha <= 20
      @ linha,20 SAY "J"
      @ linha,71 SAY "J"
      STORE linha+1 TO linha
   ENDDO
   ?? grafoff
   STORE "  " TO escolha
   @ 21,24 GET escolha
   READ NOUPDATE
   SET INTENSITY ON
ENDDO
```

Programa 1. Modificação da tela do menu, mostrando efeitos especiais

atributos de tela são manipulados em uma memória especial no terminal. Quando isso acontece, não é necessário desistir de espaço na tela para os efeitos especiais. Além disso, você muitas vezes não precisará determinar o final do efeito "desligando-o" antes de "ligá-lo". Em muitos desses terminais, o efeito é manipulado de forma mais simples.

Os gráficos do Televídeo utilizam um esquema de substituir um caractere de gráfico por um caractere de teclado. Uma barra vertical é digitada como um J, um sinal de parênteses no canto direito superior como um F, e assim por diante. Quando o modo de gráficos é ligado, o terminal interpreta os caracteres alfanuméricos e os caracteres gráficos. Ao acabar de desenhar o quadro, basta desligar o recurso gráfico a fim de se comunicar normalmente com o terminal.

A Linha de "Status"

Muitos terminais possuem uma "linha de status" em que as informações relativas à situação do terminal são apresentadas como vigésima quinta linha. Como a linha de status do terminal pode incomodar ou distrair, especialmente enquanto se executa uma entrada de dados padrão, a maioria dos terminais permite que você a substitua por uma "linha do usuário" de sua escolha. No Televídeo, por exemplo, podemos fazer isso com:

```
? CHR(27) + 'f' "DISTRIBUIDORA
■ FERREIRA" + CHR(13)
```

o que fará a substituição da linha de status padrão do Televídeo pela linha do usuário, DISTRIBUIDORA FERREIRA. Esta linha do usuário permanecerá aí até a modificarmos ou desligarmos o terminal. Ela é fixa na tela independentemente das operações na tela, inclusive ERASE.

Podemos aproveitar isso para imprimir títulos de coluna na linha de status quando estamos fazendo listas. Dessa forma, não precisamos nos preocupar em lembrar o que conterão as colunas ou em reimprimir o título da coluna periodicamente na parte superior. As informações parecerão "provir" dos títulos das colunas.

Teclas de Função Especiais

Muitos dos terminais possuem teclas de funções programáveis. O PC da IBM e muitos outros de seu tipo possuem teclas de funções especiais carregadas automaticamente quando o dBASE II é carregado. A função atribuída a cada tecla pode ser facilmente modificada por meio de SET Fx TO <comando>. Infelizmente, os terminais não são todos idênticos quanto à forma de manipulação das teclas de função. Você pode usar a função CHR para "carregar" teclas programáveis de funções especiais em muitos terminais.

O Televídeo 950, por exemplo, possui 11 teclas de funções especiais que são programáveis. Cada tecla pode ser programada com duas funções especiais (por exemplo F1 e shift F1).

Para ilustrar essa questão, vamos programar a tecla de função F1 para emitir um Control W, usado com shift, e um Control C, usado com shift. O "endereço" da tecla F1 é 1, sem a tecla shift, e <, quando com a tecla shift.

A forma do comando de controle é a seguinte:

```
ESC| p1 p2 mensagem ^ Y
```

O sinal ^ Y é o finalizador – necessário, devido à mensagem de tamanho variável.

p1 é o endereço da tecla
p2 é um código 1, 2 ou 3 em que 1 é enviado ao computador
2 é enviado à tela
3 é enviado a ambos

Vamos enviar o código ao computador p2 = 1.

Control W é o valor ASCII 23, em decimal
Control Q é o valor ASCII 17, em decimal

A tecla de controle não acompanhada por shift é carregada por meio de:

```
? CHR(27) + '| 1' + CHR(23) + CHR(25)
```

A tecla de controle utilizada com shift é carregada por meio de:

```
? CHR(27) + '|<' + CHR(17) + CHR(25)
```

Como você pode notar, é possível exercer controle sobre várias das funções do terminal com o uso da função CHR. A parte "difícil" é geralmente a tradução do manual para o terminal em termos que o usuário possa operar. Às vezes, ele terá de trabalhar por um processo de tentativa e erro; isso não poderá ocasionar qualquer dano – pelo menos não permanente. Sempre que desligar o terminal quaisquer erros cometidos serão apagados.

O Monitor e o Teclado do PC da IBM

Com ele, podemos executar muitas das mesmas operações com as funções CHR que vínhamos realizando com o uso dos comandos SET. As teclas de função no PC são automaticamente carregadas quando o dBASE II é carregado. Se você não gostar da seleção padrão, poderá modificá-la. Por exemplo, se quiser que F1 emita o comando DISPLAY, você poderá:

```
SET F1 TO "DISPLAY;"
```

O item que é acrescentado deve estar delimitado por aspas. O sinal de ponto e vírgula emite uma nova linha e um retorno de carro. Não se pode (pelo menos, como na presente edição) usar a função CHR para colocar caracteres de controle nas teclas de função do IBM. Isso não é muito grave já que as teclas de seta funcionam de forma separada, o que nem sempre acontece com muitos dos terminais de computadores.

Pode-se fazer o monitor produzir os mesmos tipos de efeitos que nós executamos com o uso do Televídeo. Com o PC, *não* usamos a função CHR. Utilizamos o comando SET COLOR TO. Esse comando controla os atributos de tela para o monitor em cores *e* para o monitor monocromático. No PC, são usados efetivamente dois bytes para cada posição de caractere na tela. Um byte é o atributo, o outro é o caractere.

Há poucos painéis de cores e monitores disponíveis para o PC. Eles não parecem ser exatamente idênticos quanto à atribuição de bits para atributos de efeitos especiais. Isso significa que você deverá fazer alguma experimentação para determinar os valores de controle da tela.

A forma do comando SET COLOR é:

SET COLOR TO <primeiroplano,fundo>

onde o primeiro plano e o fundo são números.

```
* determinar atributos de tela
SET TALK OFF
SET INTENSITY OFF
STORE 0 TO x,y,cor
ERASE
DO WHILE y < 76
   DO WHILE x < 16
      SET COLOR TO cor
      @ x,y SAY " "+STR(cor,3)+" "
      STORE cor + 1 to cor
      STORE  x + 1 TO x
   ENDDO
   STORE 0 TO x
   STORE y+5 TO y
ENDDO
SET INTENSITY ON
RETURN
```

Programa 2. Programa para determinar atributos de tela para o monitor

O valor para o fundo pode ser omitido. O Programa 2 é um programa exemplo para determinar atributos de cor (ou de apresentação monocrática) na tela.

Esse programa é relativamente simples e desenha barras verticais na tela. Cada combinação de cor (ou de efeito monocromático) será apresentada na tela. O número será o número determinado por SET COLOR, que se desejar para um efeito específico. Notaremos que no caso de números acima de 127, os próprios números estarão intermitentes.

. A Figura 2 mostra o padrão de cor produzido por esse programa em um terminal em cores Amdek. A Figura 3 mostra o padrão obtido com um monitor monocromático. O Programa 3 é um programa para criar versões em cores de nosso programa de Menu Principal do Capítulo 4.

Nesse programa primeiramente criamos um fundo azul. Isso é criado pelo loop que apaga a tela, uma linha por vez. Com a cor estabelecida como 1,23 (branca ou azul) o comando @ x,0 pinta um fundo em azul. O passo seguinte consiste em imprimir o cabeçalho da página e a data (letras brancas em um fundo azul). Agora modificamos a determinação das cores para 1,159 – ainda com letras em branco sobre o azul, mas com as letras em branco intermitentes.

A seguir, modificamos a determinação das cores para 1,112: preto sobre o branco. Observe que os espaços fornecerão o efeito de fundo de um quadrado branco no centro da tela. Finalmente, imprimimos a seleção de menu em letras pretas nesse quadrado.

A função CHR pode ser usada aqui para chamar os símbolos especiais de gráficos. Você precisará fazer alguma experimentação já que há diferenças entre os sistemas. Por exemplo, com o Princeton Graphics System, o armazenamento de um CHR(4) em D produzirá um losango toda vez que apresentarmos D.

Um pouco de esforço para conhecer os recursos de seu equipamento poderá resultar em efeitos atraentes e fantásticos com quase todos os sistemas de computador. A função CHR é usada com a impressora e com terminais que não o do PC. Ela também pode ser usada para produção de símbolos gráficos nos monitores do PC da IBM. O comando SET COLOR pode ser usado com o PC exatamente da mesma forma que a função CHR era usada para apresentações especiais nos terminais para o PC.

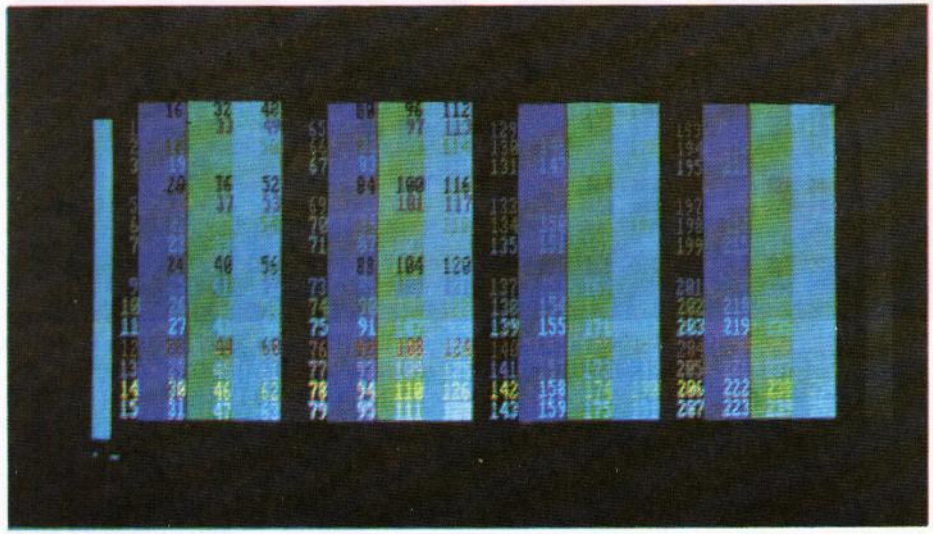


Figura 2. Padrão de cores produzido pelo Programa 2

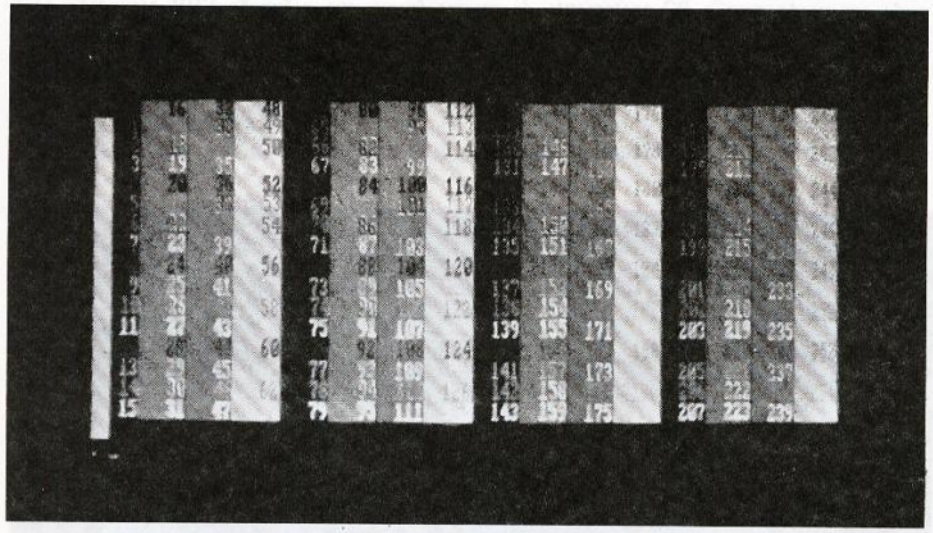


Figura 3. Efeito monocromático produzido pelo Programa 2


```
DO WHILE T
   SET INTENSITY ON
   SET TALK OFF
   STORE 0 TO x
   SET COLOR TO 1,23
   DO WHILE x < 24
      @ x,0
      STORE x+1 TO x
   ENDDO
   @ 5,30  SAY "DISTRIBUIDORA FERREIRA"
   @ 7,33  SAY "10 out 1985"
   SET COLOR TO 1,159
   @ 21,29 SAY "Escolher Uma"
   SET COLOR TO 1,112
   STORE 10 TO x
   DO WHILE x < 20
      @ x,20 SAY "
      STORE x+1 TO x
   ENDDO
   @ 10,25  SAY "1 - Mudar a Data"
   @ 11,25  SAY "2 - Arquivo de Clientes"
   @ 12,25  SAY "3 - Listas e Etiquetas de Enderecamento"
   @ 13,25  SAY "4 - Inventario"
   @ 14,25  SAY "5 - Pedidos de Venda"
   @ 15,25  SAY "6 - Faturas"
   @ 16,25  SAY "7 - Contas a Receber"
   @ 19,25  SAY "0 - Encerrado"
   STORE " " TO escolha
   @ 21,24  GET escolha
   READ NOUPDATE
ENDDO
```

Programa 3. Modificação da tela de menu, mostrando efeitos especiais em cores para o PC da IBM

APÊNDICES

APÊNDICE A *

Lista de Comandos, funções

```
? < exp > [ , < exp >]
@ < coordenadas > [SAY < exp > [USING' < picture > ' ] ]
                                    [GET < variável > [PICTURE ' < picture > ' ]]
ACCEPT  [" < cstring > "]  TO  < memvar >
APPEND  [FROM < arquivo > [SDF] [DELIMITED] [FOR < exp >]] ou [BLANK]
BROWSE [FIELDS < lista de campos >]
CANCEL
CHANGE FIELD < lista > [< escopo >] [FOR < esp >]
CLEAR  [GETS]
CONTINUE
COPY TO < arquivo > [ < escopo >] [FIELD < lista >] [FOR < exp >] [SDF]
        [DELIMITED [WITH < delimitador >]]  ou  [STRUCTURE]
COUNT   [< escopo >] [FOR < exp >] [ TO < memvar >]
CREATE  [< nome arquivo >]
DELETE  [< escopo >] [FOR < exp >]
DELETE FILE < arquivo >
DISPLAY [<escopo>][FOR <exp>][<exp lista>][OFF][FIELDS <lista de campos>]
DISPLAY STRUCTURE
DISPLAY MEMORY
DISPLAY FILES [ON < disk drive >] [LIKE < estrutura >]
```

* Fonte: *dBASE II® Product Manual* (versão 2.4, páginas 151-153).

DO < arquivo > disk drive
DO WHILE < exp >
EDIT
EJECT
ELSE
ENDDO
ENDIF
ENDTEXT
ERASE
FIND < chave >
GO ou GOTO [RECORD], ou [TOPO], ou [BOTTOM], < n >
HELP [comando]
IF < exp >
INDEX
INDEX ON < expressão de string de caracteres > TO < nome do arquivo de índice >
INPUT [" < cstring > "] TO < varmem >
INSERT [BEFORE] [BLANK]
JOIN TO < arquivo > FOR < expressão > [FIELDS < lista de campos >]
LIST
LOCATE [< escopo >] [FOR < exp >]
LOOP
MODIFY STRUCTURE
MODIFY COMMAND < arquivo de comando >
NOTE ou *
PACK
QUIT [TO < lista de comandos a nível do CP/M ou arquivos .COM >]
READ [NO UPDATE]
RECALL [< escopo >] [FOR < exp >]
REINDEX
RELEASE [< lista de varmem >] [ALL [LIKE <estrutura >]]
REMARK
RENAME < nome do arquivo atual > TO < novo nome do arquivo >
REPLACE [< escopo >] < campo > WITH < exp > [AND < campo > WITH < exp >]
REPORT [< escopo >] [FORM < arquivo de relatório > [TO PRINT] [FOR < exp >]
RESET
RESTORE FROM < arqmem > [ADDITIVE]
RETURN
SAVE TO < arquivo > [ALL LIKE < estrutura >]
SELECT [PRIMARY ou SECONDARY]
SET < parm > [ON] [OFF]
SET ALTERNATE TO < arquivo >
SET DEFAULT TO < disk drive >
SET DATE TO < string >
SET FORMAT TO < nome do arquivo de formatação >
SET HEADING TO < string >
SET INDEX TO < lista de arquivo de índice >

SKIP < + / – > [< n >]
SORT ON < campo > TO < arquivo > [ASCENDING] ou [DESCENDING]
STORE < exp > TO < varmem >
SUM < campo > [< escopo >] [TO < lista de memvar >] [FOR < exp >]
TEXT
TOTAL TO < arquivo > ON < variável chave > [FIELDS < lista de campos >]
UPDATE FROM < arquivo > ON < variável chave > [ADD < lista de campo >]
[REPLACE < lista de campo >]
USE < arquivo > [INDEX < nome do arquivo de índice >]
WAIT [TO < varmem >]

Funções:

@ (< string1 >, < string2 >)	função AT
*	função de registro excluído
#	função do número de registro
! (< string de caracteres >)	função de letras maiúsculas
$(< string de caracteres >, < início >, < tamanho >)	função de *substring*
CHR(< expressão numérica >)	número para o ASCII
DATE( )	função da data do sistema
EOF	função de fim-de-arquivo
FILE(< arquivo >)	função de existência
INT(< expressão numérica >)	função do número inteiro
LEN(< string de caracteres >)	função de tamanho
RANK(< string >)	função do valor ASCII
STR(< expressão numérica >, < tamanho > [, < decimais >])	função de *string*

VAL(< string de caracteres >)	função de valor
TRIM(< string de caracteres >)	ajusta *strings*
TYPE(< exp >)	fornece o tipo dos dados

APÊNDICE B *

Limitações e Restrições

número de campos por registro	32 máx.
número de caracteres por registro	1000 máx.
número de registros por banco de dados	65535 máx.
número de caracteres por *string* de caracteres	254 máx.
precisão dos campos numéricos	16 dígitos
maior número	aprox. $1{,}8 \times 10^{63}$
menor número	aprox. $1{,}0 \times 10^{-63}$
número de variáveis de memória	64 máx.
número de caracteres por linha de comando	254 máx.
número de expressões no comando SUM	5 máx.
número de caracteres no cabeçalho de REPORT	254 máx.

* Fonte: *dBASE® Product Manual* (versão 2.4, pág. 154).

número de caracteres na chave de índice ..... 99 máx.

número de GETS pendentes ..... 64 máx.

número de arquivos abertos ao mesmo tempo ..... 16 máx.

APÊNDICE C *

Mensagens de Erro

TAMANHO DECIMAL ILEGAL
Redefinir a Decimal.

NOME DE ARQUIVO INADEQUADO
Erro de sintaxe no nomedearquivo.

NOME DO CAMPO INADEQUADO
Redefinir o Nome.

TIPO DE CAMPO ILEGAL
Deve ser C, N, ou L.

TAMANHO DE CAMPO ILEGAL
Redefinir o tamanho do campo de dados.

NAO PODE SER INSERIDO – NAO EXISTE REGISTRO NO BANCO DE DADOS
Em vez disso, use o comando APPEND.

ARQUIVO NAO PODE SER ABERTO
Erro interno, chamar o revendedor para manutenção.

* Fonte: *dBASE® Product Manual* (versão 2.4, pág. 155-158)

ARQUIVO DE COMANDO NAO ENCONTRADO
Verificar a ortografia.

ITEM DE DADOS NAO ENCONTRADO

BANCO DE DADOS NAO ESTA INDEXADO
O comando FIND só é permitido em bancos de dados indexados.

DIRETORIO ESTA CHEIO
O diretório do disco do CP/M não comporta mais arquivos.

DISCO ESTA CHEIO
Não há espaço disponível no disco.

FIM DE ARQUIVO ENCONTRADO
O banco de dados em USE não está no formato correto. Se todos os registros estiverem corretos e presentes, use os comandos PACK e INDEX para compactá-los e reindexá-los.

CAMPO NAO ENCONTRADO
Reescrever a linha de comando.

ARQUIVO JA EXISTE

ARQUIVO NAO EXISTE

ARQUIVO JA ESTA ABERTO
Digite um comando USE ou CLEAR para encerrar o arquivo.

ARQUIVO DE FORMATACAO NAO PODE SER ABERTO

ARQUIVO DE FORMATACAO NAO ESTABELECIDO
Estabelecer (com SET) o arquivo de formatação apropriado.

TIPO DE DADOS ILEGAL

VALOR DE GOTO ILEGAL
Deve ser > 0 e < 65535

NOME ILEGAL DE VARIAVEL
Somente caracteres alfanuméricos e sinais de dois pontos são admitidos como nomes de campos e de variáveis.

INDICE NAO CORRESPONDE AO BANCO DE DADOS
O dBASE II não pôde fazer a correspondência da chave com o banco de dados. Tente um outro arquivo de índice.

INDICE NAO PODE SER ABERTO
Verifique a ortografia ou faça a indexação (INDEX) do banco de dados.

UNIFICACAO TENDE A GERAR MAIS DE 65.534 REGISTROS
A sentença FOR admite um número excessivo de registros de saída unificados; faça-a mais restrita.

CHAVES NAO TEM MESMO TAMANHO

MACRO NAO E UM STRING DE CARACTERES
Os *strings* &macro devem ser *strings* de caracteres.

MAIS DE 5 CAMPOS PARA SUM
A soma está limitada a 5 campos.

EXCEDIDO LIMITE DE VIOLACAO DO ALOJAMENTO

NAO HA EXPRESSAO PARA SOMAR

FRASE "FOR" NAO EXISTE
Reescreva o comando com a sintaxe correta.

FRASE PALAVRA-CHAVE "FROM" NAO EXISTE
Reescreva o comando com a sintaxe correta.

NAO ENCONTRADO
É mais uma mensagem de tipo diagnóstico que uma mensagem de erro. O dBASE não pôde encontrar a chave. O Registro # foi colocado em 0.

EXPRESSAO NAO-NUMERICA

ARQUIVO NAO EXISTE

FRASE PALAVRA-CHAVE "ON" NAO ENCONTRADA
Reescreva o comando com a sintaxe correta.

VARIAVEIS DE MEMORIA FORA DA MEMORIA
Reduz o número ou tamanho de variáveis de memória.

TAMANHO DO REGISTRO EXCEDE O VALOR MAXIMO (DE 1000)
Reduza o tamanho de alguns dos campos ou crie um segundo banco de dados numa chave comum.

REGISTRO NAO ESTA NO INDICE
O arquivo de índice não foi atualizado após o acréscimo de um registro.
Faça a reindexação.

REGISTRO FORA DE ALCANCE
O número do registro é maior que o número de registros no banco de dados.
O Registro não existe.

ERRO INTERNO DO CLASSIFICADOR, NOTIFIQUE SCDP
Erro interno, chame o revendedor para manutenção.

TIPOS DOS DADOS FONTE E DE DESTINO SAO DIFERENTES
Verifique se ambos os tipos de dados são numéricos ou se são ambos de caracteres.

*** ERRO DE SINTAXE ***

ESPECIFICACAO DO FORMATO DO ERRO DE SINTAXE

ERRO DE SINTAXE, ENTRAR NOVAMENTE

FRASE "TO" NAO ENCONTRADA
Reescrever o comando com a sintaxe correta.

LINHA EXCEDE MAXIMO DE 254 CARACTERES
Diminuir o tamanho da linha de comando.

MUITOS ARQUIVOS ESTAO ABERTOS
O número máximo de arquivos a ser aberto por vez é 16.

NUMERO EXCESSIVO DE VARIAVEIS DE MEMORIA
O número máximo de variáveis admitidas é 64.

NUMERO EXCESSIVO DE RETORNOS ENCONTRADOS
Provavelmente um erro na estrutura de um arquivo de comando.

FRASE "WITH" NAO ENCONTRADA
Reescreva o comando com a sintaxe correta.

Nº ARQUIVO NAO ESPECIFICADO
Erro interno, chame o revendedor para manutenção.

*** VERBO DO COMANDO IRRECONHECIVEL
Verifique a ortografia.

VARIAVEL NAO ENCONTRADA
É preciso criar a variável, ou verificar a ortografia.

APÊNDICE D

TABELA DE CÓDIGOS ASCII

CARACTERES DE CONTROLE

BINARIO	HEX	DECIMAL	SIMBOLO	CODIGO	DESCRICAO
0000000	00	0	NUL	↑@	Nulo
0000001	01	1	SOH	↑A	Inicio do Cabecalho
0000010	02	2	STX	↑B	Inicio do Texto
0000011	03	3	ETX	↑C	Final do Texto
0000100	04	4	EOT	↑D	Final da Transmissao
0000101	05	5	ENQ	↑E	Consulta
0000110	06	6	ACK	↑F	Confirmacao
0000111	07	7	BEL	↑G	Campainha
0001000	08	8	BS	↑H	Retrocesso
0001001	09	9	SH	↑I	Tabulacao Horizontal
0001010	0A	10	LF	↑J	Avanco de Linha
0001011	0B	11	VT	↑K	Tabulacao Vertical
0001100	0C	12	FF	↑L	Alimentacao de Formulario
0001101	0D	13	CR	↑M	Retorno do Carro
0001110	0E	14	SO	↑N	Mudanca para "numeros"
0001111	0F	15	SI	↑O	Mudanca para "letras"
0010000	10	16	DLE	↑P	Escape de enlace de dados
0010001	11	17	DC1	↑Q	Controle de Dispositivo 1
0010010	12	18	DC2	↑R	Controle de Dispositivo 2
0010011	13	19	DC3	↑S	Controle de Dispositivo 3
0010100	14	20	DC4	↑T	Controle de Dispositivo 4
0010101	15	21	NAK	↑U	Aviso de Recepcao Negativa
0010110	16	22	SYN	↑V	Preenchimento p/Sincronismo
0010111	17	23	ETB	↑W	Fim de Transmissao do Bloco
0011000	18	24	CAN	↑X	Cancelamento
0011001	19	25	EM	↑Y	Fim de Meio de Dados
0011010	1A	26	SUB	↑Z	Substituto
0011011	1B	27	ESC	↑[	Escape
0011100	1C	28	FS	↑\	Separador de Arquivo
0011101	1D	29	GS	↑]	Separador de Grupo
0011110	1E	30	RS	↑↑	Separador de Registro
0011111	1F	31	US	↑_	Separador de Unidades
1111111	7F	127	DEL	DEL	Apaga

CARACTERES IMPRIMIVEIS

BINARIO	HEX	DECIMAL	SIMBOLO	BINARIO	HEX	DECIMAL	SIMBOLO
0100000	20	32	ESPACO	1010000	50	80	P
0100001	21	33	!	1010001	51	81	Q
0100010	22	34	"	1010010	52	82	R
0100011	23	35	#	1010011	53	83	S
0100100	24	36	$	1010100	54	84	T
0100101	25	37	%	1010101	55	85	U
0100110	26	38	&	1010110	56	86	V
0100111	27	39	'	1010111	57	87	W
0101000	28	40	(	1011000	58	88	X
0101001	29	41	)	1011001	59	89	Y
0101010	2A	42	*	1011010	5A	90	Z
0101011	2B	43	+	1011011	5B	91	[
0101100	2C	44	,	1011100	5C	92	\
0101101	2D	45	-	1011101	5D	93	]
0101110	2E	46	.	1011110	5E	94	↑
0101111	2F	47	/	1011111	5F	95	←
0110000	30	48	0	1100000	60	96	`
0110001	31	49	1	1100001	61	97	a
0110010	32	50	2	1100010	62	98	b
0110011	33	51	3	1100011	63	99	c
0110100	34	52	4	1100100	64	100	d
0110101	35	53	5	1100101	65	101	e
0110110	36	54	6	1100110	66	102	f
0110111	37	55	7	1100111	67	103	g
0111000	38	56	8	1101000	68	104	h
0111001	39	57	9	1101001	69	105	i
0111010	3A	58	:	1101010	6A	106	j
0111011	3B	59	;	1101011	6B	107	k
0111100	3C	60	<	1101100	6C	108	l
0111101	3D	61	=	1101101	6D	109	m
0111110	3E	62	>	1101110	6E	110	n
0111111	3F	63	?	1101111	6F	111	o
1000000	40	64	@	1110000	70	112	p
1000001	41	65	A	1110001	71	113	q
1000010	42	66	B	1110010	72	114	r
1000011	43	67	C	1110011	73	115	s
1000100	44	68	D	1110100	74	116	t
1000101	45	69	E	1110101	75	117	u
1000110	46	70	F	1110110	76	118	v
1000111	47	71	G	1110111	77	119	w
1001000	48	72	H	1111000	78	120	x
1001001	49	73	I	1111001	79	121	y
1001010	4A	74	J	1111010	7A	122	z
1001011	4B	75	K	1111011	7B	123	{
1001100	4C	76	L	1111100	7C	124	|
1001101	4D	77	M	1111101	7D	125	}
1001110	4E	78	N	1111110	7E	126	~
1001111	4F	79	O	1111111	7F	127	DEL

APÊNDICE E

PROGRAMAS SUPLEMENTARES

```
* menu4-2 edita registros do arquivo de inventario

SET INDEX TO Isbn,Autor,Titulo,Codedit,Tema
SELECT SECO
USE Editoras INDEX idedit
STORE T TO editreg
ERASE
DO WHILE editreg
   SELECT PRIM
   STORE "              " TO misbn
   @ 21,1 SAY "ENTRAR O NUMERO ISBN" GET misbn PICTURE "99999999999-!"
   READ NOUPDATE
   @ 20,0
   IF misbn="    "
      STORE F TO editreg
      LOOP
   ENDIF
   FIND &misbn
   IF #=0
      DO Testisbn
      IF .NOT. testvalido
         @ 20,1  SAY misbn+" E UM NUMERO ISBN NAO VALIDO"
      ELSE
         @ 20,1  SAY misbn+" NAO ESTA NO BANCO DE DADOS"
      ENDIF
      LOOP
   ENDIF
   STORE CODEDIT TO medit
   SELE SECO
   FIND &medit
   SELE PRIMARY
   SET FORMAT TO Menu4-1
   READ
   SET FORMAT TO SCREEN
   CLEAR GETS
ENDDO
RELEASE misbn,testvalido,editreg,medit
RETURN
```

MENU4-2 EDITA REGISTROS DE INVENTÁRIO


```
* menu4-3 EXCLUI registros do arquivo de inventario

SET INDEX TO Isbn,Autor,Titulo,Codetit,Tema
ERASE
STORE T TO delereg
DO WHILE  delereg
   STORE "              " TO misbn
   @ 21,1  SAY "ENTRAR O NUMERO UISBN" GET misbn PICTURE "99999999999-!"
   READ NOUPDATE
   @ 20,0
   IF misbn="    "
      STORE F TO delereg
      LOOP
   ENDIF
   FIND &misbn
   IF #=0
      DO Testisbn
      IF .NOT. testvalido
         @ 20,1 SAY misbn+" E UM NUMERO ISBN INVALIDO"
      ELSE
         @ 20,1 SAY misbn+" NAO ESTA NO BANCO DE DADOS"
      ENDIF
      LOOP
   ENDIF
   @ 1,1 SAY TITULO
   @ 2,1 SAY AUTOR
   @ 3,1 SAY TEMA
   STORE " " TO pergunta
   DO WHILE .NOT. pergunta$"YN"
      @ 5,1 SAY "EXCLUIR ESTE LIVRO (Y/N)?" GET pergunta  PICTURE "!"
      READ
      CLEAR GETS
   ENDDO
   IF PERGUNTA="Y"
      DELETE
   ENDIF
ENDDO
RELEASE misbn,testvalido,delereg,pergunta
PACK
RETURN
```

MENU4-3 EXCLUI REGISTROS NO INVENTÁRIO

```
* Programa -----:MENU5-2.CMD (ou MENU5-2.PRG)

USE vendas
STORE T TO editreg
ERASE
DO WHILE editreg
   STORE "      " TO mpven
   @ 1,1 SAY "NUMERO DO PEDIDO DE VENDA A SER EDITADO" GET mpven PICT "999999"
   READ
   IF mpven="      "
      STORE F TO editreg
      LOOP
   ENDIF
   GO BOTTOM
   DO WHILE NUMPVEN > mpven .AND. #>0
      SKIP -1
   ENDDO
   IF #=0 .OR. NUMPVEN #mpven
      @ 3,1 SAY mpven+" NAO ESTAVA NO BANCO DE DADOS"
      LOOP
   ENDIF
   @ 3,0
   STORE NUMCLN TO mnumcln
   SELE SECO
   Use Pedabr
   LOCATE FOR NUMPVEN=mpven
   IF EOF
      @ 3,1 SAY "PEDIDO DE VENDA FOI PREENCHIDO - NAO PODE SER EDITADO"
      LOOP
   ENDIF
   CLEAR GETS
   @ 3,0
   STORE T TO loopedit
   USE Clientes INDEX Numcln
   FIND &mnumcln
   @ 3,30 SAY NOME
   DO WHILE loopedit
      SELECT PRIMARY
      @ 3,1 SAY "NUMERO DE IDENTIFICACAO DO CLIENTE" GET mnumcln
      @ 5,1 SAY "NUMERO DE P.COM O CLIENTE" GET NUMPED
      READ
      IF NUMCLN #mnumcln
         SELE SECO
         USE Clientes INDEX Numcln
         FIND &mnumcln
         IF #=0
            @ 3,30 SAY "NUMERO DE IDENTIFICACAO DO CLIENTE INVALIDO"
            LOOP
         ELSE
            @ 3,30 SAY NOME
            SELECT PRIMARY
            REPLACE NUMCLN WITH mnumcln
         ENDIF
      ENDIF
```

MENU5-2 EDITA UM PEDIDO DE VENDA


```
      STORE F TO loopedit
   ENDDO ----- para o loop de verificacao do numcln
   SELECT PRIM
   USE Detven
   DO Busca
   STORE # TO posicao
   COPY FIELD ISBN,QTIDADE TO Arqtemp WHILE NUMPVEN=mpven
   GO posicao
   STORE posicao-1 TO numreg
   SELE SECO
   USE Arqtemp
   BROWSE
   GO TOP
   DO WHILE .NOT. EOF
      STORE numreg+1 TO numreg
      SELECT PRIMARY
      GO numreg
      REPLACE P.ISBN WITH S.ISBN, P.QTIDADE WITH S.QTIDADE
      SELECT SECONDARY
      SKIP
   ENDDO
   ERASE
   @ 1,1 SAY "VERIFICACAO DE NUMEROS ISBN"
   USE Inv INDEX Isbn
   SELECT PRIMARY
   GO posicao
   DO WHILE mpven=NUMPVEN .AND. .NOT. EOF
      STORE ISBN TO misbn
      SELE SECO
      FIND &misbn
      IF #=0
         @ 3,1 SAY misbn+" E INVALIDO " GET misbn
         READ NOUPDATE
      ENDIF
      SELECT PRIMARY
      IF misbn#ISBN
         REPLACE ISBN WITH misbn
         LOOP
      ENDIF
      SKIP
   ENDDO
   SELE PRIMARY
   USE Vendas
   ERASE
ENDDO ------ para o loop principal
RELEASE editreg,posicao,mpven,misbn,numreg
SELECT SECONDARY
USE
SELECT PRIMARY
RETURN
```

MENU 5-2 EDITA UM PEDIDO DE VENDA

```
* Programa --------: Menus5-3.CMD (ou MENU5-3.PRG)
* Obs.-------------: Este programa cancela um Pedido de Venda.

USE Vendas
STORE T TO editreg
ERASE
DO WHILE editreg
   STORE"       "TO mpven
   @ 1,1 SAY "PEDIDO DE VENDA A SER CANCELADO" GET mpven PICT "999999"
   READ
   IF mpven="       "
      STORE F TO editreg
      LOOP
   ENDIF
   @ 3,0
   @ 5,0
   GO BOTTOM
   DO WHILE NUMPVEN > mpven .AND. #>0
      SKIP -1
   ENDDO
   IF NUMPVEN#mpven
      @ 3,1 SAY mpven+" NAO ENCONTRADO"
      LOOP
   ENDIF
   @ 3,1 SAY "CODIGO DE IDENTIFICACAO DO CLIENTE: "+NUMCLN
   @ 3,30 SAY "NUMERO DO PEDIDO DE COMPRA DO CLIENTE: "+NUMPED
   STORE " " TO pergunta
   DO WHILE .NOT. PERGUNTA $"YynN"
      @ 5,1 SAY "PEDIDO CANCELADO (Y/N)? " GET pergunta
      READ
   ENDDO
   IF pergunta$"nN"
      LOOP
   ENDIF
   @ 5,0
   SELECT SECONDARY
   USE pedabr
   LOCATE FOR mpven=NUMPVEN
   IF EOF
      @ 5,1 SAY mpven "JA ARQUIVADO"
   ELSE
      DELETE
      PACK
   ENDIF
   SELECT SECONDARY
   USE Detven
   DO Busca
   RLPL QTIDADE WITH 0 WHILE mpven=NUMPVEN
ENDDO --- para editreg
RELEASE editreg, enconprim, mpven
SELE SECO
USE
SELE PRIM
RETURN
```

MENU5-3 CANCELA UM PEDIDO DE VENDA


```
* menu6-2 apresenta pedidos pendentes
ERASE
USE Pedpend
STORE 20 TO TAMPAG
INPUT "DESEJA UMA COPIA IMPRESSA" TO PERGUNTA
IF PERGUNTA
   SET PRINT ON
   SET FORMAT TO PRINT
   STORE 55 TO TAMPAG
ENDIF
SET MARGIN TO 15
DO WHILE .NOT. EOF
   STORE "NOME DO CLIENTE   NUMERO ISBN      DATA       QTIDADE" TO CABECALHO
   @ 6,1 SAY CABECALHO
   ?
   STORE 8 TO LINHA
   DO WHILE .NOT. EOF .AND. LINHA < TAMPAG
      STORE LINHA +1 TO LINHA
      STORE NUMPVEN TO MPVEN
      SELE SECO
      USE VENDAS
      DO BUSCA
      STORE DATA DO DATAPED
      STORE NUMCLN TO MNUMCLN
      USE CLIENTES INDEX NUMCLN
      FIND &MNUMCLN
      ? NOME, ISBN, DATAPED, QTIDADE-QTIDENV
      SELE PRIM
      SKIP
   ENDDO
   EJECT
ENDDO
SET PRINT OFF
SET FORMAT TO SCREEN
RELEASE MNUMCLN,MPVEN,PERGUNTA,DATAPED,TAMPAG,LINHA,ENCONPRIM,CABECALHO
```

MENU6-2 APRESENTA PEDIDOS PENDENTES

GLOSSÁRIO

.AND.
Operador booleano utilizado para unir duas expressões lógicas, de modo que a expressão resultante se aplique às características partilhadas pela expressão. Por exemplo: CLASSE = "3" .AND.SALA = "122" restringe a expressão a alunos do terceiro ano, atribuídos à sala 122.

.NOT.
Operador booleano usado para se obter o oposto da expressão. Por exemplo: .NOT.CLASSE = "3" significa "todas as classes, exceto a terceira.

.OR.
Operador booleano utilizado para unir dois grupos dentro de uma expressão lógica. Por exemplo, a terceira e a quarta classe podem estar unidas dentro da expressão CLASSE = "3" .OR.CLASSE = "4".

?
Comando do dBASE II para apresentar itens escolhidos.

??
Comando para a apresentação de dados. É semelhante ao comando?, exceto que não emite um retorno de carro e um avanço de linha.

@
Operador utilizado em conjunção com os comandos SAY e GET, para gerar apresentações tanto no terminal CRT como na impressora. É também usado como uma função para localizar a posição de uma *substring* dentro de uma *string* de caracteres.

*
Normalmente usado para indicar a multiplicação. No dBASE II também é utilizado para a exclusão de dados e como comando para inserir comentários em programas.

#
É utilizado como uma função de múltiplas finalidades. Normalmente é usado como abreviação de "diferente de". No dBASE II é também utilizado como símbolo para "número de registro". Por exemplo: DISPLAY FOR # = 3 apresentaria o terceiro registro de dados.

!
Símbolo que, no dBASE II, indica ao computador que não deve distinguir entre caracteres em maiúsculas e caracteres em minúsculas.

$
Operador do dBASE II que permite ao usuário buscar por uma seqüência de caracteres contidos em um campo ou em uma variável de memória. Freqüentemente é denominado operador de *substring* ou de *string*. Também pode ser interpretado como significando "contido em". Por exemplo: "Ricardo"$NOME permitirá a busca no campo NOME da *string* de caracteres RICARDO.

ACCEPT TO [nome da variável de memória]
Comando do dBASE II que admite a entrada de *strings* de caracteres em variáveis de memória designadas pelo nome, sem a necessidade de delimitadores. Geralmente utilizado no interior de procedimentos (arquivos de comando do dBASE II).

ADL
Linguagem de Desenvolvimento de Aplicativos – a linguagem de computação usada com o dBASE II.

APPEND
Comando do dBASE II para inclusão de registro em um banco de dados.

APPEND BLANK
Variante de APPEND, que inclui registros em branco em um banco de dados. Normalmente usado no interior de procedimentos.

APPEND FROM < nomearquivo >
Usado para acrescentar dados provenientes do arquivo designado para o arquivo em uso.

ARQUIVO
Um conjunto de informações como por exemplo um arquivo de banco de dados ou um arquivo de comando armazenado como uma unidade identificável em um disco.

ARQUIVO < nomearquivo >
Função usada para testar quanto à existência de um arquivo.

ARQUIVO, NOME DE
Usado para identificar o arquivo para o computador. Deve conter oito caracteres ou menos, deve-se iniciar com uma letra e não pode conter espaços em branco.

ARQUIVO, TIPO DE
Extensão de três caracteres no nome de arquivo vindo em seguida a um nome de arquivo e a um período. Usado para fazer a distinção entre tipos diferentes de arquivos com o mesmo nome e para identificar para o computador certos tipos de arquivo com os quais ele foi programado para lidar de uma forma específica. Por exemplo, o arquivo .DBF é reconhecido pelo dBASE II como um arquivo de banco de dados.

BACKUP
Processo de copiar um disco ou arquivo em disco em um outro disco para garantia contra possíveis perdas futuras. Também a cópia backup – de reserva – do arquivo ou disco.

BANCO DE DADOS
Um "depósito" de informações armazenadas e organizadas de forma que os dados possam ser facilmente recuperados. Normalmente associado a um banco de dados armazenado no computador que é usado para aplicações múltiplas. Um exemplo simples de banco de dados não pertencente ao computador é a lista telefônica.

BASIC
Uma linguagem de computação. É a sigla para *Beginners All Purpose Symbolic Instruction Code* – Código de Instrução Simbólica de Finalidades Gerais para Principiantes.

BOOLEANO
Método de lógica do computador baseado no trabalho de George Boole, que desenvolveu um certo tipo de álgebra.

BOOTING
Processo de iniciar a atividade no computador.

BROWSE
Comando do dBASE II usado para edição. São apresentados vários registros simultaneamente para edição em tela cheia.

BYTE
O total de memória necessário para armazenar um caractere como um "A" ou "#" ou "9".

CALL
Para se obter a execução de um programa por meio de um comando emitido no interior de um outro. É também um comando do dBASE II para chamar um programa em linguagem assembly.

CAMPO
Em sistemas como o dBASE II, um campo conterá um item de informação. Corresponde a uma coluna de informação em um banco de dados em papel.

CAMPO, DESCRIÇÃO DE
Descrição do campo. Consiste em três partes: o nome, o tipo e a largura do campo, inclusive as casas decimais (se houver).

CAMPO, NOME DE
Nome de campo: é o "título" do campo. Contém dez caracteres ou menos, deve-se iniciar por uma letra e não pode conter espaços em branco.

CAMPO, TAMANHO DE
Tamanho do campo: o número de posições de caracteres necessário para conter os dados a serem colocados no campo.

CAMPO, TIPO DE
Tipo de campo: o tipo de dados que podem ser armazenados em um campo. Os três tipos de campos são os de caracteres, os numéricos e os lógicos.

CAMPO, LARGURA DE
Largura do campo: o número de espaços necessários no campo para conter os dados.

CANCEL
Comando usado para encerrar um procedimento e fazer o controle do computador retornar ao teclado.

CASE
Variante da instrução IF. Pode ser usada somente no interior de um DO CASE/ENDCASE.

CARACTERE
Qualquer símbolo do teclado que possa ser impresso, como "A" ou "$" ou "1" ou "a" (inclui espaços em branco).

CARACTERE, CAMPO
Campo de caracteres: um campo do banco de dados reservado para conter caracteres.

CARACTERE, *STRING*
String de caracteres: uma seqüência contínua de caracteres "Sérgio Dias".

CHANGE
Comando do dBASE II usado para edição.

CHR( )
Função que permite o controle direto das funções especiais de uma impressora ou de um terminal CRT.

CLEAR

Faz retornar ao dBASE II. Todos os bancos de dados em uso são encerrados, todas as variáveis de memória são abandonadas e o sistema fica exatamente como quando você inicialmente entra no dBASE II.

CLEAR GETS

Comando que elimina todos os GETS internamente, sem alterar a tela (como o faria o comando ERASE). Isso limitará o âmbito de READ para apenas os GETS emitidos após o comando CLEAR GETS.

COBOL

Linguagem de computação usada amplamente em aplicativos profissionais para computadores de grande porte. A primeira linguagem de computação próxima à linguagem humana. É a sigla de *Common Business Oriented Language* – Linguagem Orientada para Aplicações Comerciais.

COMMAND

É a terminologia do dBASE II para uma instrução do computador.

COMANDO, ARQUIVO DE

Conjunto de instruções do computador armazenadas ou gravadas em um disco para uso repetido. É um termo da terminologia do dBASE II para um procedimento do computador.

CONCATENAÇÃO

Concatenação: a união de duas *strings* de caracteres para formar uma terceira.

CONDIÇÃO

Uma expressão lógica que pode ser usada para definir mais explicitamente um comando como DISPLAY. DISPLAY FOR NOME = "SÉRGIO DIAS" modifica o comando DISPLAY de modo que ele se aplique somente a registros que atendam à condição NOME = "SÉRGIO DIAS".

CONDICIONAL, SUBSTITUIÇÃO

Uma técnica para modificar o conteúdo de um banco de dados em que o comando REPLACE seja modificado por uma condição. REPLACE PROFESSOR WITH "LUÍS" FOR SALA = "171" substituirá o conteúdo do campo PROFESSOR por "LUÍS" apenas no caso dos registros que atendam à condição "SALA = "171".

CONTINUE

Comando que posiciona no registro seguinte nas condições especificadas pelo comando LOCATE.

CONTROL, TECLA

Tecla de controle: uma tecla que potencialmente fornece um terceiro valor para todas as teclas no teclado. Semelhante à tecla shift, que dá um segundo significado a cada uma das teclas do teclado.

COPY TO < nomedearquivo >

Comando do dBASE II que copia o banco de dados em uso naquele designado pelo nome. Usado para criar uma cópia do banco de dados como reserva. Existem variantes que permitem copiar somente partes selecionadas do banco de dados em uso no banco de dados designado.

COUNT

Comando de apresentação de dados que conta o número de registros que atendem a uma determinada expressão condicional.

CP/M®

Control Program/Microcomputer – Programa do Controle/Microcomputador. Um sistema operacional de computadores muito popular. O CP/M® é uma marca registrada da Digital Research Corporation.

CPU

Unidade Central de Processamento – o processador central do sistema do computador. Contém o armazenamento principal, a unidade aritmética e grupos de registros especiais.

CREATE

Comando do dBASE II que permite estabelecer a estrutura de um novo arquivo de banco de dados.

CRT

Tubo de Raios Catódicos. Usado para denotar o dispositivo de apresentação em vídeo usado em conjunção com os computadores.

CTRL, TECLA

Tecla de Controle, usada em combinação com várias teclas para dar-lhes um segundo significado e função.

DADOS

Uma unidade de informação. Geralmente sem significado enquanto um item independente. Pode conduzir informações quando usada juntamente com um outro item. Por exemplo, o número de telefone 555-3213 é relativamente inútil sozinho. Ele transporta informações quando usado juntamente com um outro item de dados, Paulo Simões.

DASD

Dispositivo de Armazenamento de Acesso Direto, como um disco.

DBMS
Database Management System – sistema gerenciador de banco de dados.

DEBUG
Depuração: o processo de eliminação de erros em um programa.

DEFAULT
Quando o computador recebe um comando ele deve executar algum procedimento. A menos que instruído em contrário ele assumirá o procedimento pré-programado. Esse procedimento pré-programado é chamado default (padrão).

DELETE
Comando do dBASE II que marca registros para exclusão (veja também PACK).

DELETE FILE
Comando para excluir um arquivo. O arquivo não pode estar em uso.

DELIMITADORES
Delimitadores: um modo de identificar *strings* de caracteres para o computador. Por exemplo, ele permite ao computador distinguir entre o campo NOME e a palavra "NOME".

DISCO
Placa circular revestida por material magnético para armazenamento de dados. O meio pelo qual um computador pode permanente ou temporariamente armazenar informações a serem usadas ou lidas posteriormente.

DISK DRIVE
Unidade de disco: dispositivo mecânico usado para leitura e registro de informações em um disco.

DISPLAY
Comando do dBASE II para apresentar o conteúdo de um registro de dados.

DISPLAY ALL
Uma variante do comando DISPLAY. Apresenta todos os registros do banco de dados em uso – parando a cada 15 registros.

DISPLAY FILES ON < indicador da unidade de disco >
Comando do dBASE II que apresenta o nome de arquivo, número de registros e a data da última modificação de todos os arquivos de banco de dados no disco designado.

DISPLAY FOR < condição >
Variante do comando DISPLAY. A sentença FOR < condição > modifica o comando DISPLAY de modo que todos os registros que atendam à condição sejam apresentados, 15 de cada vez.

DISPLAY MEMORY
Variante do comando DISPLAY que apresentará nome, tipo, tamanho e conteúdo de todas as variáveis de memória.

DISPLAY OFF
Variante de DISPLAY que apresenta o registro do banco de dados sem o número de registro.

DISPLAY STRUCTURE
Variante do comando DISPLAY usada para apresentar a estrutura do banco de dados presentemente em uso.

DO < nomedearquivo >
Comando do dBASE II usado como alternativa às instruções com alojamento múltiplo IF, ENDIF. Deve ser acompanhado por um comando ENDCASE. Normalmente usado em procedimentos.

DOS
Disk Operating System – Sistema Operacional do Disco – software.

DO WHILE < condição >
Comando do dBASE II que indica ao computador que deve executar repetidamente a seqüência de comandos entre a instrução DO WHILE e a instrução que conclui ENDDO, enquanto a condição estiver sendo atendida. Usado nos arquivos de comando (procedimentos) do dBASE II.

EDIT < número do registro >
Comando que permite a modificação do conteúdo de um campo de dados. No dBASE II ele chama uma operação em tela cheia que permite a modificação dos conteúdos dos campos desejados pelo deslocamento do cursor para a posição apropriada e a digitação dos novos dados.

EJECT
Comando do dBASE II que emite uma página na impressora.

ELSE
Desvio alternativo do arquivo de comando da execução do comando dentro de um IF.

ENDCASE
Comando do arquivo de comando que encerra um DO CASE.

ENDDO
Encerra o arquivo de comando para um comando DO WHILE.

ENDIF
Comando de encerramento do comando IF.

EOF
End Of File – fim de arquivo: função do dBASE II; também um caractere especial de arquivo ASCII.

ERASE
Comando que limpa o CRT.

ESCAPE
Tecla de muitos dos terminais de computadores. Possui o valor ASCII de 27. Em terminais sem o ESCAPE pode-se usar CONTROL[. ESCAPE permite interromper um programa ou um comando de longa execução.

FIND < chave >
Comando do dBASE II para encontrar um registro com a chave. Pode ser usado somente com arquivos indexados.

FLOPPIES
Floppy disks: discos flexíveis.

FLOPPY DISK
Disco flexível: meio de armazenamento comumente usado em microcomputadores. As informações são armazenadas em um fino disco flexível semelhante.

FLOPPY DISK SYSTEM
Sistema de disco flexível: um sistema de computador que utiliza discos flexíveis.

FORTRAN
Linguagem tradicional de computação usada principalmente em aplicações científicas. Sigla de FORmula TRANslator (tradutor de fórmulas).

FORMAT, ARQUIVO
Arquivo de formatação: arquivo especial usado para produzir "formulários" no vídeo.

GET
Comando do dBASE II usado em conjunção com o comando @ para apresentar o conteúdo de um campo ou de uma variável de memória. Quando usado juntamente com o comando READ, o conteúdo do campo ou da variável de memória pode ser modificado pela simples digitação da nova informação.

GO BOTTOM
Comando que faz o dBASE II ir ao último registro do banco de dados em uso.

GO TOP
Comando que faz o dBASE II ir ao primeiro registro do banco de dados em uso.

GOTO < número do registro >
Esse comando é usado para reposicionar o indicador de registro no número de registro designado.

HARD DISK
Disco rígido: placa rígida de metal revestida por uma película magnética. Pode armazenar enormes quantidades de informações.

HARD DISK SYSTEM
Sistema de disco rígido: sistema de computador que utiliza discos rígidos.

I/O
Entrada/Saída: mecanismo pelo qual o computador admite e distribui as informações para os dispositivos periféricos. É a sigla de Input/Output.

IF < condição >
Instrução que indica ao computador que deve executar um conjunto de comandos enquanto essa condição estiver sendo atendida.

INDEX ON < lista de campo > TO <nomedearquivo >
Comando do dBASE II que cria um arquivo de índice com o nome de arquivo designado. O arquivo de índice faz com que o conteúdo do arquivo de banco de dados apareça para ser ordenado pela ordem lógica dos conteúdos dos campos relacionados (listados).

INPUT TO < nome da variável de memória >
Comando do dBASE II que permite a entrada pelo teclado de dados numéricos para uma variável de memória. Somente os dados numéricos serão admitidos. Geralmente usado com procedimentos.

INSERT
Comando do dBASE II que insere um novo registro de dados no meio de um arquivo de banco de dados.

INT( )
Função que arredondará um número com casas decimais, eliminando tudo que estiver à direita do ponto decimal. É a abreviação de INTeger (inteiro).

JOIN
Comando que permite a união (JOIN) de dois bancos de dados relacionais.

KEYBOARD
Teclado: dispositivo semelhante ao teclado de uma máquina de escrever, pelo qual o usuário pode "se comunicar" com o computador.

LEN (nome de variável de memória)
Função do dBASE II que informa ao usuário quantos caracteres estão na variável de memória de *string* designada.

LIST
Comando do dBASE II para apresentar todos os registros em uma seqüência contínua.

LOCATE FOR < condição >
Comando do dBASE II usado para localizar um registro que atenda à condição.

LÓGICOS, CAMPOS
Campos lógicos: campos com dois únicos conteúdos possíveis. Usado quando há apenas duas entradas possíveis e mutuamente exclusivas, como "T" e "F" (Verdadeiro e Falso) ou "Y" e "N" (Sim e Não).

LÓGICOS, REGISTROS
Registros lógicos: registros em um arquivo de índice ou outro arquivo semelhante que são "imagens" de parte dos dados efetivos. Normalmente usado para auxiliar no uso dos dados efetivos.

LOOP
Comando que faz com que a execução de um arquivo de comando salte novamente ao início de DO WHILE. É usado para desviar de alguma condição não desejada.

MASSA, MEMÓRIA DE
Memória de massa: dispositivos de armazenamento periférico, como unidades de disco e gravadores cassete.

MEMÓRIA, VARIÁVEL DE
Variável de memória: permite o armazenamento de informações na memória principal do computador para uso temporário. É um conceito semelhante ao de memória em uma calculadora eletrônica. Todas as informações se perdem quando o equipamento é desligado.

MENU
Procedimento do computador que apresenta um conjunto de escolhas para uma tarefa.

MENU, SISTEMA DE
Sistema de menu: procedimento do computador que utiliza uma seleção de menu para escolher uma seqüência de procedimentos para que o computador siga.

MICROCOMPUTADOR
Computador de pequeno porte projetado principalmente para um único usuário.

MILISSEGUNDO
1/1000 de segundo.

MINICOMPUTADOR
Pequeno computador, geralmente para uso simultâneo de um pequeno número de pessoas. Um pouco maior e mais potente que um micro.

MODIFY COMMAND
Comando do dBASE II que permite criar e/ou editar o conteúdo de um procedimento (arquivo de comando).

MODIFY STRUCTURE
Comando do dBASE II que permite modificar a estrutura de um banco de dados. É um comando que deve ser usado com cuidado.

MS-DOS
Microsoft Disk Operating System – Sistema Operacional em Disco da Microsoft: é uma marca registrada da Microsoft.

NOTE < conteúdo em texto >
Comando do dBASE II (geralmente usado em procedimentos) que faz com que o computador ignore o texto escrito em uma linha. Usado para descrever o procedimento para referência futura.

NÚMERO DE REGISTRO
Número de identificação atribuído a cada registro pelo sistema gerenciador do banco de dados. O número de registro é exclusivo a um registro (não pode haver dois registros com um mesmo número).

NUMÉRICOS, CAMPOS
Campos numéricos: campos que contêm números que se pretende usar em cálculos numéricos.

OPERACIONAL, SISTEMA
Sistema Operacional: software que transforma o sistema de hardware do computador em um computador.

PACK
Comando que exclui fisicamente todos os registros marcados para exclusão por meio do comando DELETE.

PASCAL
Linguagem de computação tendo algumas das características do dBASE II.

PC

Personal Computer – Computador Pessoal: recentemente se tornou sinônimo do PC da IBM em outros computadores semelhantes.

PERIFÉRICOS

Dispositivos usados com o computador, como a impressora e as unidades de disco.

PIP

Comando do CP/M® que permite ao usuário copiar arquivos do computador de um disco para outro.

PL/1

Uma moderna linguagem de computação. Significa *Programming Language One* – Linguagem de Programação Um.

POINTER

Indicador: mecanismo de software que dirige o computador para o registro de interesse.

PRIMÁRIA, CHAVE

Chave primária: modo único de identificar o registro que é diretamente utilizável pelo sistema do computador. No dBASE II a PRIMARY KEY é o número de registro.

PRINTER

Impressora: dispositivo periférico que faz os dados saírem do computador para o papel.

PROGRAMA

Programa: série de comandos a serem executados pelo computador, formando uma unidade. Um procedimento em um programa. No dBASE II um arquivo de comando é um programa.

PROGRAMADOR

Qualquer pessoa que desenvolva programas.

PROMPT

Indicador: uma indicação do computador de que ele está pronto para admitir comandos do teclado. Também uma solicitação de informações específicas a serem inseridas por meio do teclado.

QUERY

Solicitação do usuário de informações ao computador. Normalmente uma solicitação pelo teclado.

QUIT

Comando que faz com que o dBASE II encerre todos os arquivos presentemente em uso e saia para o sistema operacional.

RAM

Sigla para *Random Access Memory* – Memória de Acesso Aleatório. Geralmente é a memória principal do computador. A memória de acesso direto é realmente uma memória do computador que pode ser endereçada diretamente e na qual se pode gravar e a partir da qual se pode ler dados.

RANK( )

Função que fornece o valor ASCII (em decimal) ou um caractere imprimível.

READ

Comando do dBASE II usado juntamente com o comando GET, para executar edição em tela cheia (cursor) dos conteúdos de variáveis de memória e de campos de dados.

RECALL

Comando do dBASE II que "desmarca" registros anteriormente marcados para exclusão, por meio do comando DELETE.

REGISTRO

Registro: uma unidade inteira de itens de dados. No dBASE II, corresponde às informações contidas em uma linha de uma tabela retangular de linhas e colunas.

REGISTRO, BLOQUEIO DO

Artifício para proteger contra possível edição simultânea de um único item de dados em sistemas em que pode haver mais de um usuário ao mesmo tempo.

REGISTROS FÍSICOS

Os registros efetivos dos dados.

RELACIONAL, SISTEMA DE BANCO DE DADOS

Tipo de sistema gerenciador de banco de dados, baseado no uso de tabelas de linhas e colunas. Diferentes arquivos de banco de dados são vinculados (relacionados) pelo conteúdo de um bampo de dados em que o conteúdo do campo pode ser comum a ambos os arquivos de banco de dados.

RELEASE < nomes de variáveis de memória >

Comando do dBASE II que "apaga" as variáveis de memória designadas.

RENAME < nomedearquivo1 > TO < nomedearquivo2 >

Comando do dBASE II para dar novo nome a um arquivo. O arquivo não pode estar presentemente em uso.

REPLACE < nomedecampo > WITH < novo conteúdo >

Comando do dBASE II que substitui o conteúdo do campo de dados designado pelo novo conteúdo desejado. Mais freqüentemente usado no interior de procedimentos. Pode ser efetivamente usado a partir do teclado em conjunção com FOR < condição >.

REPORT
Comando do dBASE II que prepara um relatório baseado no conteúdo de um banco de dados e em suas respostas a uma série de perguntas simples. O relatório é apresentado no terminal CRT. As respostas às perguntas são fornecidas por meio do teclado para uso primeiramente com um relatório específico. As respostas são gravadas em um arquivo .FRM e o relatório será depois gerado automaticamente. Pode haver vários relatórios disponíveis ao mesmo tempo.

REPORT TO PRINT
Variante de REPORT que faz com que o relatório seja impresso.

RETURN
Comando que encerra um arquivo de comando e retorna ao procedimento mais alto a seguir. O controle volta ao teclado se não houver procedimento mais alto.

RETURN, TECLA
Semelhante à tecla de retorno do carro em máquinas de escrever. Provavelmente estará designada como ENTER em um terminal de computador.

ROM
Sigla de *Read Only Memory* – memória de leitura apenas.

SAVE
Comando que copia as variáveis de memória presentemente definidas para armazenamento de massa.

SAY
Comando do dBASE II usado em conjunção com o operador @ para apresentar informações no terminal CRT ou na impressora.

RUB
Tecla que apaga o caractere à esquerda do cursor.

SELECT PRIMARY
Comando que seleciona o banco de dados PRIMÁRIO.

SELECT SECONDARY
Comando que seleciona o banco de dados SECUNDÁRIO.

SEQÜENCIAL, ACESSO
Método de acesso para registros de dados por meio do qual o computador examina cada registro seqüencialmente – a começar pelo primeiro – até que o registro desejado (ou registros) seja localizado.

SET ECHO ON/OFF
Todos os comandos que provêm de um arquivo de comando são reproduzidos na tela como se tivessem sido digitados a partir do teclado. ECHO normalmente está desligado e deve ser ligado (ON) se necessário.

SET PRINT ON/OFF
Dirige a saída do computador para a impressora. Normalmente a saída não é direcionada para a impressora.

SET TALK ON/OFF
Os resultados de comandos são normalmente apresentados no terminal CRT. Isso muitas vezes não é desejável quando se usa arquivos de comando.

SKIP [n]
Comando que posiciona o banco de dados fazendo-o avançar ou retroceder um certo número de registros.

SORT
Comando que fará com que o banco de dados seja fisicamente reordenado em alguma ordem desejada – normalmente pelo valor numérico de algum(ns) campo(s) ou alfabeticamente, de acordo com o conteúdo de um campo de caracteres.

STORE
Comando que armazena dados em variáveis de memória.

STR( )
Função que lida com uma variável numérica como se ela fosse uma *string* de caracteres.

STRING
Abreviação de *string* de caracteres, como "Sérgio Dias".

STRUCTURE
Estrutura: a organização predefinida de seu banco de dados. É estabelecida pelo nomedearquivo, tipodearquivo e pelo tamanhodearquivo.

SUBSTRING
Parte de uma *string* de caracteres. Por exemplo, "Sérgio" é a *substring* de "Sérgio Dias".

SUM < nomedecampo >
Comando que soma o conteúdo dos campos designados.

TELA, ATRIBUTO DE
Características especiais de vídeo, como o vídeo reverso.

TERMINAL
O meio pelo qual o usuário se comunica com o computador e ele com o usuário. Os dispositivos de terminal mais comum nos microcomputadores são os terminais de vídeo.

TEXT
Comando do dBASE II para apresentar texto sem a necessidade do uso de delimitadores para identificá-lo.

TYPE
É a função de tipo de dados e produz um "C", "N", ou "L", dependendo da *string* de caractere única ser, respectivamente, de Caracteres, Numérica, ou Lógica ou "U" (undefined – não-definida).

UPDATE
Comando que combina registros provenientes de dois bancos de dados.

USE < nomedearquivo >
Comando que informa ao dBASE II que banco de dados você deseja usar.

VAL(x)
Permite o uso do conteúdo de um arquivo de dados de caracteres ou de variáveis de memória em cálculos aritméticos.

VÍDEO, TELA DE
O dispositivo de apresentação em vídeo, vinculado a seu computador.

WAIT
Comando do arquivo de comando que interrompe o seu processamento e espera uma entrada única de caractere proveniente do teclado com um indicador WAITING (esperando).

WAIT TO < variável de memória >
Igual a WAIT, exceto que a sentença TO faz com que o caractere do teclado seja armazenado na variável de memória designada.

ÍNDICE ANALÍTICO

Impresso na Prol editora gráfica ltda.
03043 Rua Martim Burchard, 246
Brás - São Paulo - SP
Fone: (011) 270-4388 (PABX)
com filmes fornecidos pelo Editor.

ISR-40-2861/85
UP Ag. Central
DR/SÃO PAULO

CARTA RESPOSTA COMERCIAL

Não é necessário selar

o selo será pago por:

DATALOGICA

01098 – São Paulo – SP

PESQUISA

A opinião do leitor é fundamental para nós.
Pedimos a gentileza de preencher esta pesquisa, para podermos responder cada vez melhor às suas necessidades.
Seu nome será incluído na nossa listagem de mala direta, para que você possa receber, periodicamente, informações sobre novas publicações e novos produtos.

Nome: _____

Empresa: _____

Cargo: _____

Endereço: _____

Cidade: _____ Estado: _____ CEP: _____

Telefone: ( ) _____ Telex: ( ) _____

Qual é a atividade comercial de sua empresa?

A. No setor de computação
( ) Fabricação
( ) Loja de sistemas
( ) Loja de suprimentos de PD
( ) Software
( ) Varejo
( ) Outros

B. Empresa fora do setor de computação
( ) Fabricação
( ) Comércio no varejo
( ) Comércio no atacado
( ) Financeira, Banco
( ) Imobiliária, Seguros
( ) Engenharia
( ) Governo
( ) Educação
( ) Militar
( ) Serviços de Saúde
( ) Serviços Jurídicos
( ) Transportes
( ) Serviços Gerais
( ) Comunicações
( ) Artes, Música, Cinema
( ) Outros

Quantos empregados trabalham na sua empresa?
( ) menos de 10
( ) de 10 a 25
( ) de 26 a 100
( ) de 101 a 300
( ) de 301 a 1000
( ) mais de 1000

1) Como conheceu esta publicação?
( ) Alguém viu ou comprou
( ) Revendedor de software ou vendedor
( ) Revendedor de hardware ou vendedor
( ) Propaganda
( ) Crítica publicada
( ) Livraria
( ) Diretamente na DATALÓGICA

2) Como comprou este livro?
( ) Diretamente na DATALÓGICA
( ) No seu revendedor de software
( ) No seu revendedor de Hardware
( ) Livraria

3) Já comprou algum outro livro ou publicação da DATALÓGICA e/ou da McGraw-Hill?
( ) Sim ( ) Não
Em caso positivo, assinale quais:
( ) Framework Para Principiantes
( ) Framework Finanças/Administração/Negócios
( ) dBase II Para Principiantes
( ) dBase II Aplicações Comerciais
( ) dBase III Banco de Dados para Todas as Aplicações
( ) Outros (favor especificar)

_____

_____

4) Com quais finalidades você comprou este livro?
( ) Pessoais
( ) Empresariais

5) Que marca e modelo de computador você utiliza?

_____

_____

6) Quantos e quais computadores existem na sua empresa?

_____

_____

7) Que tipos de programa de software você está usando no momento?
( ) Contábil
( ) Planilha
( ) Processador de texto
( ) Outros (favor especificar)

_____

_____

8) Que tipo de programa de software você pretende comprar nos próximos 12 meses?
( ) Contábil
( ) Vendas
( ) Inventário
( ) Outros (favor especificar)

_____

_____

9) Você conhece os softwares da DATALÓGICA?
( ) Sim ( ) Não
Em caso positivo, quais?
( ) dBase II
( ) dBase III
( ) Framework

10) Gostaria de receber maiores informações sobre os produtos da McGraw-Hill/DATALÓGICA? Em caso positivo, especifique.
( ) dBase II
( ) dBase III
( ) Framework
( ) Treinamento
( ) Simpósios
( ) Outras publicações

OUTROS LIVROS NA ÁREA *

SÉRIE McGRAW-HILL/DATALÓGICA

Byers – dBase II: Banco de Dados para Todas as Aplicações
Fishback – Framework: Aplicações – Finanças/Administração/Negócios
Freedman – dBase II para Principiantes
Harrison – Framework para Principiantes

GUIAS DO USUÁRIO

Baras – Lotus 1-2-3 – Guia do Usuário
Castlewitz – VisiCalc – Guia do Usuário
Curtis – WordStar – IBM PC – Guia do Usuário
Ettlin – WordStar – Guia do Usuário (versão CP/M)
Hoffman – MS/DOS – Guia do Usuário
Hogan – CP/M – Guia do Usuário
Poole – Apple II – Guia do Usuário
Sachs – IBM PC – Guia do Usuário
Sanders – Manual do Apple – Macintosh
Townsend – dBase II – Guia do Usuário

GUIAS DO OPERADOR

Gifford – Apple II – Comandos Básicos
Ingrahan – CP/M – Comandos Básicos
Wilson – VisiCalc – Comandos Básicos
Wilson – IBM PC – Comandos Básicos

* Com referência a outros títulos, consulte nosso Catálogo de Informática.

0-07-450043-0