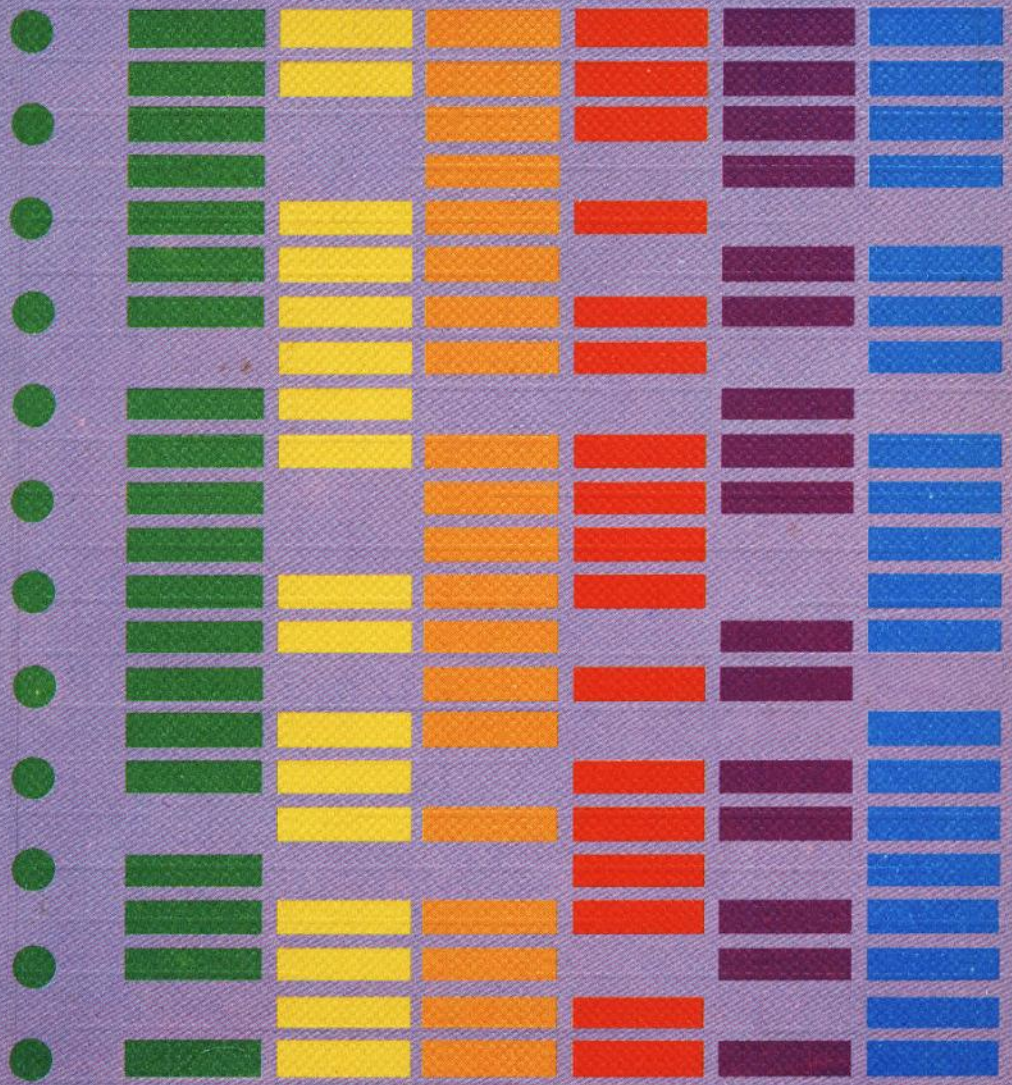
OSBORNE/ McGRAW-HILL

Edição Revisada

dBASE II®

GUIA DO USUÁRIO



Carl Townsend

McGraw-Hill

dBASE II®

Guia do Usuário

dBASE II®

Guia do Usuário

EDIÇÃO REVISADA

Carl Townsend

Tradução
Irineu Francisco da Silva

Supervisão Técnica
José Rubens Salles Toledo

Revisão Técnica
Eduardo Ponte da Conceição

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

Do original
Using dBASE II®

Coordenadora de Revisão: Daisy Pereira Daniel
Supervisor de Produção: Edson Sant'Anna

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

T675d 2. ed.	Townsend, Carl, 1938 dBASE II® : guia do usuário / Carl Townsend ; tradução Irineu Francisco da Silva. – São Paulo : McGraw-Hill do Brasil, 1986. Bibliografia. 1. dBase II (Programa de computador) 2. Linguagens de programação (Computadores) I. Título.
84-2047	17. CDD-651.8 18. -001.6425 18. -001.6424

Índices para catálogo sistemático:

1. dBASE II : Computadores : Programas : Processamento de dados 651.8 (17.) 001.6425 (18.)
2. Computadores : Linguagem de programação : Processamento de dados 651.8 (17.) 001.6424 (18.)
3. Linguagem de programação : Computadores : Processamento de dados 651.8 (17.) 001.6424 (18.)

AGRADECIMENTOS

Expresso meus agradecimentos à Ashton-Tate pelo apoio e pela cessão de cópia do dBASE, à Climax Manufacturing e a Bob Harkema da Climax por suas informações e desenvolvimento do programa de conexão TeleVídeo no Capítulo 11.

Sumário

Prefácio

Durante os últimos anos houve um rápido crescimento de um novo tipo de consultor. Este consultor trabalha com usuários de microcomputadores e freqüentemente tem experiência em grandes computadores e microcomputadores, como também algum conhecimento em análise de sistemas. Utilizando-se de novas ferramentas para microcomputadores como o dBASE II, este consultor está apto a desenhar conjuntos de programas para suas próprias necessidades a um custo próximo ao de muitos programas comercializados, desenhados para uma grande variedade de aplicações.

dBASE II®, Guia do Usuário é desenhado para ajudar estes leitores a se iniciarem no uso de um dos mais populares programas ferramenta – o dBASE II.

dBASE II®, Guia do Usuário dá a você uma introdução ao dBASE II, assim como penetra em um completo processo de desenho de sistemas. Exemplos lhe dão o núcleo de programas que podem ser usados em uma grande variedade de aplicações. Os sistemas-exemplo utilizados incluem programas de controle de estoque e custos. O livro é uma enciclopédia de minhas experiências com o dBASE II durante dois anos. Esta obra acentua o desenho estruturado e dá ênfase à programação de uma maneira estruturada utilizando análise do tipo "de cima para baixo".

C.T.

UM

Introdução ao dBASE II

Oficina de Automóveis do Fred. Todos conhecem Fred, o melhor mecânico de automóveis Volvo da cidade, e a maior parte de seus consertos envolvem Volvos e alguns outros carros estrangeiros. Ele mantinha um grande estoque de peças novas e usadas, sabendo que seus custos eram de $ 30 por hora e que precisava estar apto a localizá-las rapidamente e encomendar as peças necessárias para manter o estoque atualizado. Fred sabia que necessitava de um computador com um tipo qualquer de controle de estoques e assim foi visitar uma loja de computadores da cidade.

"Nós não temos nenhum programa específico para oficinas de automóveis", disse-lhe o vendedor. "Nós temos o Superestoque, que é realmente destinado a uma indústria de manufaturas e seu estoque. Para a sua aplicação você precisará de um compilador BASIC e escrever seu próprio programa. Na verdade, você terá que escrever vários programas. Uma alternativa seria comprar um programa que atenderia algumas das suas necessidades."

Fred não entendia profundamente de computadores; na verdade, ele nunca havia utilizado um computador. A idéia de escrever seu próprio programa para processar e extrair relatórios sobre seu estoque de peças pareceu-lhe um trabalho impossível. Além disso Fred tinha muito pouco tempo livre para aprender a programar. Ele deixou a loja de computadores de mãos vazias.

Poucas semanas depois, Bill Braxton, um programador de computadores, levou seu Volvo para trocar um alternador. Fred sabia que tinha um alternador em estoque, mas não conseguia encontrá-lo. "Devo tê-lo colocado no Volvo que consertamos a alguns dias atrás", disse Fred enquanto iniciava uma procura entre as faturas para encontrar uma que indicasse onde o alternador tinha sido instalado. Bill disse a Fred que ele estava necessitando de um microcomputador e Fred riu.

"Eu sei disso, eu sei disso, mas quem irá programá-lo?"

"Olhe", disse Bill, "eu escreverei um programa para processar os seus dados de estoque e alugarei meu computador portátil por um dia por $ 100. Eu lhe mostrarei como dar entrada dos dados no computador, depois eu os apanharei e os imprimirei na impressora lá de casa. Se você gostar, eu posso mostrar-lhe como expandir o programa para as suas necessidades específicas".

"Negócio fechado", disse Fred. "Quando começamos?".

Dois dias depois Bill trouxe seu computador portátil e instruiu a secretária como registrar os dados do estoque. Bill imprimiu a listagem em sua casa. Poucos dias depois Fred comprou um computador e Bill mostrou-lhe como obter as informações sobre o estoque, como atualizar as quantidades disponíveis assim como adicioná-las ou retirá-las do estoque. Logo, Bill adicionou um programa para emissão de etiquetas endereçadas e pouco depois Fred distribuía propaganda a seus clientes pelo correio. Fred gastou com seus programas quase $ 1000, quase o mesmo valor dos programas de estoque que havia visto e que não atendiam as suas necessidades. Bill escreveu todos os programas em uma nova linguagem desenvolvida especificamente para microcomputadores – dBASE II.

O dBASE II é comercializado como um sistema de gerenciamento de databases relacionais. Um sistema de database relacional armazena informações em tabelas de duas dimensões nas quais os dados podem ser obtidos pela relação de informação na tabela. Mais precisamente, o dBASE II é um sistema de gerenciamento de arquivos com uma avançada linguagem de programação. A linguagem é muito fácil de aprender e é "estruturada", tornando possível aos programadores desenvolver programas mais rapidamente que utilizando outra linguagem de alto nível como BASIC ou Fortran. Originalmente desenvolvida para o Projeto Viking (descida em Marte), esta linguagem de programação é disponível agora para quase todos sistemas CP/M.

O GERENCIADOR DE DATABASES

O sistema gerenciador de databases, ou SGDB, é essencialmente um grupo ou uma coleção de programas que conectam o usuário a uma ou mais coleções de informações, ou *pool*, chamadas de database (veja Figura 1-1). Programas de aplicação utilizando de um SGDB incluem: livros contábeis, contas a receber, sistemas de pesquisa e endereçamento postal, catalogação, entrada de pedidos, bibliografias – essencialmente qualquer aplicação na qual arquivos de dados são formados, atualizados, analisados e relatados.

Com a maioria dos gerenciadores de databases, os programas de aplicação estão escritos em BASIC, PASCAL, Fortran ou outra linguagem de alto nível e chamadas especiais são usadas para comunicar-se com o gerenciador de databases. O dBASE II, por sua vez, utiliza uma linguagem interna de forma que nenhuma outra linguagem é necessária.

VANTAGENS DE UM GERENCIADOR DE DATABASES

Gerenciadores de databases previnem a duplicação de dados. Com o crescimento do sistema, a informação de um arquivo logo aparecerá em outros arquivos. Nomes, endereços e outras informações similares começam a aparecer em muitos arquivos mesmo que cada arquivo contenha informações idênticas. Alguém pode entrar estes dados duplicados em cada arquivo, criando uma sobrecarga de digitação e aumentando a chance de ocorrerem erros. Usa-se memória extra em disco e as modificações devem ser feitas arquivo por arquivo. Gerenciadores de databases por armazenarem tudo em um simples database eliminam este problema.

Os gerenciadores de databases também reduzem o tempo para desenvolvimento de um programa. Muito do trabalho em rotinas de programação para gerenciamento de arquivos, indexação, classificação e geração de relatórios é feito automaticamente com simples chamadas ao

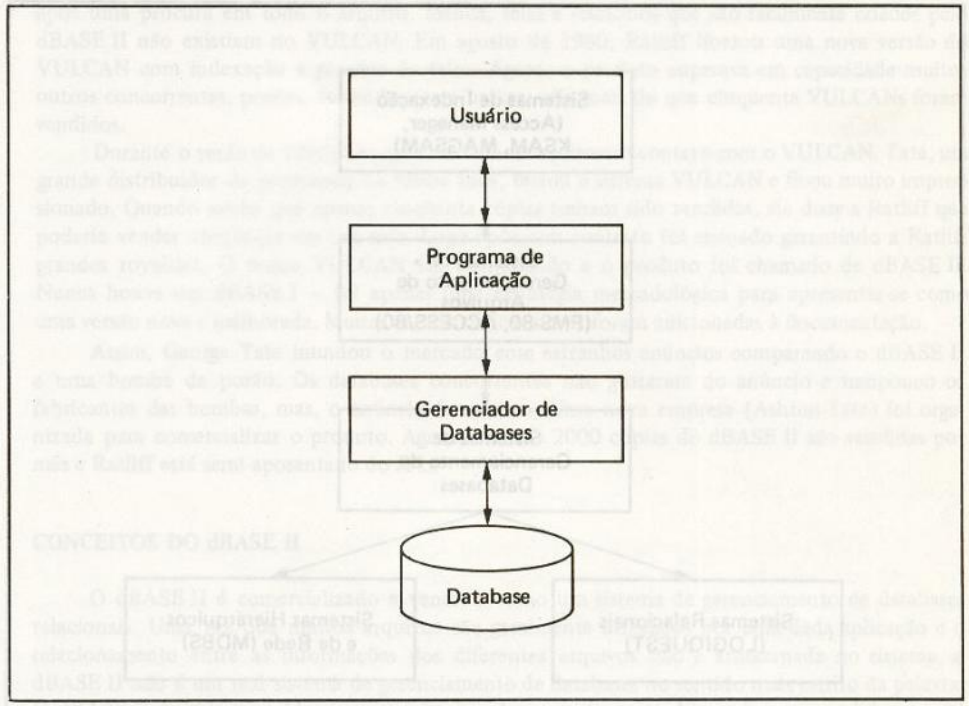


Figura 1-1. O Gerenciador de Databases

gerenciador de databases. Os programas podem facilmente ser desenvolvidos para atender as necessidades específicas do usuário.

Gerenciadores de databases aumentam a confiança nos dados. A integração e o relacionamento da informação dentro de um database é feito automaticamente pelo gerenciador de databases. Ele libera o programador da necessidade de utilização de indicadores e cadeias de indicadores para a localização da informação desejada. A confiança nos dados é aumentada. Por exemplo, se sexo é sempre "M" ou "F", o gerenciador de databases pode prevenir qualquer outro tipo de entrada. Se o código de endereçamento postal é sempre numérico, qualquer outra entrada não numérica deverá ser rejeitada.

TIPOS DE SISTEMAS DE GERENCIAMENTO DE DATABASES

Se você está interessado na compra de um sistema de gerenciamento de databases para o seu microcomputador, encontrará um número muito pequeno de sistemas gerenciadores de databases *reais* disponíveis no mercado. Como mostra a Figura 1-2, a maioria das ferramentas

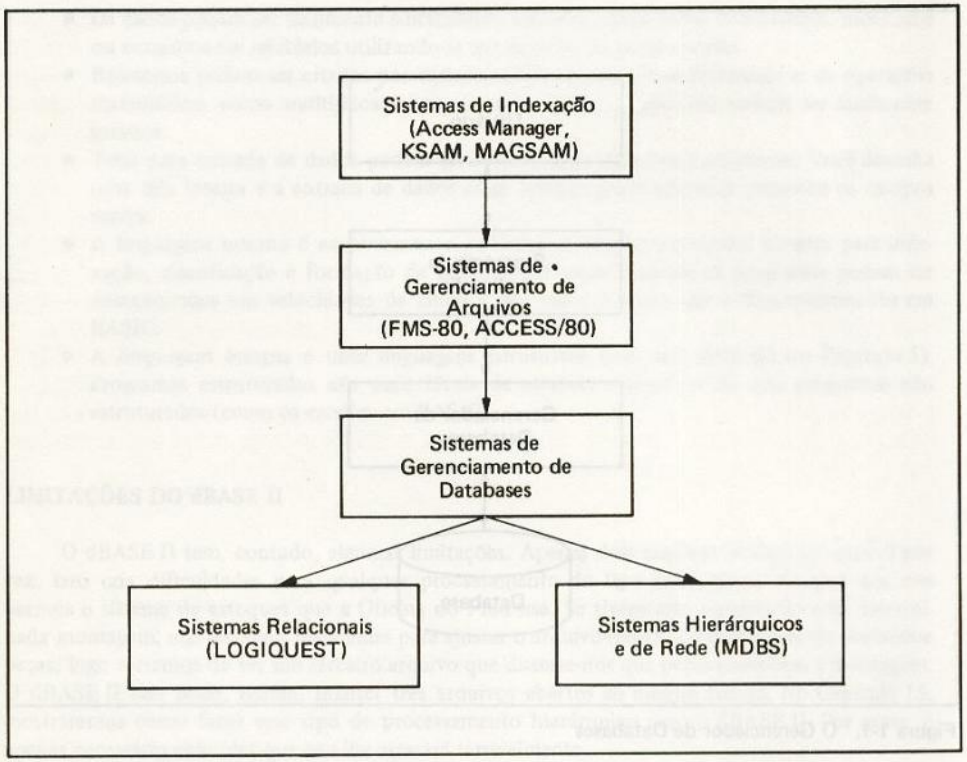


Figura 1-2. Tipos de sistemas de gerenciamento de informações

existentes para os programadores nesta área são, mais precisamente, sistemas de indexação ou sistemas de gerenciamento de arquivos.

O mais simples (e mais barato) desses sistemas são os sistemas de indexação.

Estes produtos foram concebidos para trabalhar com as linguagens de alto nível existentes (como o BASIC) para simplificar o processo de indexação. Estes produtos tornam disponível ao programador adicionar uma capacidade de indexação aos programas que antes tinham apenas possibilidade de acesso direto ou de acesso seqüencial (veja o Capítulo 3 para maiores detalhes). Contudo, a programação continua sendo feita em uma linguagem de alto nível. Exemplos desses produtos incluem MAGSAM e ACCESS MANAGER. Estes produtos são baratos, mas requerem experiência em programação.

Em um outro nível temos os gerenciadores de multi-arquivos. Eles são geralmente mais caros que os sistemas de indexação e freqüentemente incluem algum tipo de sistema de indexação interno. Os gerenciadores de multi-arquivos também incluem características extras que variam de produto para produto. Como exemplo destas características extras temos os geradores de relatórios e as rotinas de adição ou atualização de dados nos arquivos. A maioria desses produtos

pode ser utilizada por alguém que tenha pouca experiência com computadores. Como exemplo desses produtos temos o FMS-80 e o ACCESS/80.

No nível superior temos o verdadeiro sistema de gerenciamento de databases. Em um sistema de gerenciamento de databases "perfeito" o armazenamento e a atualização dos dados é transparente ao usuário. Todos os dados são armazenados em um mesmo "poço" ou database. O usuário preocupa-se apenas com o fluxo de informação. Caso o desenho do sistema seja alterado, pouca alteração será necessária nos programas existentes. Embora esse ideal nunca seja alcançado, os melhores sistemas aproximam-se dele. Todos os sistemas de gerenciamento de databases estão em uma das três categorias, hierárquicos, de rede ou relacionais. Uma vez que a categoria dos hierárquicos é um caso especial dos sistemas de rede, temos realmente dois tipos apenas. Embora uma discussão completa entre os conceitos dos sistemas de rede e dos relacionais esteja fora dos objetivos deste livro, uma rápida descrição deles é importante para se entender o dBASE II.

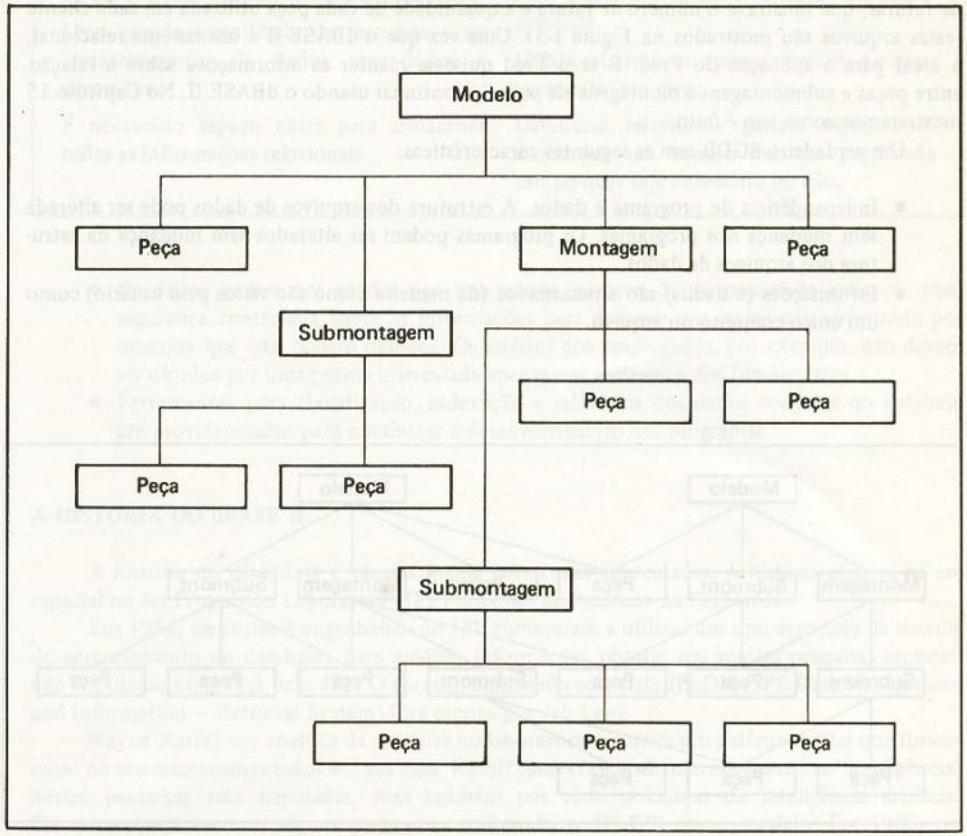


Figura 1-3. Estrutura hierárquica de dados

Em um sistema hierárquico, ou de rede, a informação é armazenada em uma estrutura que se assemelha a uma árvore invertida (veja a Figura 1-3). Com esta árvore é possível representar o sistema de estoque da Oficina de Automóveis do Fred. Um carro é composto de várias montagens e cada montagem é composta de várias submontagens. As submontagens, por sua vez, são compostas por peças. Cada peça é um "filho" de um "pai", submontagem ou montagem. Em um sistema hierárquico um "filho" não pode ter mais do que um "pai". Um sistema de rede seria similar, exceto que um "filho" pode ter mais que um "pai" (veja Figura 1-4).

Um sistema relacional é diferente em sua estrutura organizacional. As informações são armazenadas em tabelas bidimensionais que são mais convenientemente chamadas de arquivos. As informações das tabelas são obtidas pelo usuário com base em qualquer relacionamento desejado. As diferenças entre databases de rede e relacionais é mostrada na Tabela 1-1.

O sistema de estoque que Bill desenhara para a Oficina do Fred era formado de três arquivos para armazenamento das informações. Um arquivo continha todo o estoque com o número das peças, descrição, quantidade disponível, custo e preço para cada item. O segundo arquivo continha o nome, endereço e o número de identificação de cada cliente. O terceiro arquivo era o arquivo de faturas, que mostrava o número de fatura e a quantidade de cada peça utilizada em cada cliente (estes arquivos são mostrados na Figura 1-5). Uma vez que o dBASE II é um sistema relacional, é ideal para a aplicação do Fred. E se o Fred quisesse manter as informações sobre a relação entre peças e submontagens e montagens ele poderia continuar usando o dBASE II. No Capítulo 15 mostraremos como isso é feito.

Um verdadeiro SGDB tem as seguintes características:

- Independência de programa e dados. A estrutura dos arquivos de dados pode ser alterada sem mudança nos programas. Os programas podem ser alterados sem mudança na estrutura dos arquivos de dados.
- Informações (e dados) são armazenados (da maneira como são vistos pelo usuário) como um único conjunto ou arquivo.

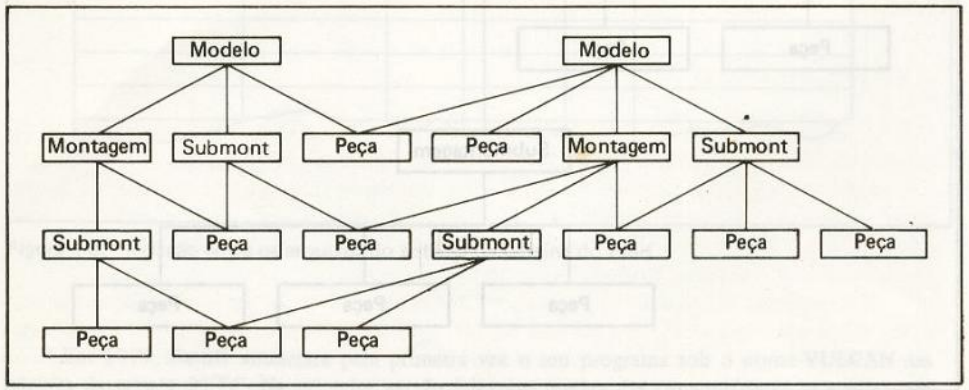


Figura 1-4. Exemplo de estrutura de dados em rede

Tabela 1-1. Diferenças entre Sistemas de Rede e Sistemas Relacionais

Databases hierárquicas e de rede	Databases Relacionais
As relações entre os itens do database são armazenadas fisicamente no database.	As relações entre os itens não são armazenadas no database e são criadas logicamente e não fisicamente.
Um relacionamento complexo entre itens que fazem parte fisicamente do database pode ser criado.	Simples de entender e de usar.
O Database não é facilmente alterado, caso se necessite de um novo relacionamento físico.	Databases podem ser facilmente alterados para atender novas configurações.
Bom desempenho em máquina se o tamanho do processador e da memória são adequados.	O desempenho da máquina varia de acordo com a maneira como a aplicação foi implementada.
Uma grande quantidade de memória e de armazenamento de dados secundários é necessária.	Utiliza relativamente pouca memória e espaço secundário em disco.
É necessário espaço extra para armazenar todas as informações relacionais.	Databases relacionais gastam espaço por armazenar um espaço máximo para cada campo quer seja necessário ou não.

- Os dados podem ser usados por um grande número de programas de aplicação com segurança controlada sobre as informações para prevenir um acesso não autorizado por usuários que não podem obtê-los. Os salários dos empregados, por exemplo, não devem ser obtidos por uma pessoa interessada apenas nos endereços dos funcionários.
- Ferramentas para classificação, indexação e relatórios dos dados contidos no database são providenciados para minimizar o desenvolvimento dos programas.

A HISTÓRIA DO dBASE II

A história do dBASE II é tão fascinante como suas capacidades. A história começa na era espacial no Jet Propulsion Laboratory (JPL) próximo de Pasadena na Califórnia.

Em 1974, cientistas e engenheiros do JPL começaram a utilizar um tipo avançado de sistema de gerenciamento de databases para analisar informações obtidas nas muitas pesquisas espaciais não tripuladas efetuadas pelo JPL. O sistema, conhecido pela sigla JPLDIS (JPL Data Management and Information – Retrieval System) fôra escrito por Jeb Long.

Wayne Ratliff era analista de sistemas no laboratório e queria um sistema similar que funcionasse no seu microcomputador em sua casa. Ratliff estava fascinado não apenas com a "inteligência" destas pesquisas não tripuladas, mas também por todo o campo da inteligência artificial. Ele começou a escrever alguns programas utilizando o JPLDIS como modelo. Em 1980, seu programa tornou-se mais poderoso que o JPLDIS. Jeb Long estava impressionado.

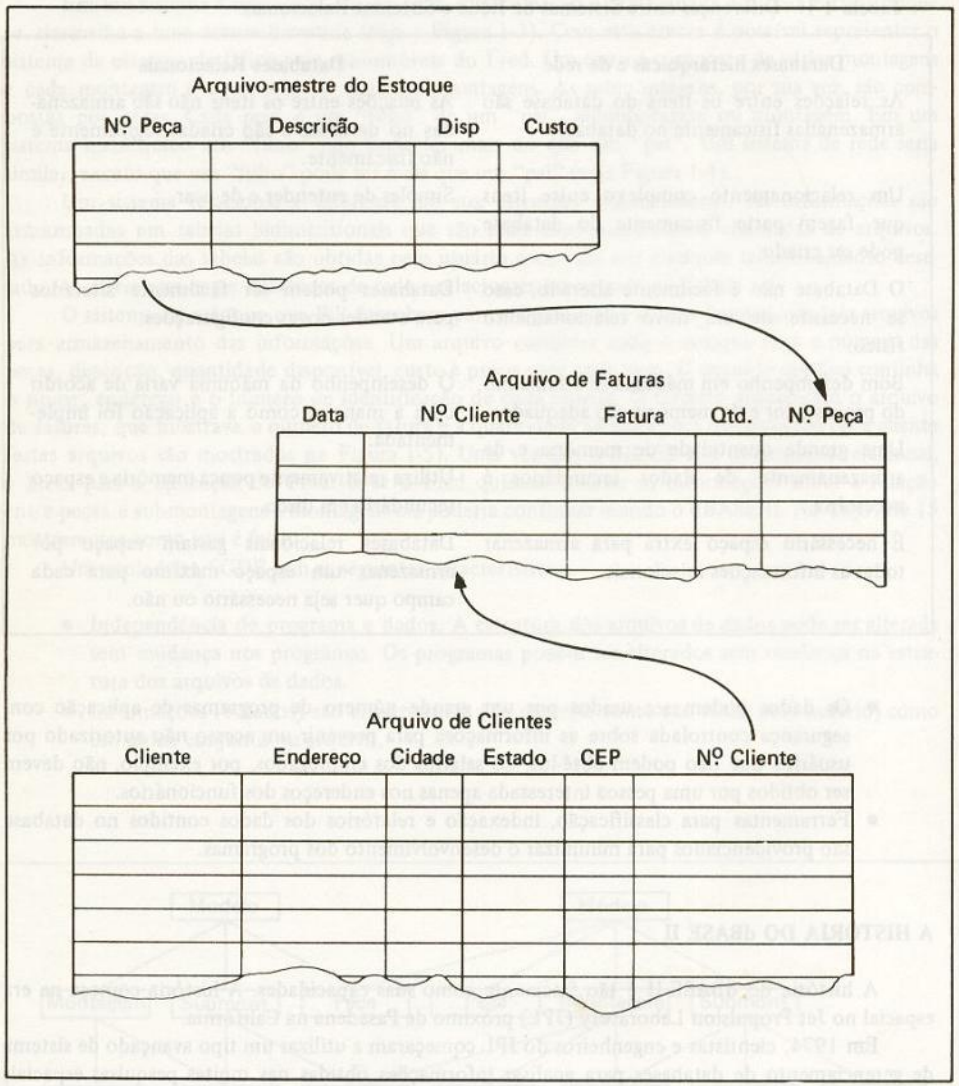


Figura 1-5. Relação entre os arquivos do sistema da Oficina do Fred

Em 1979, Ratliff anunciara pela primeira vez o seu programa sob o nome VULCAN nas páginas da revista *BYTE*. Na primeira versão faltavam muitas das características existentes nos concorrentes. Não existiam as características de indexação e os registros só podiam ser localizados após uma procura em todo o arquivo. Menus, telas e relatórios que são facilmente criados pelo dBASE II não existiam no VULCAN. Em agosto de 1980, Ratliff liberou uma nova versão do VULCAN com indexação e gerador de telas. Agora, o produto superava em capacidade muitos outros concorrentes, porém, as vendas eram baixas; não mais do que cinqüenta VULCANs foram vendidos.

Durante o verão de 1980, George Tate tomou o primeiro contato com o VULCAN. Tate, um grande distribuidor de programas há vários anos, testou o sistema VULCAN e ficou muito impressionado. Quando soube que apenas cinqüenta cópias tinham sido vendidas, ele disse a Ratliff que poderia vender cinqüenta em um mês. Logo após, um contrato foi assinado garantindo a Ratliff grandes royalties. O nome VULCAN foi abandonado e o produto foi chamado de dBASE II. Nunca houve um dBASE I – foi apenas uma estratégia mercadológica para apresentar-se como uma versão nova e melhorada. Muitas características novas foram adicionadas à documentação.

Assim, George Tate inundou o mercado com estranhos anúncios comparando o dBASE II a uma bomba de porão. Os databases concorrentes não gostaram do anúncio e tampouco os fabricantes das bombas, mas, o anúncio funcionou. Uma nova empresa (Ashton-Tate) foi organizada para comercializar o produto. Agora, mais de 2000 cópias do dBASE II são vendidas por mês e Ratliff está semi-aposentado do JPL.

CONCEITOS DO dBASE II

O dBASE II é comercializado e vendido como um sistema de gerenciamento de databases relacionais. Uma vez que muitos arquivos são geralmente utilizados por uma dada aplicação e o relacionamento entre as informações dos diferentes arquivos não é armazenada no sistema, o dBASE II não é um real sistema de gerenciamento de databases no sentido mais estrito da palavra. O dBASE II é mais parecido com um sistema de gerenciamento de arquivos ao qual foram adicionadas características relacionais. Contém uma linguagem de programação própria, permitindo ao usuário, com um mínimo de experiência, desenvolver programas extremamente complexos e poderosos que vão de encontro a aplicações como livros contábeis, custos e controle de estoques. Algumas aplicações típicas do dBASE II nos campos domésticos, manufatureiro, financeiro e comercial são:

- Contabilidade geral, contas a receber, contas a pagar, folha de pagamento, custos e livros contábeis.
- Controle de estoques, sistemas para ordens de compras, gerenciamento de projetos, escalas de pessoal, planejamento de necessidade de materiais, sistemas de faturamento e de entrada de pedidos.
- Gerenciamento de carteira de ações, jogos, simulações, análise nutricional, gerenciamento de coleções de fitas e discos, empréstimos, controle do arquivo das receitas culinárias, análise do orçamento doméstico e cálculo do imposto de renda.
- Gerenciamento do patrimônio imobiliário, endereçamento postal, faturamento de horários, personalização de grandes quantidades de correspondência.

Algumas características do dBASE II compreendem:

- Alto grau de independência de dados e programa. As estruturas de dados podem ser alteradas sem necessidade de muitas mudanças nos programas.

- Os dados podem ser facilmente adicionados, editados, eliminados, classificados, indexados ou extraídos em relatórios utilizando-se um mínimo de programação.
- Relatórios podem ser criados por dados contidos no database utilizando-se de operações matemáticas como multiplicação ou divisão. Totais e subtotais podem ser facilmente gerados.
- Telas para entrada de dados podem ser formatadas a um nível profissional. Você desenha uma tela inteira e a entrada de dados exige apenas que o operador preencha os campos vazios.
- A linguagem interna é extremamente poderosa e contém comandos simples para indexação, classificação e formação de relatórios. Freqüentemente os programas podem ser desenvolvidos em velocidades de cinco a dez vezes maiores que o desenvolvimento em BASIC.
- A linguagem interna é uma linguagem estruturada (isto será definido no Capítulo 5). Programas estruturados são mais fáceis de escrever e atualizar do que programas não estruturados (como os escritos em BASIC).

LIMITAÇÕES DO dBASE II

O dBASE II tem, contudo, algumas limitações. Apenas dois arquivos podem ser usados por vez. Isto cria dificuldades para qualquer processamento do tipo hierárquico. Imagine que nós usamos o sistema de estoques que a Oficina do Fred usa. Se tivéssemos consertado uma determinada montagem, nós teríamos problemas para ajustar o arquivo-mestre do estoque se ele contivesse peças, logo teríamos de ter um terceiro arquivo que dissesse-nos que peças compõem a montagem. O dBASE II não pode, porém, manter três arquivos abertos ao mesmo tempo. No Capítulo 15, mostraremos como fazer esse tipo de processamento hierárquico com o dBASE II. Por agora, é apenas necessário entender que isso lhe atrasará terrivelmente.

O dBASE II permitirá apenas o armazenamento de trinta e dois itens de informação em um registro. Um nome ou um endereço podem ser, cada um deles, um item. Isto é suficiente para a maioria das aplicações, mas, imagine que você precise armazenar informações médicas sobre um paciente e você tem 96 itens para armazenar. Você pode combinar dois arquivos em um único arquivo e ter 64 itens para cada parte do conjunto, mas se você fizer isso só poderá manter um arquivo aberto.

As aplicações com o dBASE II são mais vagarosas do que programas já compilados como processadores de texto ou calculadores de planilhas que você talvez esteja utilizando. O dBASE II utiliza 48.000 bytes de memória e assim há muito pouco espaço para o programa na memória. As linhas do programa são lidas e executadas, isto significa que existirão muitas operações sobre os discos durante a execução dos programas, o que torna o processamento demorado. O dBASE II só pode "conversar" com a console ou com a impressora. Se você precisar enviar informações a um computador que está distante, deve processar as informações em um arquivo separado e utilizar-se de um programa de transmissão para enviar o arquivo. Isto limita bastante as aplicações, com aplicações em tempo real, como, por exemplo, terminais em pontos de venda e rede de informações.

Com muitos processadores você terá problemas ao utilizar o dBASE II em processamento por lotes. o dBASE II faz um processamento por lotes por meio da construção de um arquivo de submissão. Como o processamento por lotes em sistemas de multi-usuários e sistemas de 16 bits

funciona de uma maneira diferente, você verá que a sua versão do dBASE II não permitirá processamento por lotes no seu computador. Mais ainda, não existe proteção contra duas pessoas atualizando o mesmo arquivo ao mesmo tempo, em um sistema multi-usuários. Existem algumas maneiras de enganar o dBASE II de forma a permitir usá-lo (com restrições) em um sistema com multi-usuários. Porém, sempre você encontrará algumas espécies de problemas. A Ashton-Tate está trabalhando no sentido de colocar no mercado uma versão do dBASE II que permite sistemas do tipo multi-usuários.

Como limitação final temos que a linguagem de programação permite apenas 64 variáveis e não dispõe de variáveis indexadas. A maioria das linguagens de alto-nível carrega informações na memória e inicia, então, seu processamento. O dBASE II por utilizar muita memória não permite que sobre muito espaço na memória para variáveis indexadas ou não. Existem maneiras que o usuário pode utilizar para criar variáveis indexadas sob o dBASE II bem como para permitir centenas de variáveis em um programa. Isto será mostrado no Capítulo 4. De qualquer maneira, isto exige algum tempo de programação e um certo trabalho em manter o número de variáveis menor do que 64 em determinados instantes de seu programa que, também, ficará mais lento.

EXIGÊNCIAS DO SISTEMA

Para usar o dBASE II você necessitará de um dos dois tipos de processadores:

- Um processador baseado em CP/M-80 utilizando-se de microprocessadores do tipo 8080, 8085 ou Z80. Estes processadores são fabricados por centenas de indústrias, como por exemplo, Compupro, Vector Graphic, Systems Group e TeleVídeo e tem um CP/M-80 de versão 2,0 ou superior, uma ou mais unidades de discos flexíveis ou discos rígidos, 48000 bytes de memória ou mais, um terminal de vídeo com endereçamento do cursor com dimensões de 24 linhas por 80 colunas e uma impressora de textos.
- Um processador baseado em CP/M-86 ou MS-DOS, como por exemplo, o IBM PC, o Rainbow-100 ou o Victor 9000. Estes processadores têm memória de 128K bytes, uma ou mais unidades de discos flexíveis ou rígidos, terminal de vídeo com endereçamento do cursor (24 linhas por 80 colunas) e uma impressora de textos.

O dBASE II opera sob os sistemas operacionais CP/M-80, CP/M-86 ou MS-DOS. Quando é vendido o dBASE II não é acompanhado de nenhum sistema operacional. É muito importante que você compre uma versão do dBASE II que corresponda ao seu sistema operacional e ao tipo de seus discos.

A Ashton-Tate dá uma das mais interessantes garantias do mercado. Se você quer testar o dBASE II em seu computador, peça-o e receberá dois discos. Um deles em um envelope lacrado, o outro, não lacrado, contém um disco de demonstração. A versão de demonstração do dBASE II tem todas as características e comandos, porém você não poderá ter mais do que quinze registros (endereços, peças ou qualquer outra coisa) no database. Você pode escrever programas, testá-los, adicionar dados no database e editá-los com a versão de demonstração. Se em trinta dias você decidir que o dBASE II não lhe interessa, devolva os discos e será totalmente reembolsado, caso você não tenha aberto o envelope lacrado que contém a versão completa.

DOIS

Instalando e Utilizando o dBASE II

O dBASE II chega em dois discos, como foi descrito no Capítulo 1. Não abra o envelope lacrado até você ter certeza de que ficará com o dBASE II. Não trabalhe com o disco de demonstração que recebeu. Ao invés disso, faça uma cópia do disco de demonstração e utilize-a. Isso necessitará de três passos:

1. Formate um disco flexível. Ele será o seu disco de trabalho.
2. Faça uma cópia do seu sistema operacional neste disco recém-formatado.
3. Copie todos os arquivos do disco de demonstração para o disco de trabalho.

Estes procedimentos variam de máquina para máquina e de sistema operacional para sistema operacional. Verifique os seus manuais e utilize-os ao criar a cópia do disco.

Não faça mais nada até que o seu disco de demonstração esteja copiado e guardado em um local seguro.

INSTALAÇÃO

Você criou uma cópia de trabalho do disco de demonstração e agora o sistema deverá ser instalado para atender as suas necessidades particulares.

O dBASE II foi desenhado para trabalhar com computadores fabricados por várias indústrias e deve ser instalado em seu computador utilizando-se de um programa que está no disco de trabalho: INSTALL. Este programa altera o dBASE II para atender as particularidades do seu terminal de vídeo. Se posteriormente você desejar mudar o terminal de vídeo, deverá reinstalar a sua cópia do dBASE II. Se você não instalar o dBASE II ou instalá-lo incorretamente terá telas impossíveis de serem lidas.

O vendedor poderá instalar o dBASE II para você. Caso isto tenha acontecido, você não precisará preocupar-se. Se você necessita instalá-lo, não precisa ter medo ou chamar um programador profissional, você mesmo pode fazê-lo em dois minutos.

Uma cópia do programa INSTALL do dBASE II deverá estar no seu disco de instalação do dBASE II. Para iniciar, coloque o disco de trabalho com a cópia do dBASE II de demonstração em seu computador e após o sinal de pronto "A >" digite o comando INSTALL e em seguida pressione a tecla RETURN.

```
INSTALL
```

Você ouvirá um "click" no disco e então aparecerá a seguinte pergunta:

```
dBASE II INSTALLATION PROGRAM VER 2.9
ARE FULL SCREEN OPERATIONS WANTED (Y/N)? Y
```

Digite um Y (sem pressionar a tecla RETURN) e um menu será mostrado contendo uma lista de todos os terminais com que o dBASE II trabalha:

```
SELECT TERMINAL TYPE
A – HAZELTINE 1500          B – SOROC, TELEVIDEO
C – HEATH 89                D – PERKIN ELMER 1100
E – ADM-3A                  F – ADM-31
G – VDP-80                  H – INTECOLOR
I – GNAT-SYSTEM 10          J – TRS-80 II (P&T)
K – APPLE II 40 COL         L – VECTOR GRAPHICS
M – SUPERBRAIN              N – VISUAL-100/VT-100
O – OSBORNE                 P – HP 2621, HP-125
Q – CROMEMCO 3102           R – TRS-80 II (FMG)
S – ADDS VIEWPOINT          T – XEROX 820
U – NEC                     V – EAGLE AVL
W – TRS-80 III
    Y – MODIFY PREVIOUS INSTALLATION
    Z – USER SUPPLIED TERMINAL COMMANDS
```

Veja qual é a letra que corresponde ao seu terminal e tecle-a.
Se você tiver a versão 2.3 do dBASE II serão apresentadas algumas perguntas sobre macros.

```
ENTER A CHARACTER TO BE USED FOR
INDICATING MACROS OR A RETURN FOR
DEFAULT CHARACTER OF AMPERSAND (&):
```

Apenas tecle RETURN, será apresentada uma segunda questão sobre o diálogo para correção de erros.

```
TYPE A RETURN IF THE ERROR CORRECTION
DIALOGUE IS TO BE USED OR ANY OTHER
KEY IF NO DIALOGUE IS WANTED:
```

Novamente, pressione RETURN. Na maioria das versões do dBASE II você verá uma pergunta sobre o tipo do seu sistema operacional.

```
ENTER OPERATING SYSTEM
  A – CP/M 1.4 OR CP/M 2.2
  B – CDOS SYSTEM
  C – CROMIX SYSTEM
  D – MP/M II SYSTEM
```

Tecle a letra correspondente ao seu sistema operacional. A pergunta final oferece uma chance de desprezar tudo o que foi feito até o momento, caso você tenha errado alguma coisa.

```
TYPE "Y" TO SAVE, ANY OTHER KEY TO ABORT INSTALL
```

Tecle Y (se você não errou nada) e a instalação estará completa.
Se você está com a nova versão 2.4, após a escolha do tipo do terminal, você terá apenas uma pergunta:

```
CHANGE MACRO, DATE, ETC. (Y/N)?
```

Se você está usando sistemas de um único usuário, tecle N. Esta pergunta dá a chance de pular as três primeiras perguntas sobre macros, correção de erros e seleção de sistema fazendo com que você instale o dBASE II imediatamente. Caso você teclar Y (se você tem sistemas com multi-usuários), serão apresentadas as mesmas perguntas da versão 2.3.

APÓS O DISCO DE DEMONSTRAÇÃO

Agora o disco de trabalho está pronto para ser usado em seu computador. Como você deve ter lido neste livro, deve saber que poderá utilizar a cópia de trabalho do disco de demonstração durante várias semanas. Comece a aprender o dBASE II com esta cópia. Mais tarde neste livro (Capítulo 3) você verá quais são as limitações dela.

Com este disco você pode escrever e testar programas. Depois, após familiarizar-se com o dBASE II e ter decidido a mantê-lo, abra o envelope com o disco mestre e repita as operações de cópia e instalação. Uma vez que você iniciou a utilização da cópia do disco mestre, identifique as cópias de trabalho com as etiquetas de copyright da Ashton-Tate para o dBASE II e guarde os discos originais.

UTILIZANDO O HELP

Se você está com uma versão do dBASE II 2.4 ou posterior, poderá ter uma explicação sobre qualquer comando sem a necessidade de utilizar o manual. Por exemplo, para saber detalhes do comando APPEND, entre o seguinte comando:

```
HELP APPEND
```

Após a última linha da mensagem de explicação, o sistema necessitará por volta de 15 segundos para pesquisar o restante do arquivo.

Para obter informações sobre as novidades de sua versão do dBASE II, entre com o comando

```
HELP  NEW
```

O HELP é um grande economizador de tempo quando alguém "deu uma emprestadinha" no seu manual. Algumas explicações são maiores que uma tela. A mensagem "WAITING" aparecerá após cada tela cheia; pressionando qualquer tecla você passará para a próxima tela de informação.

UTILIZANDO O dBASE II

Para utilizar o dBASE II a partir do seu sistema operacional entre com o comando

```
dBASE
```

Você será solicitado a entrar com a data corrente

```
ENTER TODAYS DATE OR RETURN FOR NONE
(MM/DD/YY) :
```

Entre com a data. Se a data for válida, você receberá a mensagem de identificação. A próxima linha terá apenas um ponto que é o sinal de pronto do dBASE

```
*** dBASE II  Ver 2.4X40  4 Nov 1982
```

Você poderá entrar com comandos ao sinal de pronto que serão executados imediatamente. Tente colocar o seu nome por exemplo:

```
? 'JOHN'
JOHN
```

Tente uma operação matemática como esta:

```
? 2+4
6
```

Para sair do dBASE, entre com o comando QUIT:

```
QUIT
```

Os comandos são sempre entrados ao sinal de pronto (o ponto). Como você verá pelo livro, comandos são verbos, usualmente seguidos por algum texto de forma que o dBASE II sabe o que é necessário fazer.

Se você fizer algo errado ou desejar parar qualquer coisa que o dBASE II esteja fazendo, simplesmente aperte a tecla ESCAPE. Ela é a mesma coisa que um botão de emergência e traz de volta o sinal de pronto do dBASE II.

TRÊS

Arquivos, Registros e Databases

Um computador é utilizado para armazenar e processar informações. Normalmente, nós utilizamos o computador para armazenar informações sobre alguma coisa que existe no mundo real. Isto é conhecido como uma entidade, que pode ser os livros de sua biblioteca, as árvores do noroeste ou o estoque de uma empresa. Qualidades selecionadas e quantidades de uma entidade são selecionadas para um nível não-físico ou um nível matemático. Estas qualidades e quantidades são conhecidas como informações. Porém, esse nível é sempre uma abstração de forma que nós não podemos, por exemplo, armazenar tudo sobre os livros de nossa biblioteca neste mundo representativo. A informação que nós abstraímos pode ser armazenada em um sistema de computador para análise ou processamento. O computador utiliza bits e bytes para representar as informações, e a representação das informações no computador é conhecida como dados (veja a Figura 3-1).

ABRINDO E FECHANDO ARQUIVOS

Os dados são armazenados em arquivos e os arquivos devem ser abertos antes que possam ser utilizados. Quando um arquivo é aberto, é criada pelo sistema operacional uma área temporária na memória de forma que o sistema operacional possa transmitir dados da memória interna para a memória em disco e vice-versa. Certos conjuntos de informações são colocados na memória do computador possibilitando-o a saber onde os dados serão armazenados no disco. Logo, o arquivo estará pronto para ser usado.

Quando o usuário termina de utilizar um arquivo, este deve ser fechado. Os conjuntos de informações sobre o arquivo são retirados para o disco e a área de transmissão temporária de dados é liberada da memória de forma a ser utilizada por outros arquivos. O disco flexível que esteja sendo atualizado nunca deverá ser removido do computador antes de ter todos os seus arquivos fechados. Se o disco for removido, informações serão perdidas, uma vez que alguns dados talvez não tenham sido registrados. Com o dBASE II isto pode causar sérios problemas. Nós lhe mostraremos como prevenir isto,nos próximos capítulos.

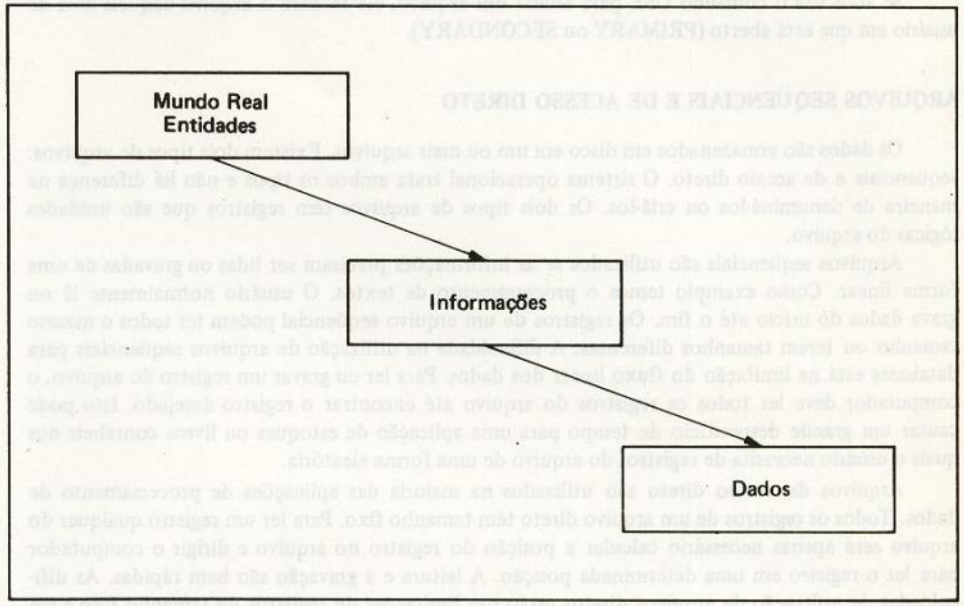


Figura 3-1. Relacionamento entre o mundo real e os dados

O dBASE II pode manter dois arquivos abertos por vez. Eles podem ser considerados como duas áreas de playground disponíveis para o usuário. O primeiro playground é a área PRIMARY e o segundo a área SECONDARY. O usuário pode mover-se para qualquer dos playgrounds pelo comando SELECT. Os dados também podem ser movimentados entre as duas áreas. Por exemplo:

```
SELECT PRIMARY
USE MESTRE
SELECT SECONDARY
USE LOTE
```

serão abertos dois arquivos (MESTRE e LOTE) e o usuário estará na segunda área.

Com o dBASE II a maneira mais simples de fechar todos os arquivos é pelo comando CLEAR. Um arquivo correntemente selecionado em uma área também pode ser fechado utilizando-se do comando USE sem nenhum nome de arquivo. Por exemplo:

```
USE
```

De uma outra forma, abrindo-se outro arquivo, o anterior que estava sendo usado será fechado, como se segue:

```
USE LOTE
```

Se você usa o comando USE para fechar um arquivo, ele fechará o arquivo daquela área de usuário em que está aberto (PRIMARY ou SECONDARY).

ARQUIVOS SEQÜENCIAIS E DE ACESSO DIRETO

Os dados são armazenados em disco em um ou mais arquivos. Existem dois tipos de arquivos: seqüenciais e de acesso direto. O sistema operacional trata ambos os tipos e não há diferença na maneira de denominá-los ou criá-los. Os dois tipos de arquivos têm registros que são unidades lógicas do arquivo.

Arquivos seqüenciais são utilizados se as informações precisam ser lidas ou gravadas de uma forma linear. Como exemplo temos o processamento de textos. O usuário normalmente lê ou grava dados do início até o fim. Os registros de um arquivo seqüencial podem ter todos o mesmo tamanho ou terem tamanhos diferentes. A dificuldade na utilização de arquivos seqüenciais para databases está na limitação do fluxo linear dos dados. Para ler ou gravar um registro do arquivo, o computador deve ler todos os registros do arquivo até encontrar o registro desejado. Isto pode causar um grande desperdício de tempo para uma aplicação de estoques ou livros contábeis nos quais o usuário necessita de registros do arquivo de uma forma aleatória.

Arquivos de acesso direto são utilizados na maioria das aplicações de processamento de dados. Todos os registros de um arquivo direto têm tamanho fixo. Para ler um registro qualquer do arquivo será apenas necessário calcular a posição do registro no arquivo e dirigir o computador para ler o registro em uma determinada posição. A leitura e a gravação são bem rápidas. As dificuldades de utilização de arquivos diretos estão nas limitações de registros de tamanho fixo e em um trabalho extra de programação envolvendo não só programação, mas também em manter uma seqüência da ordem dos registros.

Você pode, por exemplo, manter uma lista de associados utilizando-se de um arquivo direto. O número de sócio de cada pessoa pode representar o número do registro no arquivo. Para ter dados de uma certa pessoa é necessário apenas entrar com o número do sócio. O programa estará apto a encontrar as informações sobre a pessoa imediatamente.

Imagine que você tem uma lista de endereços e que deseja recuperar informações por nomes ou por CEPs. O dBASE II utiliza registros de tamanho fixo e cada registro contém campos definidos por nome, tamanho e tipo. Um arquivo (ou arquivos) separado de índice pode ser criado pelo usuário para diminuir o tempo de acesso a um determinado registro do arquivo com base no valor de um determinado campo. Por exemplo, um índice por sobrenomes permitirá ao usuário encontrar um registro específico de um arquivo por um dado sobrenome. Uma vez que o dBASE II está procurando por meio de um índice o acesso ao arquivo será mais rápido do que procurar por todo o database.

Com a maioria das linguagens para computadores você necessita de um sistema de indexação externo ligado ao seu programa. Muitos fabricantes desenvolveram ferramentas de indexação de informações que trabalham com a maioria das linguagens de alto nível. Por exemplo, a Digital Research vende o ACCESS MANAGER que trabalha com CBASIC, PASCAL-MT+ e PL/I. São ferramentas importantes, mas requerem experiência em programação e tempo de desenvolvimento para utilizá-las. Uma outra forma de se ver o armazenamento de informações é observar a informação armazenada em uma tabela bidimensional na qual os registros são linhas e os campos são colunas (como mostra a Figura 3-2). As informações podem ser recuperadas pela especificação de qualquer relacionamento entre as colunas e as linhas como, por exemplo, "encontre o primeiro sócio cujo último nome seja Thomas" ou "encontre todas as peças de custo maior que $ 100".

	Campos				
	Nome	Endereço	Cidade	Estado	CEP
Registro 1	Jones, Bill	3309 Main St.	New York	NY	10001
Registro 2	Morrow, Jane	129 Oak St.	Los Angeles	CA	94131
	.	.	.	.	.
	.	.	.	.	.
	.	.	.	.	.

Figura 3-2. Registros e campos

DESENVOLVENDO O DATABASE

O primeiro passo ao desenhar um sistema com o dBASE II é desenhar o database. Que informações você necessita para tomar decisões? Que informações você precisa nos relatórios? Qual a freqüência desses relatórios? Qual o formato dos relatórios? Pegue alguns exemplos de relatórios com os usuários. Você precisará definir as telas, as informações que necessitará nas telas e também de que forma pesquisará o database.

Após esta saída estar definida, você pode começar a definir as informações de entrada para obter esta saída. Você também precisará decidir como organizar estas informações em arquivos. A Oficina do Fred necessitava de um arquivo com todas as peças do estoque. Para cada peça Fred necessitava obter relatórios sobre a quantidade disponível, o preço e o valor total do estoque (o que requer as informações sobre o custo e sobre a quantidade disponível).

Para cada peça, Fred precisava saber:

- O número da peça
- A descrição da peça
- Notas (material ou referência de montagem)
- A quantidade disponível
- O custo da peça
- O preço da peça

Em um sistema de databases, cada item (ou uma peça do estoque neste exemplo) é armazenado em um registro. Cada registro, por sua vez, é composto de seis campos como foi mostrado. A cada campo é dado um nome que diz que informações ele contém. Se pensarmos nestas informações como se estivessem armazenadas em uma tabela, as peças do estoque seriam as linhas e os campos seriam as colunas neste exemplo (veja a Figura 3-3).

No dBASE II, os nomes dos campos devem ter dez caracteres ou menos. Espaços não podem ser utilizados em nome de campos e cada nome deve começar com uma letra. Vamos dar nomes aos campos do nosso exemplo:

- Número da peça = NOPEÇA
- Descrição = DESC

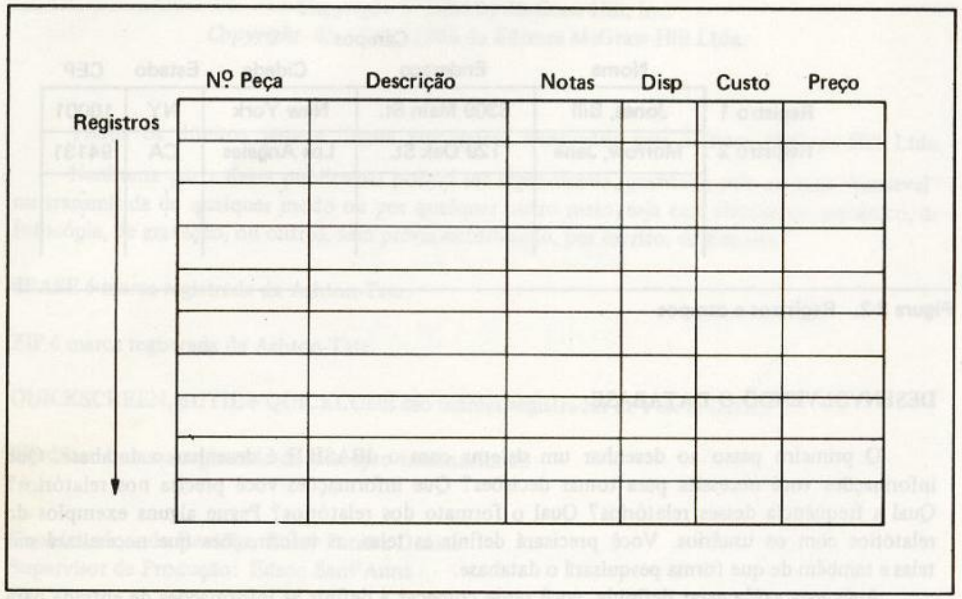


Figura 3-3. O estoque em forma de tabela

- Notas = NOTAS
- Quantidade disponível = DISP
- Custo da peça = CUSTO
- Preço da peça = PREÇO

Para cada campo deve ser definido o tipo de dados que contém. O campo poderá ser de um dos seguintes três tipos: de caracteres, numérico ou lógico. Um campo de caracteres é composto por uma ou mais letras, números ou sinais de pontuação; este tipo de campo é ideal para a descrição da peça. Um campo *numérico* contém apenas números, como o custo de uma peça. Um campo lógico contém um de dois valores possíveis: ou é verdadeiro ou falso.

Fred designou no seu sistema os campos NOPEÇA, DESC e NOTAS como de caracteres e os campos DISP, CUSTO e PREÇO como tipo numérico.

O próximo passo é designar a cada campo um tamanho. Você deve dar o tamanho máximo que se pode esperar para aquele dado. A maior descrição de uma peça no estoque do Fred tem 32 caracteres. Todos os números de peças têm seis caracteres. Nos campos numéricos você deve dar o tamanho máximo do campo levando em conta o ponto decimal; você também deve especificar o número de casas decimais. Por exemplo, o maior número no campo CUSTO será 9999.99, assim nós podemos designá-lo como de tamanho 7 e 2 casas decimais.

Os campos estão listados na Tabela 3-1.

Quanto maior os campos, maior será o espaço a ser utilizado no disco. Você pode calcular o tamanho de um registro pela soma do tamanho de cada campo, o que no nosso exemplo é de

Tabela 3-1. Campos do arquivo de estoque

Nome	Tipo	Tamanho	Casas Decimais
NOPEÇA	C	6	
DESC	C	32	
NOTAS	C	40	
DISP	N	5	0
CUSTO	N	7	2
PREÇO	N	8	2

98 caracteres. Dividindo-se o tamanho de sua memória em disco pelo tamanho do registro você terá quantos registros existirão em um disco. Se na Oficina do Fred usar discos de 250.000 bytes, teremos 2551 peças (25000 ÷ 98) no disco, caso não existam nele outros arquivos ou outras informações. Porém, a memória disponível será menor do que isso, pois, o dBASE II criará um campo inicial de informações que será mantido no arquivo.

CRIANDO O DATABASE

Com o dBASE II, na criação de um arquivo, informações sobre este arquivo, como nome de campos e seus tipos, são definidas. Estas informações são colocadas no início do arquivo constituindo um "cabeçalho" para este arquivo. Após a sua criação, todas as vezes que o arquivo for utilizado, o dBASE II obterá automaticamente informações desse "cabeçalho", o que significa que o usuário nunca mais precisará definir como as informações estão armazenadas. O dBASE II já as conhece. Mais do que isso, torna-se muito fácil para o usuário alterar campos, bem como as aplicações. Campos poderão ser incluídos, alterados ou eliminados do arquivo por um trabalho mínimo e poucas modificações nos programas. Com linguagens como o BASIC, se você modifica um campo, todos os programas que usam o arquivo devem ser modificados, o que, freqüentemente, exigirá grandes mudanças. Isto não é válido com o dBASE II. Os programas podem evoluir dinâmica e naturalmente para atingir suas necessidades assim como as necessidades de mudanças.

Criar um database com o dBASE II é muito fácil. Nós já temos definidos os campos necessários, bem como o tamanho e o tipo de cada campo. Estas definições sobre o arquivo serão determinadas pela aplicação e pelo programa. Agora você precisa criar o database com um índice.

Uma vez que você está no dBASE II (o ponto do sinal de pronto está na tela), entre com o comando

```
CREATE MESTRE
```

CREATE é um verbo – e é o comando. MESTRE é o nome do arquivo que você vai criar. Se você não especificar um segundo nome ao denominar um arquivo do dBASE II, ele designará DBF como o segundo nome do arquivo. No nosso caso, o nome completo que existirá no disco, será o que foi automaticamente designado como

```
MESTRE.DBF
```

Agora vamos definir os campos. O dBASE II mostrará o número do campo indicando estar pronto para receber informações. Você entrará com o nome do campo, o seu tipo, o seu comprimento e com o número de casas decimais, caso ele seja numérico. O sinal de pronto para receber a primeira definição de campo é o seguinte:

```
ENTER RECORD STRUCTURE AS FOLLOWS:
  FIELD  NAME,TYPE,WIDTH,DECIMAL PLACES
  001
```

Ao completarmos a descrição dos campos do exemplo da Oficina do Fred teremos o seguinte:

```
ENTER RECORD STRUCTURE AS FOLLOWS:
  FIELD  NAME,TYPE,WIDTH,DECIMAL PLACES
  001    NOPECA,C,6
  002    DESC,C,32
  003    NOTAS,C,40
  004    DISP,N,5
  005    CUSTO,N,7,2
  006    PRECO,N,8,2
  007
INPUT DATA NOW? N
```

Se você entrou com alguma coisa errada (como um tamanho muito grande ou brancos no nome do campo), o dBASE II rejeitará a linha dando-lhe a oportunidade de redigitá-la.

Após ter entrado o seu último campo, o dBASE II estará mostrando o número para o próximo campo que se seguiria. Tecle RETURN. O dBASE II irá perguntar-lhe se você deseja entrar com os dados para o arquivo. Tecle N. Assim, o dBASE II sairá da rotina de criação e mostrará o sinal de pronto.

Caso você tente criar um arquivo que já existe, o dBASE II irá preveni-lo de que o arquivo já existe e dará a você a chance de abortar esta criação de arquivo.

Uma vez criado, a estrutura do arquivo poderá ser mostrada a qualquer hora. Abra o arquivo com o comando USE e então entre com o comando DISPLAY STRUCTURE:

```
STRUCTURE FOR FILE: A:MESTRE .DBF
NUMBER OF RECORDS:   00010
DATE OF LAST UPDATE: 01/01/80
PRIMARY USE DATABASE
FLD      NAME       TYPE WIDTH   DEC
001      NOPECA     C    006
002      DESC       C    032
003      NOTAS      C    040
004      DISP       N    005
005      CUSTO      N    007     002
006      PRECO      N    008     002
** TOTAL **              00099
```


Isto lhe dará o nome, o tipo e o tamanho de cada campo. O número de registros existentes no arquivo também é mostrado, bem como a data em que o arquivo sofreu a última atualização.

Experimente obter a estrutura do arquivo que você acabou de criar. Agora tente o seguinte:

1. Experimente criar novamente um arquivo de nome MESTRE. O que aconteceu? (Responda N à pergunta).
2. Entre com o comando CREATE sem o nome do arquivo. O que aconteceu?

INDEXANDO O DATABASE

Quando você deseja obter alguma informação do database ou atualizar os valores de uma peça, existem vários caminhos para localizar as informações da peça sem ter que percorrer o database por inteiro. É a mesma coisa que tentar encontrar alguma coisa em um livro, a estratégia é a mesma – nós utilizamos um índice. O índice é um arquivo separado, criado para conter informações de onde as coisas estão localizadas no arquivo de estoque.

Os registros no arquivo de dados (MESTRE.DBF) estão armazenados na ordem em que foram colocados. Para encontrar o último registro que você colocou, deveria ler cada registro do database. Utilizando-se de um índice, você poderia localizar qualquer registro do arquivo em menos de um segundo.

Um índice pode ser criado a partir de um campo, de uma combinação de campos ou de parte de campos. Em outras palavras, você pode criar um índice do database a qualquer hora em qualquer ordem que desejar. Esta ordem será definida pela chave.

Se a chave de cada registro é única e não é utilizada por nenhum outro item do database, ela é chamada de chave *primária*. Se a chave não é única, ela é chamada de chave *secundária*. (NOTA: tenha certeza de não fazer confusão com o conceito de chaves primárias e secundárias com as áreas de usuário que são a primária e a secundária como mencionado anteriormente). No nosso exemplo do estoque da oficina, uma peça é um número único e é a chave primária. Se tivéssemos um database composto dos sócios de um clube, o número do sócio seria uma chave primária e o nome e o CEP poderiam ser chaves secundárias (o sobrenome poderia ser definido como um campo separado do nome permitindo-se uma ordem alfabética dos sobrenomes).

Ao criarmos um database, nós normalmente teremos uma chave primária e um número variável de chaves secundárias. Os índices podem ser atualizados automaticamente quando os registros são adicionados, retirados ou criados temporariamente por ocasião da impressão de relatórios. O melhor é definir apenas um índice que seja automaticamente atualizado. Se tivermos muitos índices sendo atualizados automaticamente, o tempo de resposta do sistema será dramaticamente prejudicado.

Para indexar o estoque do Fred, nós poderíamos entrar com

```
USE MESTRE
INDEX ON NOPECA TO MESTRE
```

Isto criaria um índice pelo número das peças. O arquivo poderia ser processado a qualquer hora pelo número da peça, caso fosse aberto pelo comando

```
USE MESTRE INDEX MESTRE
```

ALTERANDO A ESTRUTURA DOS DADOS

Talvez você necessite alterar a estrutura dos dados do database que já tenha definido. Caso no arquivo existam informações que você deseja guardar, necessitará usar os procedimentos descritos no Capítulo 12. Se nada foi armazenado no database, o comando MODIFY poderá ser usado para alterar a estrutura. Por exemplo, para modificar o arquivo MESTRE, entre com o comando

```
MODIFY STRUCTURE
```

Primeiro, o dBASE II advertirá você que todos os registros serão destruídos. Se você apenas definiu a estrutura e não carregou nada no database, tudo estará bem.

Após a confirmação, a tela será limpa e a estrutura atual mostrada:

```
          NAME      TYP  LEN  DEC
FIELD 01  :NOPECA    C   006  000  :
FIELD 02  :DESC      C   032  000  :
FIELD 03  :NOTAS     C   040  000  :
FIELD 04  :DISP      N   005  000  :
FIELD 05  :CUSTO     N   007  002  :
FIELD 06  :PRECO     N   008  002  :
FIELD 07  :                        :
FIELD 08  :                        :
FIELD 09  :                        :
.
.
.
```

Para alterar qualquer nome, tipo ou tamanho, simplesmente escreva sobre os valores existentes. Os demais comandos disponíveis estão descritos na Tabela 3-2.

Tabela 3-2. Códigos de controle utilizados no comando MODIFY

CONTROL-T	Elimina o campo em que o cursor está posicionado e move para a posição o campo que está abaixo.
CONTROL-N	Move para baixo todos os campos abaixo da posição do cursor permitindo ao usuário inserir uma nova definição de campo no local.
CONTROL-Y	Limpa o campo com brancos sem mover nenhum outro.
CONTROL-Q	Término sem alterar a definição deixando a estrutura do database em sua forma original sem nenhum registro entrado.
CONTROL-W	Término alterando a estrutura para a nova definição.

Experimente modificar e alterar campos no database que você criou. Após ter terminado, saia do comando MODIFY com o comando *CONTROL-Q* ou *CONTROL-W*.

TIPOS DE ARQUIVO

O dBASE II utiliza sete tipos diferentes de arquivos. Cada arquivo tem um nome que pode ser designado pelo usuário. O nome consiste de uma parte primária e de uma parte secundária, separadas por um ponto como MESTRE.DBF. O nome da primeira parte é dado pelo usuário. O nome da parte secundária é determinado pelo modo que o arquivo será utilizado.

O dBASE II é muito generoso e permite a você utilizar nomes longos nos arquivos (oito) caracteres) e caracteres especiais. Contudo, isto poderá causar problemas, caso você planeje utilizar com estes arquivos utilitários que fazem parte do seu sistema operacional. O mais seguro é limitar seus nomes primários a oito caracteres e evitar qualquer caractere especial como pontos ou dois pontos.

O segundo nome designado pelo dBASE II representa o tipo do arquivo. Os tipos de arquivo estão descritos na Tabela 3-3.

Tabela 3-3. Tipo de arquivo utilizado pelo dBASE II

.DBF	Arquivos databases utilizados para armazenar informações. Eles são especialmente formatados e não devem ser alterados ou modificados por programa que não o dBASE II (como processadores de texto, por exemplo).
.NDX	Arquivos índices para arquivos DBF criados pelo comando INDEX.
.CMD	Arquivos de programas ou arquivos de comandos (só para CP/M-80). Eles contêm uma seqüência de instruções de programa que executam uma função desejada.
.PRG	Arquivos de programas ou arquivos de comandos (CP/M-86 e MS-DOS). O mesmo que os arquivos .CMD.
.FRM	Arquivo de modelos do gerador de relatórios. É utilizado pelo gerador de relatórios para criação de relatórios.
.FMT	Arquivo de formatos. Utilizado na criação de telas para que sejam utilizados por vários programas.
.MEM	Arquivos de memória auxiliar. Usados temporariamente no armazenamento de informações variáveis.
.TXT	Arquivos de texto ou de registro. Podem ser criados para registrar e guardar qualquer coisa enviada à tela ou à impressora.

QUATRO

Introdução à Programação

Até o momento, vimos instruções individuais, como CREATE e MODIFY STRUCTURE, que são entradas no computador e executadas imediatamente. Uma seqüência de instruções para o computador pode ser escrita e colocada em um arquivo e então executada várias vezes, bastando chamar aquele arquivo. Esta seqüência de instruções é chamada programa. Existem várias linguagens de programação como BASIC, COBOL, Pascal e FORTRAN cada uma com finalidades diferentes. Elas são conhecidas como *linguagens de alto nível*, pois o computador traduzirá cada linha de instrução entrada pelo usuário em mais de uma instrução em linguagem de máquina. O dBASE II utiliza sua própria linguagem de alto nível para definir as operações que serão executadas. Enquanto similar ao BASIC e a outras linguagens, ela é uma linguagem de alto nível extremamente poderosa que permite ao usuário executar procedimentos que exigiriam muitas linhas de codificação BASIC ou FORTRAN. E como qualquer linguagem, a do dBASE II tem certas regras de gramática e sintaxe que devem ser seguidas pelo usuário.

Como na maioria das linguagens, as instruções podem ser executadas de um modo direto, de um modo indireto ou "em lotes". No modo direto, o usuário pode escrever a linha de instruções e imediatamente elas serão executadas. Um exemplo no dBASE II seria o uso da instrução.

```
? "TESTE"
```

que imediatamente mostraria a palavra TESTE. Se a linguagem for usada de um modo indireto, uma seqüência de instruções será colocada em um arquivo e então executadas seqüencialmente sem intervenção do operador.

O dBASE II permite ao usuário utilizar-se tanto do modo direto como do indireto para executar as instruções. Geralmente as idéias que você tem são testadas no modo direto. Os programas são executados no modo indireto. O modo direto também é bem prático quando se quer fazer uma pesquisa rápida em um arquivo ou para emitir um relatório imprevisto.

VARIÁVEIS

Uma variável em um programa de computador é definida como uma posição de memória que pode ser usada para guardar dados. O conteúdo desta posição de memória (e o valor da variável) pode ser alterado durante a execução do programa. Como exemplo, nós poderíamos dar o comando

```
STORE 80 TO velimite
```

VELIMITE é a variável do exemplo. Ela representa uma posição na memória do computador utilizada para armazenar uma informação. O valor 80 foi armazenado temporariamente nessa posição.

No dBASE II são três os tipos de variáveis disponíveis:

- Quantidades numéricas, inteiras ou com decimais. Porém, o dBASE II armazena todos os números com ponto decimal.
- Conjunto de caracteres, que são variáveis formadas de uma ou mais letras como "JOÃO", "CAIXA 475" etc.
- Lógicas, variáveis que têm um valor verdadeiro ou falso.

Conjuntos de caracteres devem ser definidos entre apóstrofos, aspas ou parênteses. Por exemplo:

```
STORE "TESTE DO SISTEMA" TO título
```

e

```
STORE 'TESTE DO SISTEMA' TO título
```

são permitidas, mas

```
STORE 'TESTE DO SISTEMA" TO título
```

fará com que apareça uma mensagem de erro.

Na maioria das aplicações você pode usar tanto aspas como apóstrofos, mas lembre-se de ser coerente. Se você precisa usar apóstrofos ou aspas no seu conjunto de caracteres, utilize o caractere alternado para separar o conjunto. Por exemplo, se deseja usar apóstrofos no texto, você poderá usar aspas na definição total do texto, assim

```
STORE "Contas d'Água" TO título
```

As variáveis lógicas podem ser definidas como T, t, Y ou y para valores verdadeiros e F, f, N ou n para valores falsos.

Existem duas classes de variáveis que podem ser utilizadas no dBASE II. *Variáveis de memória* residem na memória do computador e podem ser alteradas pelo programa durante sua execução. Por exemplo:

```
STORE 4 TO x
```

Estas variáveis (X neste caso) são alteradas com comandos do tipo STORE, COUNT, SUM, WAIT e INPUT.

Campos de dados variáveis são nomes de campos utilizados nas partes dos registros do arquivo. Eles são alterados pelo comando REPLACE. Alguns comandos, como o comando GET, podem alterar tanto a memória como o campo de dados variáveis.

Apenas 64 variáveis de memória podem ser definidas por vez e não são permitidas variáveis indexadas. Isto se deve ao fato do dBASE II utilizar muita memória (48K), não deixando muito espaço livre para memória variável.

Existem várias maneiras de se contornar este problema. A primeira regra é eliminar as variáveis de memória que não mais estejam sendo utilizadas. O comando CLEAR deve ser usado após a execução de qualquer programa para eliminar todas as variáveis utilizadas no programa. Quando o programa estiver executando, o comando RELEASE poderá ser usado para eliminar algumas variáveis abrindo espaço para novas variáveis. Por exemplo,

```
RELEASE título, campo
```

liberará o espaço utilizado pelas variáveis TÍTULO e CAMPO permitindo que duas novas variáveis sejam definidas. A qualquer momento o comando DISPLAY MEMORY poderá ser usado para mostrar todas as variáveis definidas até aquele ponto.

Há uma outra técnica de se evitar um grande número de variáveis. Todas as variáveis de memória existentes podem ser salvas em um arquivo pelo comando SAVE. Por exemplo,

```
SAVE TO temp
```

salvará todas as variáveis no arquivo TEMP.MEM. Estas variáveis poderão ser recuperadas a qualquer hora através do comando RESTORE:

```
RESTORE FROM temp
```

Se você tem certas variáveis que são usadas na maioria dos seus programas (como o nome de empresas, por exemplo), o melhor é armazená-las em um arquivo MEM e posteriormente chamá-las com o comando RESTORE após limpar o "lixo" das variáveis utilizadas no programa anterior. Escrevendo

```
RESTORE FROM título
```

automaticamente serão eliminadas da memória todas variáveis existentes e as variáveis do arquivo TÍTULO.MEM serão carregadas.

As novas versões do dBASE II (versão 2.4 e posteriores) têm grandes variações dos comandos SAVE, RESTORE e RELEASE. Agora é possível usar o comando RESTORE sem eliminar as variáveis existentes pelo uso da opção ADDITIVE. Os comandos SAVE e RELEASE agora contam com as opções ALL LIKE e ALL EXCEPT. Nestas opções, é especificado um "modelo" de nome de variável e serão salvas ou liberadas apenas as variáveis que são iguais ao modelo. O modelo utiliza os sinais "*" e "?" para testar os nomes. Por exemplo, escrevendo SAVE ALL LIKE 'cont' serão salvas apenas as variáveis cujos nomes comecem com as letras 'cont'.

O nome de uma variável precedido por um "&" é interpretado pelo dBASE II como significando o conteúdo da variável. Você pode calcular o nome da variável (na memória ou campo de dados) e armazenar a quantia nesta variável. Por exemplo, na seqüência de instruções


```
STORE 'LOTE' TO arqm
USE &arqm
```

Criará uma variável chamada ARQM e o conjunto de caracteres "LOTE" será armazenado nesta variável. A próxima linha é uma referência indireta e o comando seria interpretado como

```
USE LOTE
```

Variáveis indexadas virtuais ("não reais") podem ser criadas utilizando-se esta referência indireta da seguinte maneira:

```
STORE 10 TO ctr
STORE 'N'+STR(ctr,2) TO var
STORE 3 TO &var
```

A primeira linha define CTR como uma variável numérica e lhe dá o valor de 10. A segunda linha cria a variável N10 por meio da função de "N" e do conjunto de caracteres que representam o número 10. O nome da variável é armazenado na variável VAR. O programa então armazena o número 3 na variável N10 que foi referenciada pelo VAR. Nós teremos mais exemplos posteriormente. No momento o importante é ver que N10 tornou-se componente de uma variável indexada virtual. Uma série de variáveis poderiam ser criadas (N00, N01, N02, N03 etc.) poderiam armazenar uma série de valores, e também ser nomes de campos do database.

Uma terceira maneira de se economizar espaço para variáveis é salvá-las em um arquivo de dados como campos de dados variáveis. Por exemplo, se você está desenvolvendo um sistema de estoque e deseja mostrar uma parte do estoque na tela. O arquivo mestre do sistema poderia ser usado para criar um arquivo para exibição, onde, cada registro desse arquivo, representaria uma linha na tela. Uma vez que você estaria mostrando campos do registro, não usaria espaço com variáveis de memória.

ALTERANDO UM PROGRAMA

Se você utiliza discos flexíveis, a melhor maneira de criar e testar programas é criar um disco do sistema contendo o seu processador de textos e o dBASE II na unidade A e manter todos os programas e arquivos na unidade B. Se você usa disco rígido, provavelmente vai querer manter tudo no disco rígido. Na criação e edição da maioria dos programas, você provavelmente desejará utilizar o seu processador de textos. O dBASE II, de qualquer maneira, tem um editor. Se necessário, você poderá editar (ou mesmo criar) um programa com ele. Isto é feito utilizando-se o comando MODIFY.

Como um exemplo, monte, com o seu processador de textos, o seguinte programa:

```
STORE 3 TO ctr
STORE 1 TO ptr
STORE ctr + ptr TO x
? x
```

Isto colocará o valor 3 na localização chamada CTR e o valor 1 na localização chamada PTR. Os conteúdos de CTR e PTR são, então, adicionados e o resultado é armazenado em uma localização chamada X. O comando "?" mostrará o conteúdo de X. Prepare o programa com seu processador de textos e salve-o como TESTE.CMD.

Agora ative o dBASE II:

```
DBASE
```

e entre com a data quando pedida. Agora execute o seu programa.

```
DO TESTE
```

Caso o programa esteja em uma unidade diferente da unidade do dBASE II, você precisará colocar o nome da unidade junto com o nome do programa como, por exemplo, DO B:TESTE).

Você veria o valor armazenado com cada instrução bem como o valor final. Agora, entre com o comando

```
SET TALK OFF
```

e comece a executar o seu programa novamente com

```
DO TESTE
```

Desta forma você não verá os valores intermediários e apenas o valor final será mostrado. Por isso, na maioria dos programas você vai querer começar com a instrução SET TALK OFF. TALK é muito útil para corrigir e tirar erros de programas e será comentado no Capítulo 17.

Agora, ainda sob o dBASE II, entre com o comando

```
MODIFY COMMAND TESTE
```

(ou B:TESTE se o programa está na unidade B).

Agora você pode alterar o programa no que for necessário. Você verá que chaves de controle podem ser usadas para movimentar o cursor e editar o programa. As chaves de controle estão descritas na Tabela 4-1.

Note que quando o cursor está na última linha da tela, CONTROL-X moverá para a próxima tela. De maneira semelhante, CONTROL-E e CONTROL-R moverão para a tela anterior se você estiver na primeira linha.

Altere o programa adicionando SET TALK OFF como primeira linha e alterando o valor salvo em PTR para 2. Agora salve o programa e reexecute-o.

O comando MODIFY COMMAND será útil para efetuar mudanças rápidas no programa, mas ele tem algumas limitações:

1. O tamanho da linha é de apenas 77 caracteres ou menos (embora você possa continuar usando ponto e vírgula para conectar-se a linha seguinte).
2. Qualquer caractere de tabulação é convertido em espaço.
3. O cursor só pode ser movimentado para trás por sobre 4000 caracteres.

Tabela 4-1. Controles utilizados no comando MODIFY COMMAND

CONTROL-S		Retorna um caractere
CONTROL-D		Avança um caractere
CONTROL-E **CONTROL-R**	ou	Retorna uma linha
CONTROL-X		Avança uma linha
CONTROL-N		Insere uma linha em branco
CONTROL-T		Remove a linha
CONTROL-Y		Limpa a linha
CONTROL-V		Ativa ou desativa inserção
CONTROL-G		Elimina o caractere onde está o cursor
DELETE		Elimina os caracteres a esquerda do cursor
CONTROL-C		Avança para a próxima tela
CONTROL-Q		Termina a operação sem alterar o arquivo
CONTROL-W		Salva o arquivo em disco e termina

4. Não existe movimentação por grandes quantidades de linhas nem formas para pesquisar argumentos como existe na maioria dos editores e processadores de texto.

EXPRESSÕES

Expressões matemáticas e de conjuntos podem usar números ou conjunto de caracteres. Uma expressão é uma série de operações e operandos (valores) que podem ser calculados pelo dBASE II. Por exemplo, no comando do dBASE II

```
STORE (2+3) TO x
```

"(2+3)" é uma expressão. O resultado da adição (5) é armazenado em X. As operações matemáticas estão descritas na Tabela 4-2.

É importante o uso de parênteses nas expressões sob o dBASE II, uma vez que existem regras bem definidas sobre a seqüência de cálculo de uma expressão.

EXIBINDO VARIÁVEIS E EXPRESSÕES

Por meio do comando "?" temos a forma mais rápida de exibir uma variável, um texto, valores numéricos e campos de dados na tela ou na impressora.

Tabela 4-2. Operações no dBASE II

	Matemática		Lógicas
+	Adição	.AND.	E
–	Subtração	.OR.	Ou
*	Multiplicação	.NOT.	Não
/	Divisão		
	Relacionais		**Conjuntos**
<	Menor que	+	Encadeamento
>	Maior que	–	Encadeamento com brancos à direita
=	Igual a		
<=	Menor que ou igual a	$	Subconjunto (existência de um conjunto em outro)
=>	Maior que ou igual a		
<>	Diferente de		
#	Diferente de		

Normalmente o valor é enviado para a tela, como podemos observar em exemplos como

```
STORE 2+3 TO x
```

O comando "??" poderá ser usado caso não se queira que, antes de ser feita a exibição, se vá para a próxima linha, como

```
? "Tudo"
?? "bem!"
```

Se o comando SET PRINT ON foi dado, a saída será dirigida para a impressora e para a console. Ao final do programa o comando SET PRINT OFF deverá ser usado para suspender a saída na impressora. O comando SET PRINT ON é como uma chave dentro do dBASE II para mandar ou não saídas para a impressora.

Se a saída deve ser apenas na impressora (não se quer saída na console), deverão ser usados os comandos SET CONSOLE OFF e SET PRINT ON. Esteja certo de voltar a dar saída na console por meio do comando SET CONSOLE ON após o término da impressão. Por exemplo,

```
SET CONSOLE OFF
SET PRINT ON
? 'TESTANDO'
SET PRINT OFF
SET CONSOLE ON
```


Lembre-se de que conjuntos de caracteres e números não podem ser encadeados. Por exemplo, o comando

```
STORE 1 TO x
? 'TESTE' + x
```

produzirá uma mensagem de erro. Use a função STR (descrita na próxima seção) para converter números em conjuntos caso você deseje juntá-los.

FUNÇÕES

Uma variedade de funções, ou operações com uma finalidade especial, estão disponíveis ao usuário:

— Mostra o número do registro atual como uma variável numérica.

***** — Usada para determinar se o registro foi retirado do arquivo. Por exemplo:

```
IF *
   SKIP
   LOOP
ENDIF
```

fará a rotina saltar o registro caso ele tenha sido marcado para ser eliminado.

! (" < caracteres > ") — Função para transformação em maiúsculas. Por exemplo:

```
? ! ('teste')
```

será exibido como

```
TESTE
```

$ (" < caracteres", início, tamanho >) — Função subconjunto, parecida com a função MID do BASIC. Pode ser usada para extrair parte de um conjunto.

```
STORE $ ('SAMBA',2,3) TO y
? y
```

mostrará

```
AMB
```

< "caracteres 1" > $ (< "caracteres 2" >) — Pesquisa de subconjunto. Retornará um valor de verdadeiro se caracteres1 for encontrado em caracteres2. Se caracteres1 não for encontrado, será retornado um valor de falso.

@ (<"caracteres 1">, <"caracteres 2">) Função em. Se caracteres1 for encontrado em caracteres2, será devolvido um valor numérico correspondente à posição inicial em que está caracteres1 em caracteres2. Caso o caracteres1 não seja encontrado em caracteres2 será devolvido um valor 0.

CHR(<número>) Função de conversão de numéricos para ASCII. Por exemplo:

```
? CHR (7)
```

fará com que soe o alarme do terminal.

DATE( ) Devolve a data (conforme a inicialização do dBASE II) no formato MM/DD/AA, formando um conjunto de oito caracteres.

EOF Devolverá um valor de verdadeiro se o usuário está no final do arquivo. Caso contrário um valor de falso será devolvido ao usuário.

FILE ("<nome do arquivo>") Devolverá um valor de verdadeiro, caso o arquivo exista; caso contrário será devolvido um valor de falso.

INT(<número>) Função inteiro. Devolve o valor inteiro de uma variável ou de uma expressão.

```
? INT (3/2)
```

LEN(<caracteres>) Função tamanho. Dá o tamanho de um conjunto de caracteres em uma variável numérica.

STR(número,tamanho,decimais) Função conjunto. Converte um número em um conjunto de caracteres. Tamanho é o comprimento do número incluindo o ponto decimal. Decimais é o número de casas decimais que existem.

VAL(caracteres) Dá o valor numérico dos números que estão em um conjunto de caracteres – ela é o inverso de função conjunto, por exemplo,

```
STORE '123.3' TO x
STORE VAL(x) TO Y
```

"y" terá o valor numérico de 123.3.

TRIM(caracteres) Função ajuste. Remove os brancos da direita de um conjunto de caracteres.

TYPE(nome da variável) Função tipo. Devolve um caractere dizendo qual é o tipo da variável

```
STORE 1 TO x
? TYPE(x)
N
```

TEST(expressão) Devolve um valor de verdadeiro se a expressão é válida; caso contrário devolverá um valor de falso.

RANK(<caracteres>) (Apenas na versão 2.4) Dá o valor numérico, em ASCII, do caractere mais a esquerda de um conjunto de caracteres. É similar à função ASC do BASIC

```
? RANK('BANCO')
66
```

Todas as funções que requerem caracteres como argumento devem tê-los entre aspas ou ser uma variável composta de caracteres.

OPERAÇÕES MATEMÁTICAS

É possível fazer certas operações matemáticas com campos de dados variáveis e armazenar os resultados em variáveis na memória ou em campos de dados. Os três comandos disponíveis são COUNT, SUM e TOTAL.

O comando COUNT é utilizado para contar o número de registros do database que satisfazem uma determinada condição. Suponha, por exemplo, que temos um arquivo de estoque definido da seguinte maneira:

```
STRUCTURE FOR FILE: MESTRE     .DBF
NUMBER OF RECORDS: 00010
DATE OF LAST UPDATE: 05/13/83
PRIMARY USE DATABASE
FLD     NAME       TYPE   WIDTH DEC
001     NOPECA     C      006
002     DESC       C      032
003     NOTAS      C      040
004     DISP       N      005
005     CUSTO      N      007    002
006     PRECO      N      008    002
** TOTAL **              00099
```

Neste arquivo existem 10 registros (como mostra a Figura 4-1). Para saber o número de registros que tem valor zero em DISP, digite o comando

```
USE MESTRE
COUNT FOR disp=0 TO x
? x
2
```

Nós também podemos definir uma extensão para o comando COUNT. Por exemplo, para verificar apenas os três primeiros registros do arquivo, digite

```
USE MESTRE
COUNT NEXT 3 FOR disp=0 TO x
? x
1
```

NOPEÇA	DESC	DISP	CUSTO	PREÇO
0001	CABO DA BATERIA	4	1.98	2.98
0002	RELÉ	0	8.50	11.33
0003	CORREIA DO VENTILADOR	2	6.95	9.27
0004	REGULADOR DE VOLTAGEM	2	11.50	15.29
0005	JUNTA UNIVERSAL	1	44.95	59.93
0006	CONJ. PINO DO FREIO	6	1.50	1.99
0007	GUARNIÇÃO DO FREIO	0	15.00	20.00
0008	MONTAGEM SILENCIOSO TRASEIRO	1	15.00	20.00
0009	LANTERNA LATERAL (E)	2	4.95	6.60
0010	LANTERNA LATERAL (D)	2	4.95	6.60

Figura 4-1. Registros do estoque da Oficina do Fred

Na versão 2.4 do dBASE II, uma condição de término poderá ser especificada para parar o comando por meio da opção WHILE. O formato completo do comando COUNT é

```
COUNT[<EXTENSÃO>][FOR<expressão>][TO<variável>][WHILE<expressão>]
```

O comando SUM pode ser usado para somar expressões numéricas ou valores. Por exemplo:

```
SUM disp TO x
? x
20
```

Imagine que você tenha um arquivo de estoque e deseja calcular o valor desse estoque, que normalmente é conhecido como custo total. O comando SUM poderia ser usado para calcular o valor de cada peça multiplicando o seu custo pelo número de peças disponíveis e totalizar as multiplicações em uma variável dando, desta forma, o valor total do estoque. Tudo isso seria feito por um comando como

```
SUM disp*custo TO x
? x
133.57
```

Um outro exemplo poderia ser um sistema de custo da mão-de-obra em que um grande número de registros de detalhes seriam criados para mostrar o tempo gasto por um empregado em um certo cliente durante o mês. Ao final do mês o total de horas gastas em cada serviço seria transferido para um arquivo de resumo. Esse total poderia ser calculado com o comando SUM e transferido para o arquivo de resumo por meio de:

```
SUM hrs TO thrs FOR serv='924'
```


O comando SUM também pode ser usado para verificar totais.

Mais de um arquivo poderá ser somado, porém, os campos devem ser numéricos. O formato completo do comando SUM é

```
SUM [<campo>][TO<lista de variáveis>][<extensão>][FOR<expressão>][WHILE<expressão>]
```

(a cláusula WHILE faz parte da versão 2.4)

O comando TOTAL é mais complexo sendo útil para consolidar muitos detalhes de transações em um único arquivo que sumariza os totais. Por exemplo, suponha que vários clientes usem várias peças durante o mês (veja a Figura 4.2). Ao final do mês este arquivo deverá ter todos os detalhes de cada transação. Existirão diversas requisições para uma mesma peça em vários lugares. Agora, imagine que desejamos o total de cada peça vendida sem nos preocuparmos com o cliente ou com a data da venda. O comando TOTAL seria utilizado para calcular esses totais e transferir essas informações para um arquivo sumário. A sintaxe do comando

```
TOTAL ON <CHAVE> TO <database> [FIELDS<nomes>][FOR<expressão>][WHILE<expressão>]
```

Se o database de destino já existe, a sua estrutura será utilizada para os dados que irá receber. Caso o database não exista, todos os campos serão copiados em sua forma original. Você desejará ter o database já definido, pois, desta forma, apenas os campos que lhe interessam serão copiados.

Apenas os campos descritos em FIELDS são totalizados em cada mudança da chave. O arquivo básico deverá estar classificado ou indexado pela chave que você quer usar.

```
TOTAL ON nopeca TO sumario
```

O database de resumo tem a seguinte estrutura:

```
STRUCTURE FOR FILE: SUMÁRIO .DBF
NUMBER OF RECORDS: 00010
DATE OF LAST UPDATE: 05/13/83
PRIMARY USE DATABASE
FLD     NAME      TYPE   WIDTH   DEC
001   NOPECA        C      006
002   DISP          N      005
** TOTAL **                00012
```

FATURA	NÚMERO DA PEÇA	QUANTIDADE
1039	001642	2
1039	001962	1
1041	003215	1
1042	001642	1
1043	001643	2
1044	001643	2

Figura 4-2. Dados do arquivo mensal de detalhes

NÚMERO DA PEÇA	QUANTIDADE
001642	3
001643	4
001962	1
003215	1

Figura 4-3. Arquivo sumário criado pelo comando TOTAL

A Figura 4-2 mostra um arquivo de detalhes típico da Oficina do Fred com as suas transações do mês. A Figura 4-3 mostra o arquivo sumário que resultaria de um comando TOTAL.

Os comandos COUNT, SUM e TOTAL são um tanto vagarosos, pois trabalham com o database inteiro. O programador deve lembrar-se disso e o usuário deve ser avisado do que está acontecendo e que o sistema não está "travado". Veja também que as opções FOR e WHILE são mutuamente exclusivas, ou seja, você pode usar apenas uma delas em um comando COUNT, SUM ou TOTAL.

CINCO

Desenhando o Sistema

Nós já vimos que desenhar o arquivo database é apenas uma das primeiras etapas no desenho de um sistema. Normalmente vários arquivos são usados no sistema total. Vários programas são utilizados para processar esses arquivos e gerar vários relatórios. O fluxo do desenho pode mais facilmente ser visto da seguinte maneira:

I. Fase de Análise
   1. Definição dos objetivos do sistema.
   2. Definição das informações de saída necessárias para atingir a esses objetivos.
   3. Definição dos relatórios, formulários e telas para atingir esses objetivos. Isso é definido como especificações de saída.
   4. Definição das informações de entrada necessárias para obter as saídas. Isso é definido como especificações de entrada.

II. Fase de Desenho
   1. As funções do sistema são separadas em entidades funcionais claramente definidas de maneira como o usuário as vê.
   2. Os arquivos são projetados para atender as funções específicas.
   3. São definidos os índices para cada arquivo.
   4. Um conjunto de programas é definido para atender cada função.

III. Programação
   1. O programa do menu principal é escrito definindo cada módulo funcional.
   2. Outros menus são criados para cada uma das funções.
   3. Os programas são escritos em uma forma modular, implementando-se um módulo (ou função) por vez.

Neste capítulo, nós iremos desenhar um sistema de cálculo de custos para um advogado.

FASE DE ANÁLISE

Um advogado tem vários clientes (com um ou mais serviços que devem ser contabilizados) e necessita acompanhar o tempo e os gastos efetuados com cada cliente em cada serviço, de forma que possam ser cobrados. Existem vários tipos de serviço a serem cobrados (consultas, acompanhamentos, ações etc.) e um tempo necessário para o acompanhamento de cada um desses serviços.

Ao final do mês, o advogado necessitará de um relatório que mostre o tempo total gasto com cada cliente, em cada um de seus serviços ou consultas, de forma que possa cobrá-los. Outros relatórios auxiliares também serão necessários para mostrar detalhadamente os tempos, por assistente, serviço e cliente.

Para obter esses relatórios, o database deverá ter registros informando sobre o cliente, serviço, tipo de serviço, tempo e profissionais envolvidos. O sistema também deverá ter informações sobre a porcentagem de cada profissional envolvido para calcular o custo verdadeiro de cada transação.

DESENHO ESTRUTURADO

O próximo passo é separar o sistema em funções distintas. No nosso caso teríamos quatro funções:

- Organização. Clientes, serviços, profissionais e tipos de serviços são definidos para o sistema.
- Operações diárias. Os dados são entrados, editados e relatados diariamente.
- Geração de relatórios. Criação dos relatórios mensais para cobrança.
- Outras. Arquivos, funções anuais e outras são indexadas.

Cada uma dessas funções será atendida por um ou mais de um programa.

O DIAGRAMA DO FLUXO DOS DADOS

O próximo passo na criação do sistema é desenhar o "diagrama do fluxo de dados". Este diagrama mostra o fluxo de informação do sistema proposto e serve como uma ferramenta de comunicação entre o programador e o usuário.

O diagrama do fluxo dos dados mostra apenas os blocos funcionais, os arquivos e o fluxo de informação. Ele é muito diferente do diagrama do fluxo do programa; é familiar a muitos programadores e os dois não devem ser confundidos. O diagrama do fluxo dos dados mostra o fluxo da informação, enquanto o diagrama do fluxo do programa mostra a forma com que o programa encontra uma solução.

A Figura 5-1 é o diagrama do fluxo dos dados do nosso sistema de custos. Foram definidos quatro blocos funcionais:

1. Operações de organização
2. Operações por lote (as operações diárias)
3. Operações de lançamento
4. Geração de relatórios

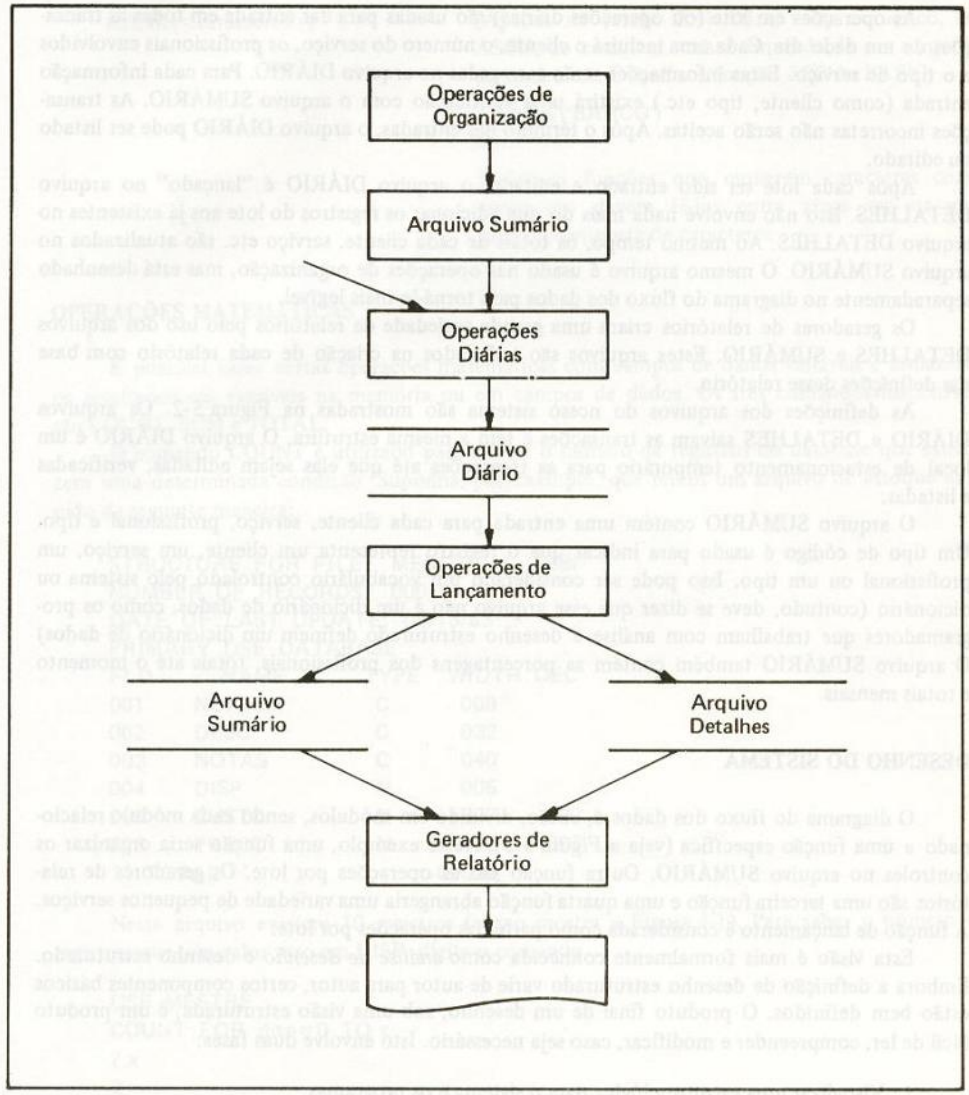


Figura 5-1. Diagrama do fluxo dos dados do sistema de custos

As operações de organização permitem ao operador cadastrar clientes, serviços, profissionais (e suas porcentagens) e tipos de serviço. Também é possível editar, eliminar ou listar clientes, serviços, profissionais e tipos de serviço. Estes dados estão carregados no arquivo SUMÁRIO.

As operações em lote (ou operações diárias) são usadas para dar entrada em todas as transações de um dado dia. Cada uma incluirá o cliente, o número do serviço, os profissionais envolvidos e o tipo de serviço. Estas informações serão carregadas no arquivo DIÁRIO. Para cada informação entrada (como cliente, tipo etc.) existirá uma verificação com o arquivo SUMÁRIO. As transações incorretas não serão aceitas. Após o término das entradas, o arquivo DIÁRIO pode ser listado ou editado.

Após cada lote ter sido entrado e editado, o arquivo DIÁRIO é "lançado" no arquivo DETALHES. Isto não envolve nada mais do que adicionar os registros do lote aos já existentes no arquivo DETALHES. Ao mesmo tempo, os totais de cada cliente, serviço etc. são atualizados no arquivo SUMÁRIO. O mesmo arquivo é usado nas operações de organização, mas está desenhado separadamente no diagrama do fluxo dos dados para torná-lo mais legível.

Os geradores de relatórios criam uma grande variedade de relatórios pelo uso dos arquivos DETALHES e SUMÁRIO. Estes arquivos são indexados na criação de cada relatório com base nas definições desse relatório.

As definições dos arquivos do nosso sistema são mostradas na Figura 5-2. Os arquivos DIÁRIO e DETALHES salvam as transações e têm a mesma estrutura. O arquivo DIÁRIO é um local de estacionamento temporário para as transações até que elas sejam editadas, verificadas e listadas.

O arquivo SUMÁRIO contém uma entrada para cada cliente, serviço, profissional e tipo. Um tipo de código é usado para indicar que o registro representa um cliente, um serviço, um profissional ou um tipo. Isso pode ser considerado um vocabulário controlado pelo sistema ou dicionário (contudo, deve se dizer que esse arquivo não é um dicionário de dados, como os programadores que trabalham com análise e desenho estruturado definem um dicionário de dados) O arquivo SUMÁRIO também contém as porcentagens dos profissionais, totais até o momento e totais mensais.

DESENHO DO SISTEMA

O diagrama do fluxo dos dados é, então, dividido em módulos, sendo cada módulo relacionado a uma função específica (veja a Figura 5-3). Neste exemplo, uma função seria organizar os controles no arquivo SUMÁRIO. Outra função são as operações por lote. Os geradores de relatórios são uma terceira função e uma quarta função abrangeria uma variedade de pequenos serviços. A função de lançamento é considerada como parte das operações por lote.

Esta visão é mais formalmente conhecida como *análise de desenho* e desenho estruturado. Embora a definição de desenho estruturado varie de autor para autor, certos componentes básicos estão bem definidos. O produto final de um desenho, sob uma visão estruturada, é um produto fácil de ler, compreender e modificar, caso seja necessário. Isto envolve duas fases:

1. Visualizar uma estrutura lógica para o sistema e os programas.
2. Utilizar na concepção uma visão "de cima para baixo", definindo primeiramente as grandes funções e gradualmente trabalhar em direção aos componentes menores.

Outras características de uma programação estruturada são:

1. Desenho "de cima para baixo". Começando por definir as metas, as entradas e as saídas.
2. Um desenho modular deve ser usado. Desenhado a partir do formato "de cima para baixo".

STRUCTURE FOR FILE: B:SUMÁRIO1.DBF
NUMBER OF RECORDS: 00058
DATE OF LAST UPDATE: 00/00/00
PRIMARY USE DATABASE

FLD	NME	TYPE	WIDTH	DEC
001	TIPO	C	001	
002	NOME	C	006	
003	TOTAL	N	010	002
004	TAXAS	N	010	002
005	THRS	N	008	002
006	DESC	C	032	
007	VENDAS	C	001	
008	TOTAL2	N	008	002
TOTAL			00077	

STRUCTURE FOR FILE: B:DETALHES.DBF
NUMBER OF RECORDS: 00040
DATE OF LAST UPDATE: 00/00/00
PRIMARY USE DATABASE

FLD	NAME	TYPE	WIDTH	DEC
001	CLIENTE	C	006	
002	SERV	C	006	
003	DATA	C	006	
004	DESC	C	020	
005	PROF	C	003	
006	TIP	C	003	
007	HRS	N	008	002
008	CUSTO	N	010	002
009	VENDAS	C	001	
TOTAL			00064	

STRUCTURE FOR FILE: B:DIÁRIO.DBF
NUMBER OF RECORDS: 00001
DATE OF LAST UPDATE: 00/00/00
PRIMARY USE DATABASE

FLD	NAME	TYPE	WIDTH	DEC
001	CLIENTE	C	006	
002	SERV	C	006	
003	DATA	C	006	
004	DESC	C	020	
005	PROF	C	003	
006	TIP	C	003	
007	HRS	N	008	002
008	CUSTO	N	010	002
009	VENDAS	C	001	
010	LANC	L	001	
TOTAL			00065	0

Figura 5-2. Estrutura dos arquivos do sistema de custos

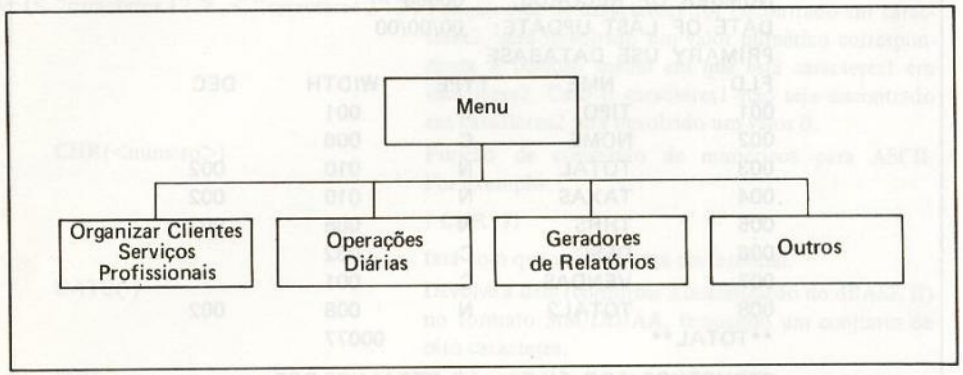


Figura 5-3. Diagrama dos blocos funcionais

3. Cada módulo tem uma entrada e uma saída e é uma entidade independente.
4. Cada módulo deve ser documentado com o nome, funções, entradas e necessidades de saída, relacionamento com outros módulos, última atualização e outras observações necessárias.
5. Cada módulo deve ser limitado até 50 linhas de instruções. Use quantos comentários quiser.
6. Os controles lógicos permitidos incluem IF/THEN/ELSE, DO/WHILE, DO/UNTIL e CASE. Os comandos GOTO devem ser raramente utilizados (se possível).
7. Cortes devem ser usados após todos os controles lógicos.

O dBASE II obriga o usuário a pensar em termos de desenho estruturado e a utilizar uma programação estruturada. Não existe o comando "GOTO" como no BASIC (existe um GOTO, mas é utilizado com uma outra finalidade).

Não existem nomes para as instruções (como no BASIC) e você não pode saltar de um ponto ao outro dentro do programa. O BASIC não é uma linguagem estruturada. Com o dBASE II todos os programas são modulares e muitas vezes pequenos módulos são usados em muitos programas. Os programas são legíveis e facilmente modificados.

SEIS

Desenhando Menus

Uma vez que os blocos funcionais do sistema estejam definidos, os programas de menu podem ser elaborados de maneira a atender os blocos funcionais. Muitos níveis de menus podem estar envolvidos. O usuário não necessita saber o nome dos programas ou dos arquivos utilizados pelo sistema. Uma tela de menu mostra os nomes funcionais e o usuário seleciona a função desejada.

Um menu principal é criado para chamar uma área desejada como organização, processamento diário e outras. A função, por sua vez, é implementada utilizando um submenu. Este submenu pode chamar qualquer um dos muitos programas existentes. O menu, os submenus e os programas têm um relacionamento hierárquico. O diagrama para este relacionamento mostra as telas. As telas para o sistema de custos que desenhamos no Capítulo 5 são mostrados na Figura 6-1. As quatro funções anteriormente discutidas estão claramente definidas. Todas as operações de organização são feitas a partir de ORGMENU. Os lançamentos são considerados como parte das operações diárias e por isso aparecem no diagrama como parte de LOTMENU (operações por lotes). Os relatórios serão impressos a partir de RELMENU e outras operações serão iniciadas pelo OUTMENU.

O menu principal controla a execução de todos os programas. É o *único* programa que o usuário conhece.

Um exemplo de listagem do programa de menu para o sistema de custos é mostrada a seguir:

```
* menu principal do sistema de custos
* por carl townsend
* menu
* 12/27/82
* inicializa o sistema
SET TALK OFF
SELECT PRIMARY
SET FORMAT TO SCREEN
SET PRINT OFF
SET CONSOLE ON
*· pesquisa à opção efetuada
```

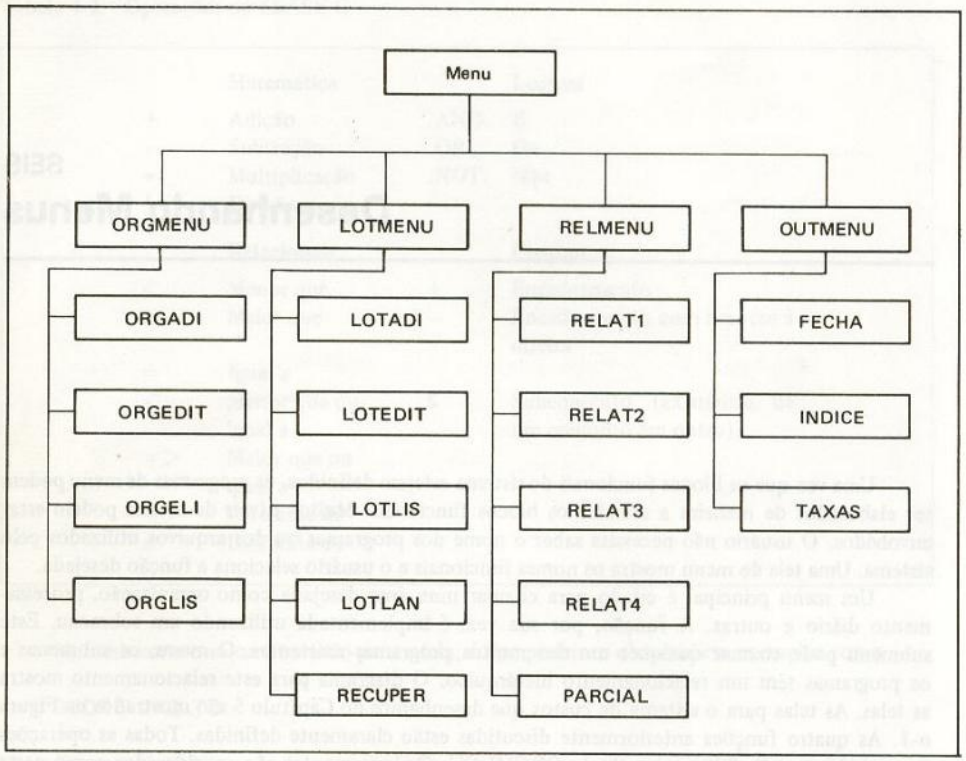


Figura 6-1. Telas para o sistema de custos

```
DO WHILE t
        CLEAR
        STORE 'SISTEMA PARA TESTE          ' TO mtit
        SAVE TO título
        ERASE
        @ 1,1 SAY  '**********'+MTIT+'**********'
        @ 2,1 SAY  'MENU PRINCIPAL'
        @ 2,60 SAY DATE ( )
        @ 4,1 SAY  'OPÇÕES:'
        @ 5,1 SAY  '    0    VOLTA AO SISTEMA OPERACIONAL'
        @ 6,1 SAY  '    1    CLIENTES,SERVIÇOS,PROFISSIONAIS,TIPOS'
        @ 7,1 SAY  '    2    OPERACOES POR LOTE          '
        @ 8,1 SAY  '    3    GERADORES DE RELATORIOS     '
        @ 9,1 SAY  '    4    OUTRAS      '
        @ 10,1 SAY '    5    VOLTA AO DBASE'
        @ 13,1 SAY 'SELECIONE A OPCAO:  '
```



```
        SET CONSOLE OFF
        WAIT TO opcao
        SET CONSOLE ON
        * verifica se a opcao e valida e desvia para o submenu apropriado
        DO CASE
                CASE opcao='0'
                        ERASE
                        QUIT
                CASE opcao='1'
                        DO orgmenu
                CASE opcao='2'
                        DO lotmenu
                CASE opcao='3'
                        DO relmenu
                CASE opcao='4'
                        DO outmenu
                CASE opcao='5'
                        SET TALK OFF
                        ERASE
                        CANCEL

        OTHERWISE
                @ 23,1 SAY 'OPCAO INVALIDA
                STORE 1 TO xx
                DO WHILE xx<35
                        STORE xx+1 TO xx
                ENDDO
ENDCASE
ENDDO WHILE t
```

A tela, da forma como será utilizada, está na Figura 6-2.

As linhas que começam com um asterisco ou com o comando NOTE são tratadas como

```
********* SISTEMA DE TESTE *********
MENU PRINCIPAL                              02/23/83
OPCOES:
   0    VOLTA AO SISTEMA OPERACIONAL
   1    CLIENTES, SERVIÇOS, PROFISSIONAIS, TIPOS
   2    OPERACOES POR LOTE
   3    GERADORES DE RELATORIOS
   4    OUTRAS
   5    VOLTA AO DBASE
   SELECIONE A OPCAO:
WAITING
```

Figura 6-2. Tela do menu principal

comentários. São utilizadas para indicar o objetivo do programa, o nome do autor, nome dos arquivos utilizados e última atualização, além de outras observações técnicas necessárias.

A melhor maneira de se iniciar o menu principal é com os comandos

```
SET TALK OFF
SELECT PRIMARY
SET FORMAT TO SCREEN
SET PRINTER OFF
SET CONSOLE ON
```

Isto fará com que as opções anteriormente ativas sejam desligadas e preparado o dBASE II para o menu.

Um *loop* foi definido na execução do menu por um DO WHILE com saídas apenas para o sistema operacional (opção=0) e para o dBASE II (opção=5) pelos seguintes comandos

```
DO WHILE t
```

e

```
ENDDO
```

Um *loop* DO WHILE é uma série de instruções que continuarão sendo executadas, enquanto a condição que aparece no comando DO WHILE é verdadeira.

Como exemplo nós temos

```
DO WHILE t
```

o que significa DO WHILE VERDADEIRO ou continue para sempre. Duas saídas, contudo, existem no loop. Se o usuário entrar com a opção 5, o sistema de custos terminará e aparecerá o sinal de pronto do dBASE II. Se a opção 0 for utilizada, o sistema sairá do dBASE II para o sistema operacional.

Um outro loop DO WHILE existe mais abaixo no programa:

```
store 1 to xx
DO WHILE xx<35
        store xx+1 to xx
ENDDO
```

Este é um loop para retardamento. O valor "XX" é inicialmente colocado como 1. O loop é executado enquanto "xx" é menor do que 35 e incrementado de um em cada execução. Assim, o dBASE II incrementa o contador 35 vezes e então termina o loop.

A variável "mtit" é definida e salva no arquivo TÍTULO.MEM. A tela é limpa e o menu criado na tela. Nós temos que o usuário pode criar uma tela especificando a linha e a coluna em que um texto vai ser apresentado pelo comando @ com SAY:

```
@    2,1 SAY 'MENU PRINCIPAL'
```

Após apresentarmos toda a nossa tela, nós gostaríamos de obter a opção entrada pelo usuário. O comando


```
WAIT TO opcao
```

fará o programa esperar até que um caractere seja entrado pelo operador (da mesma maneira que INKEY no BASIC).

No BASIC, os tipos das variáveis podem ser definidos pelos sufixos ao nome da variável, como A$ para um conjunto de caracteres e B% para uma variável inteira.

No dBASE II, porém, não existem sufixos nos nomes das variáveis. Assim, ele não sabe quanto espaço reservar para uma variável ou tipo de variável. Por exemplo, no comando WAIT, "opção" é a variável, mas o dBASE II não sabe se está aguardando um conjunto de caracteres, um número ou uma variável lógica. Em algumas linguagens, como Pascal, o tipo da variável é definido por uma declaração no início do programa. Com o dBASE II a única maneira de se declarar uma variável é pelo armazenamento de algo nela. Por isso, nós dissemos ao dBASE II que a variável "opção" é um conjunto de caracteres pelo armazenamento de um caractere em branco antes do comando WAIT. Se você esquecer de fazer isto em seu programa obterá um erro.

A próxima parte do programa, que começa com um DO CASE, não é um *loop*. Ela verifica as várias opções para saber qual o usuário solicitou e também se a opção é válida ou não. Se uma opção não válida foi entrada, a mensagem de "OPÇÃO INVÁLIDA" será apresentada e, fora do loop do WHILE, é permitido ao usuário uma nova chance de tentar outra opção.

A escolha da opção fechará todos os arquivos pelo comando CLEAR e sairá do dBASE II para o sistema operacional. A opção 5 sairá do loop com o comando CANCEL e voltará ao dBASE II. As outras opções chamarão os submenus permitindo ao usuário outras escolhas.

SUBMENUS

O submenu é semelhante ao menu principal, exceto por chamar apenas o programa específico de função.

```
* submenu de organizacao para o sistema de custos
* por carl townsend
* orgmenu
* 12/27/82
SET TALK OFF
SELECT PRIMARY
* pesquisa para selecionar o menu
DO WHILE t
        CLEAR
        RESTORE FROM titulo
        ERASE
        @ 1,1 SAY  '**********' +mtit+ '**********'
        @ 2,1 SAY  'MENU DE ORGANIZACAO '
        @ 2,60 SAY DATE( )
        @ 4,1 SAY  'OPCOES: '
        @ 5,1 SAY  '    0          TERMINA'
        @ 6,1 SAY  '    1          INCLUI CLIENTES,SERVICOS ETC.'
        @ 7,1 SAY  '    2          EDITA CLIENTES,SERVICOS ETC.'
```

```
@ 8,1 SAY  '    3          ELIMINA CLIENTES,SERVICOS ETC.'
@ 9,1 SAY  '    4          LISTA CLIENTES,SERVICOS ETC.'
@ 17,1 SAY 'OPCAO:'
STORE ' ' TO escolha
WAIT TO escolha
* verifica a entrada de uma opcao valida e desvia para o programa apropriado
DO CASE
        CASE escolha='0'
                SET TALK ON
                ERASE
                RETURN
        CASE escolha='1'
                DO orgadi
        CASE escolha='2'
                DO orgedit
        CASE escolha='3'
                DO orgeli
        CASE escolha='4'
                DO orglis
        OTHERWISE
                @ 23,1 SAY 'OPCAO INVALIDA  '
                STORE 1 TO xx
                DO WHILE xx<35
                        STORE xx+1 TO xx
                ENDDO WHILE xx<35
ENDCASE
ENDDO WHILE t
```

Note que o comando CLEAR é executado dentro de qualquer programa e as variáveis são restauradas a partir do arquivo TÍTULO.MEM. Pela opção 0, a execução do comando RETURN trará o menu principal novamente para o usuário. A tela deste submenu é mostrada na Figura 6-3.

```
********* SISTEMA PARA TESTE *********
MENU DE ORGANIZACAO                    02/23/83
OPCOES:
  0    TERMINA
  1    INCLUI CLIENTES, SERVICOS ETC.
  2    EDITA CLIENTES, SERVICOS ETC.
  3    ELIMINA CLIENTES, SERVICOS ETC.
  4    LISTA CLIENTES, SERVICOS ETC.
  OPCAO:
WAITING
```

Figura 6-3. Tela do submenu

ARQUIVO DE PARÂMETROS

Em alguns sistemas você quererá que o sistema seja controlado por certos parâmetros que possam ser alterados pelo usuário. Isto pode ser feito pela criação de um *arquivo de parâmetros* que contenha esses valores. Eles podem ser os títulos dos relatórios ou o tamanho do papel para a impressora.

O arquivo de parâmetros contém um único registro e cada campo do registro o valor de uma variável. O registro pode ser alterado pelo usuário por intermédio do comando EDIT1. Estes parâmetros são, normalmente, lidos na execução do menu principal e armazenados em variáveis. As variáveis, então, são armazenadas em arquivos tipo .MEM.

```
SET TALK OFF
SELECT PRIMARY
SET FORMAT TO SCREEN
SET PRINT OFF
SET CONSOLE ON
CLEAR
ERASE
@ 15,1 SAY 'CARREGANDO O ARQUIVO DE PARAMETROS . . . '
* pega os valores do arquivo de parametros
STORE DATA ( ) TO xdata
USE param
STORE tit TO mtit
STORE itit TO mitit
STORE itam TO mitam
STORE icont TO micont
STORE isupport TO misuprt
STORE ipercent TO miprcnt
STORE itotal TO mitotal
STORE ibruto TO mibruto
STORE iliq TO miliq
STORE btit TO mbtit
STORE btam TO mbtam
STORE bcont TO mbcont
STORE bsupport TO mbsuprt
STORE bpart TO mbpart
STORE dmarca TO mdmarca
STORE dacao TO mdacao
STORE conta TO mconta
STORE period TO mperiod
SAVE TO paramet
* verifica selecao no menu principal
```

CONTROLANDO A DATA DE INICIALIZAÇÃO

Você normalmente começa a trabalhar com o dBASE II pelo comando

```
DBASE
```

Este comando fará com que o dBASE II solicite a data e depois dê o sinal de pronto. Todavia, você pode chamar o arquivo de comandos imediatamente:

```
DBASE MENU
```

Isto fará com que o programa MENU comece a ser executado imediatamente, saltando a solicitação da data. Para incluir a data, faça com que o programa requisite a data da seguinte maneira:

```
SET TALK OFF
STORE '          ' TO mdata
ERASE
@ 15,1 SAY 'ENTRE COM A DATA (MM/DD/AA) ' GET mdata PICTURE '99/99/99'
READ
SET DATE TO &mdata
SET TALK ON
RETURN
```

Isto determinará a data de inicialização que será apresentada em todos os relatórios que usam DATE( ) como variável fictícia.

O menu também pode ser usado para estabelecer teclas com funções para fazerem coisas especiais em certos programas. Você verá exemplos disso no Capítulo 11.

Em alguns sistemas você pode querer usar o menu mestre do dBASE II para chamar outros programas fora do dBASE II, como o processador de textos, por exemplo. Isto pode ser feito em sistemas com um único usuário, pelo comando QUIT:

```
QUIT TO "SELECT", "DBASE MENU"
```

Isto terminará o programa SELECT. Na saída do programa SELECT você retornará para o dBASE II e executará o arquivo de comandos MENU. Note que a opção "QUIT TO" não funciona sob MP/M.

SETE

Adicionando Registros ao Database

A criação do database, como foi feita no Capítulo 3, apenas criou a estrutura para armazenamento da informação. Nenhum dado foi armazenado no database. Para utilizar o database, devemos desenvolver programas para adicionar dados ao database, para editá-los e finalmente para selecionar e listar as informações que desejamos numa maneira simples de usar. O dBASE II providencia várias maneiras de fazer cada uma dessas funções. Neste capítulo nós veremos a primeira delas: adicionar registros ao database.

O COMANDO APPEND

A maneira mais simples de adicionar registros em um database é pelo uso do comando APPEND. No sistema de estoque da Oficina do Fred seria

```
USE mestre
APPEND
```

Isto fará com que a tela seja limpa e um simples registro em branco será mostrado na tela, com todos os campos vazios (veja a Figura 7-1). O usuário simplesmente preenche a tela. Após o cursor ter se movido no último campo, o registro é armazenado e um outro registro vazio é mostrado na tela. A edição e o controle são feitos pela utilização de chaves de controle, conforme descrito na Tabela 7-1.

O modo de inserção (CONTROL-V) é útil para colocarmos caracteres entre os já existentes. Caso este modo não esteja ativo, o que é o normal, os caracteres digitados simplesmente serão sobrepostos sobre os caracteres existentes.

O comando APPEND também pode ser usado com databases indexados. Na Oficina do Fred com o arquivo MESTRE do estoque, poderíamos usar

```
USE mestre INDEX mestre
APPEND
```

Dessa forma, cada registro é adicionado ao database e o índice é automaticamente atualizado. Este método é muito bom para pequenos databases que precisam ser construídos e necessitam de muito pouca edição, mas ele tem algumas desvantagens. O usuário não tem controle sobre o processo de edição dos registros:

```
RECORD # 000010
NOPECA      :      :
DESC        :             :
NOTAS       :                    :
DISP        :   :
CUSTO       :      :
PREÇO       :   :  :
```

Figura 7-1. Tela do comando APPEND

O usuário pode readicionar registros mesmo se o registro já existir no arquivo de dados. Em um sistema de estoques, isto significa que você pode ter um número da peça duplicado no estoque. O usuário também não tem controle sobre a tela utilizada para a entrada dos dados. Você poderá usar este tipo de enfoque para arquivos de controle que não sejam mantidos pelo usuário (como o dicionário de dados, visto mais tarde neste Capítulo).

Tabela 7-1. Comandos de Controle com o Comando APPEND

CONTROL-E ou CONTROL-A	Volte um campo
CONTROL-X ou CONTROL-F	Avance para o próximo campo
CONTROL-S	Volte um caractere
CONTROL-D	Avance um caractere
CONTROL-Y	Limpe o campo com brancos
CONTROL-G	Elimine o caractere sob o cursor
DELETE	Elimine os caracteres à esquerda do cursor
CONTROL-Q	Termine a operação APPEND sem adicionar o registro
CONTROL-C ou CONTROL-W	Termine a operação APPEND e adicione o registro ou (se pelo menos um campo foi entrado) vá para a próxima tela de entrada de registro

ADICIONANDO REGISTROS SOB O CONTROLE DE UM PROGRAMA

Para a maioria das aplicações, deverá ser criado um programa para controlar a adição dos registros no database. A seguir, o exemplo de um programa para adicionar registros no database do estoque da Oficina do Fred.

```
* organiza o arquivo de estoque da Oficina do Fred
* por carl townsend
* 12/27/82
SELECT PRIMARY
RESTORE FROM TITULO
USE mestre INDEX mestre e
STORE t TO marca
* entrada das pecas
DO WHILE t
        * entra o numero da peca
        store '          ' to mno
        ERASE
        @ 1,1 SAY '**********'  +mtit+  '**********'
        @ 2,1 SAY   ' ADICIONANDO AO ESTOQUE'
        @ 2,60 SAY DATE ( )
        @ 15,1 SAY  'ENTRE O NUMERO DA PECA ' GET mno PICTURE '999999'
        @ 23,1 SAY  'ENTRE EM BRANCO PARA TERMINAR'
        READ
        IF $(mno,1,5)='      '
                ERASE
                RETURN
        ENDIF $(mno,1,5)='      '
        FIND &mno
        IF #<>0
                @ 23,1 SAY 'NUMERO JA EXISTENTE NO ARQUIVO'
                @ 23,60 SAY CHR(7)
                STORE 1 TO xx
                DO WHILE xx<35
                        STORE xx+1 TO xx
                ENDDO WHILE xx < 35
                LOOP
        ENDIF #<>0
        * numero de peca valido, adicione o valor
        ERASE
        @ 1,1 SAY '**********' +mtit+ ' **********'
        @ 2,1 SAY   'ADICIONANDO AO ESTOQUE '
        @ 2,60 SAY DATE( )
        STORE '                                                    ' TO mdesc
        STORE '
        TO mnotas
        STORE 0 TO mdisp
        STORE 0.0 TO mpeso
```

```
STORE 0.00 TO mcusto,mpreco,mpv
@ 5,1 SAY  ' NUMERO DA PECA '+mno
@ 6,1 SAY  'DESCRICAO ' GET mdesc
@ 7,1 SAY  'DISPONIVEL ' GET mdisp
@ 8,1 SAY  'CUSTO NO ATACADO ' GET mcusto
@ 9,1 SAY  'PRECO ' GET mpv
@ 10,1 SAY 'NOTAS ' GET mnotas
@ 23,1 SAY 'ENTRE UMA DESCRICAO EM BRANCO PARA TERMINAR'
READ
IF $(mdesc,1,5)='     '
        ERASE
        RETURN
ENDIF
* adiciona o registro
APPEND BLANK
REPLACE nopeca WITH mno,desc WITH mdesc,notas;
WITH mnotas
REPLACE disp WITH mdisp,custo WITH mcusto,preco;
WITH mpreco
ENDDO WHILE t
```

O programa começa abrindo o arquivo do estoque, entrando, então, em um loop DO WHILE que controla as adições repetidamente. A tela é apresentada na Figura 7-2.

Lembre-se de que você deve declarar a variável pelo armazenamento de algum valor nesta variável antes de usá-la, por exemplo:

```
STORE '     ' TO mno
```

Isto declara que "mno" é variável formada por um conjunto de caracteres. Isto é similar ao exemplo do menu, como foi visto no capítulo anterior.

A cláusula "picture" no comando GET não é necessária, mas, neste caso, assegura que apenas valores numéricos serão entrados para o número da peça. Se você entrar um caractere alfabético, o teclado travará, produzindo um som característico e o caractere não será entrado.

```
********** SISTEMA DE ESTOQUE **********

ADICIONANDO AO ESTOQUE                04/30/83

ENTRE O NUMERO DA PECA:
ENTRE EM BRANCO PARA TERMINAR
```

Figura 7-2. Tela para obter o número da chave

Você também pode ter uma máscara parcial para valores numéricos, como por exemplo:

```
@ 15,1 SAY "ENTRE O NUMERO DA PECA" GET MNO PICTURE '9X999'
```

Isto permite que o segundo caractere seja alfanumérico enquanto os demais sejam apenas caracteres numéricos. Os códigos para a cláusula PICTURE estão na Tabela 7-2. O comando

```
@ 20,1 SAY 'ENTRE O NUMERO DO TELEFONE' GET PNO PICTURE '999-9999'
```

seria o exemplo de entrada de um número telefônico.

O comando READ é necessário para que o programa pare e obtenha o valor correspondente ao número da peça. Se você esquecê-lo no programa ele não parará.

As três linhas após o comando READ regulam a condição de saída. Se o usuário entrar com um número de peça branco (ou apenas teclar RETURN), ele voltará ao menu anterior.

O comando FIND é então usado para determinar se a peça já existe no estoque. Se a peça é encontrada, uma mensagem é apresentada sobre a duplicação e um loop de retardamento é executado para manter a mensagem na tela por alguns segundos. O restante do loop DO WHILE é saltado com o comando LOOP e um outro número de peça é solicitado.

Se a peça não é encontrada, a tela é limpa e uma tela semelhante a do comando APPEND é apresentada (como na Figura 7-3). Observe, contudo, algumas diferenças:

1. A tela tem título e data.
2. Os valores entrados são carregados temporariamente em variáveis na memória ao invés de diretamente no database. Se o processo de adição for abortado acidentalmente pelo usuário, o database estará intacto.
3. O valor chave (o número da peça) não pode mais ser alterado e é apenas apresentado.

Como na primeira tela, todas as variáveis são declaradas primeiro pelos valores vazios armazenados nelas.

Neste ponto qualquer dos valores pode ser verificado antes do database ser atualizado. Isto é feito normalmente colocando-se esta segunda tela dentro de um loop que terminará somente após o usuário preencher os campos corretamente.

Tabela 7-2. Formato dos caracteres para o comando GET

Símbolo	Significado
#	Apenas dígitos, operações (+, –, etc.) e espaços podem ser entrados
9	(Igual a #)
X	Qualquer caractere pode ser entrado
A	Apenas caracteres alfabéticos podem ser entrados
!	Converte para maiúsculos se forem minúsculas
	Outros: apresenta o caractere na tela

```
********** SISTEMA DE ESTOQUE **********
ADICIONANDO AO ESTOQUE                    04/30/83

NUMERO DA PECA 000020
DESCRICAO:                    :
DISPONIVEL:                   :
CUSTO NO ATACADO:             :
PRECO:                        :
NOTAS:                        :

ENTRE UMA DESCRICAO EM BRANCO PARA TERMINAR
```

Figura 7-3. Tela para adicionar peças ao estoque sob o controle de um programa

Uma vez que você decidiu que os dados estão corretos, deve carregá-los das variáveis na memória para o database. O nosso programa primeiro emite um comando APPEND BLANK. Isto adiciona um novo registro ao database com os valores de cada campo com zeros ou brancos.

O comando REPLACE é utilizado, então, para carregar os dados das variáveis na memória para o database, como se segue:

```
REPLACE nopeca WITH mnopeca
```

O programa limpará a tela e requisitará um outro número de peça repetindo o ciclo.

Deve ser mencionado, também, que REPLACE pode ser utilizado para o database em sua totalidade pelos parâmetros condicionais. Por exemplo, se o Fred organizasse um outro arquivo no sistema com alguns totais mensais de algumas variáveis, como MTOTAL, ele poderia zerar estes campos para todos os registros do arquivo como

```
REPLACE ALL mtotal WITH 0
```

ou, ainda, fazê-lo condicionalmente se o número da peça for maior do que 1000 com o comando

```
REPLACE ALL mtotal WITH 0 FOR VAL (nopeca)>1000
```

Faça os seus programas o mais próximo possível da compreensão do usuário. Apresente mensagens sobre o que ocorrerá no próximo passo e como terminar a sessão. Uma boa prática é utilizar a última linha (linha 23) como linha de mensagens. A linha 0 não é aconselhável para o programador.

USANDO O DICIONÁRIO DE DADOS

Com alguns programas de adição de registros você pode querer verificar os valores de entrada contra um *dicionário de dados* assegurando a validação dos dados antes de serem colocados no arquivo. A seguir temos um exemplo desse tipo de programa para o sistema de custos. Deve ser mencionado que o conceito de "dicionário de dados" utilizado neste programa não é o mesmo usado normalmente no desenho de databases. No desenho de databases, o dicionário é um catálogo de todos os campos e arquivos utilizados no sistema, como também qualquer outra entidade de informação utilizada como referência no desenho do sistema. Neste capítulo, o conceito de dicionário de dados refere-se a termos que são utilizados pelo operador quando estiver entrando dados para o sistema. No nosso sistema de custos, o dicionário incluirá todos os clientes válidos, bem como serviços, profissionais e tipos.

```
* sistema de custos – adicao dos arquivos diários
* por carl townsend
* 12/27/82
SELECT PRIMARY
USE b:lote
SELECT SECONDARY
USE b:sumario INDEX b:sumario
SELECT PRIMARY
STORE '          ' TO mdata
STORE '                    ' TO xclient
STORE 0.00 TO xhrs
DO WHILE t
        * Inicialize as variáveis
        STORE  t  TO  xmarca
        STORE '                    ' TO mclient
        STORE '          ' TO mserv
        STORE '                    ' TO  mdesc
        STORE '  ' TO mprof
        STORE '  ' TO mtip
        STORE 0.00 TO mhrs
        * seleciona o tipo de entrada
        DO WHILE xmarca
            ERASE
            RESTORE FROM TITULO ADDITIVE
            @ 1,1 SAY  '**********' +mtit+ ' **********'
            @ 2,1 SAY  'ENTRADA POR LOTES'
            @ 2,60 SAY DATE( )
            @ 5,1 SAY  'CLIENTE' GET mclient
            @ 6,1 SAY  'SERVICO' GET mserv
            @ 7,1 SAY  'DATA (MM/DD/AA) ' GET mdate PICTURE '99/99/99'
            @ 8,1 SAY  'DESCRICAO' GET mdesc
            @ 9,1 SAY  'PROFISSIONAL' GET mprof
            @ 10,1 SAY 'TIPO ' GET mtip
```

```
        @ 11,1 SAY 'HORAS ' GET mhrs
        @ 21,50 SAY 'ULTIMO CLIENTE ENTRADO:' +xclient
        @ 22,50 SAY 'ULTIMA HORA ENTRADA ' +str(xhrs,8,2)
        @ 23, 50 SAY 'ENTRE UM CLIENTE EM BRANCO PARA TERMINAR'
        READ
        IF mclient=' '
                RETURN
        ENDIF mclient=' '
        STORE STR(mhrs,14,2) TO y
        STORE $(y,12,3) TO y
        SELECT SECONDARY
        @ 5,30 SAY '                    '
        @ 6,30 SAY '                    '
        @ 9,30 SAY '                    '
        @ 10,30 SAY '                   '
        @ 11,20 SAY '                   '
        * verifica o dicionário para validação
        STORE 'C' +mclient TO x
FIND '&x'
STORE f TO errmara
IF #=0
        @ 5,30 SAY 'CLIENTE NAO CONSTA NO ARQUIVO'
        STORE t TO errmara
ENDIF #=0
STORE venda TO mvenda
STORE 'J' +mserv TO x
FIND '&x'
IF #=0
        @ 6,30 SAY 'SERVICO NAO CONSTA NO ARQUIVO'
        STORE t TO errmarc
ENDIF #=0
STORE 'E' +mprof TO x
FIND '&x'
IF #=0
        @ 9,30 SAY 'PROFISSIONAL NAO CONSTA NO ARQUIVO'
          STORE t TO errmarc
ENDIF #=0
STORE mcusto*mhrs TO taxa
STORE 'A' +mtrp TO x
FIND '&x'
IF #=0
                @   10,30 SAY 'TIPO NAO CONSTA DO ARQUIVO'
        STORE t TO errmara
ENDIF #=0
IF y<>'.25' .AND. y<> '.00' .AND. y<> '.75' .AND. y<> '.50'
        @ 11,30 SAY 'HORARIO INCORRETO'
```



```
        STORE t TO errmara
ENDIF y<> '.25' .AND. y<> '.00' .AND. y<> '.75'.AND. y <> '.50'
IF .not. errmara
        STORE f TO xmara
        LOOP
ENDIF .not. errmara
STORE 1 TO xx
                DO WHILE xx<35
                    @ 23,60 SAY chr(7)
                    STORE xx+1 TO xx
                ENDDO WHILE xx<35
        ENDDO WHILE xmarc
        SELECT PRIMARY
        * adicione o registro
        APPEND BLANK
        REPLACE client WITH mclient
        REPLACE data WITH $(mdata,7,2)+$(mdata,1,2)+$(mdata,4,2)
        REPLACE desc WITH mdesc, prof WITH mprof,tip WITH mtip,hrs;
                                                        WITH mhrs
        REPLACE custo WITH mcusto
        REPLACE venda WITH mvenda
        REPLACE lanc WITH f
        STORE mclient TO xclient
        STORE mhrs TO xhrs
        STORE '          ' TO mclient
        STORE '          ' TO mserv
        STORE '                    ' TO mdesc
        STORE '      ' TO mprof
        STORE '      ' TO mtip
        STORE 0.00 TO mhrs
ENDDO WHILE t
```

A saída deste programa é um arquivo de lotes para armazenar os clientes do sistema de custos. Os clientes, serviços, profissionais e os tipos válidos estão armazenados em um arquivo dicionário com um tipo de código. Após serem entrados, cada um é verificado contra o arquivo dicionário para validação. Se qualquer das entradas não for encontrada no dicionário, uma mensagem de erro aparecerá em frente à entrada incorreta e o alarme soará. Então, é permitida ao usuário corrigi-la. Note que os campos são limpos apenas se a entrada está correta e o usuário não necessita preenchê-los novamente.

VALIDAÇÃO DE DADOS

Ao entrar com os dados, você desejará verificá-los para estar certo que valores errados não sejam aceitos. Nós já vimos duas maneiras de fazer isso, a cláusula PICTURE no comando GET e o dicionário de dados. Dois outros métodos adicionais existem, por controle do programa e por controle de um arquivo de dados.

A cláusula PICTURE pode ser usada para verificar uma digitação correta. Por exemplo, se você espera um valor numérico para o CEP ou para um número de telefone, a cláusula PICTURE é excelente.

O dicionário de dados é usado para verificar conjuntos de caracteres já válidos. Os conjuntos válidos podem ser facilmente alterados pela atualização do arquivo do dicionário de dados.

O programa de controle é usado para verificar dados pela sua comparação com uma gama de valores conhecidos. Se o valor não pertencer a esta gama, a entrada é rejeitada e novamente solicitada. Um exemplo poderia ser um programa de impressão de etiquetas de endereçamento. O programa solicitaria o número de linhas por etiquetas (tamanho da etiqueta) e também o número de linhas a imprimir. Se o número de linhas a imprimir excede o número de linhas de cada etiqueta, o programa solicitaria novamente o número de linhas a imprimir.

O método do arquivo de dados é útil para minimizar o trabalho de entrada de dados pela obtenção de certos valores do arquivo de dados. Um exemplo poderia ser um programa de endereçamento postal no qual são entrados o endereço e o número do telefone. Se o usuário não entrar com o código DDD para o número telefônico, o programa poderá calculá-lo pela comparação do endereço com os demais existentes no arquivo.

OITO

Modificando Registros do Database

Uma vez que o database foi criado e os dados carregados, você, ocasionalmente, desejará modificar os dados. Isto envolve dois passos: encontrar e modificar o registro.

ENCONTRANDO O REGISTRO

Quando os registros foram adicionados ao database, os dados foram normalmente colocados em registros seqüenciais no database. Isto significa que se usar o comando LIST ALL em um arquivo não indexado, verá os registros na ordem em que você os colocou. Por exemplo, para a Oficina do Fred, você usaria

```
USE mestre
LIST ALL
```

Se você sabe o número do registro que quer editar, o registro será obtido simplesmente indo-se até ele:

```
GOTO 3
```

Como nós vimos resumidamente, a localização e a edição do registro podem ser combinadas em uma só instrução

```
EDIT #
```

NOTA: EDIT x, onde x é uma variável, não funcionará. Lembre-se de que # é uma função do dBASE II que representa o registro atual do database.

Se você conhece o conteúdo de um campo específico do registro, poderá encontrá-lo em um database não indexado utilizando o comando LOCATE da seguinte maneira:

```
LOCATE FOR disp=0
```

Você raramente usará este comando, uma vez que ele é muito demorado para databases de qualquer tamanho. Se vários comandos LOCATE são necessários em um programa, é melhor construir um índice e usá-lo para encontrar o registro.

Por exemplo, no arquivo de estoque da Oficina do Fred existe um índice baseado no número da peça; você poderá localizar uma peça utilizando o índice com

```
USE mestre INDEX mestre
STORE '001234' TO nopeca
FIND &nopeca
```

Na programação é muito útil o uso de "e comercial" (&) no comando FIND, uma vez que você pode facilmente mudar o valor da variável de maneira a encontrar um outro registro. Por exemplo:

```
STORE 1 TO x
STORE 'L'+STR(x,1) TO var
STORE 6 TO &var
```

Aqui nós criamos o nome de variável L1 que é armazenado na variável VAR. O terceiro comando armazena 6 na variável armazenada em VAR, o que armazenará 6 em L1. O "e comercial" permite ao usuário referenciar uma variável indiretamente.

Verifique sempre se você encontrou o registro correto antes de começar a editá-lo, para ter certeza de que você está editando aquilo que estava esperando. Se o comando FIND não for bem sucedido o número do registro sempre será zero.

```
FIND  '&nopeca'
IF #=0
        @ 23,1 SAY NOPECA +' NAO ESTA NO ARQUIVO MESTRE'
        STORE 1 TO xx
        DO WHILE xx<70
                STORE xx+1 TO xx
        ENDDO
        @ 23,60 SAY CHR(7)
        SKIP
        LOOP
ENDIF
```

MODIFICANDO PELO COMANDO EDIT

Uma vez que o registro desejado foi localizado, ele é facilmente modificado com o comando EDIT. Existem duas formas de fazer isto. A primeira consiste em chegar ao registro e então usar o comando EDIT #. A segunda é armazenar o número como um conjunto de caracteres e então fazer uma referência indireta a este número.

A primeira forma foi apresentada anteriormente como

```
STORE 3 TO x
GOTO x
EDIT #
```

A segunda forma, utilizando um conjunto de caracteres, pode ser usada desta forma

```
STORE '3' TO x
EDIT &x
```

A seguinte forma não funcionará:

```
STORE 3 TO x
EDIT x
```

A próxima forma funcionará, pois o número é assumido como sendo um conjunto de caracteres.

```
EDIT 3
```

Uma vez que o comando EDIT foi emitido, a tela será limpa e o registro será mostrado na tela (veja a Figura 8-1) com os valores atuais. O cursor pode ser movido para frente ou para trás através dos campos, modificando-os quando necessário pelos controles do cursor listados na Tabela 8-1. Os registros alterados são recolocados no arquivo teclando-se CONTROL-W. Se você desejar manter o registro inalterado, pode terminar a edição teclando CONTROL-Q.

Os registros são marcados para serem eliminados pelo CONTROL-U. O registro eliminado permanecerá no arquivo até que você emita um comando PACK. O uso do EDIT é muito prático para arquivos pequenos, mas você raramente o usará em programas de produção. Com o comando EDIT o usuário está sempre trabalhando com um arquivo "vivo", ou seja, você está entrando dados diretamente no arquivo. Não existe acompanhamento das mudanças efetuadas nos registros, bem

```
RECORD #  000003
NOPECA    :000003:
DESC      :CORREIA DO VENTILADOR:
NOTAS     :
DISP      :00002:
CUSTO     :    12.50:
PRECO     :    15.50:
```

Figura 8-1. Tela do comando EDIT

Tabela 8-1. Controles para o comando EDIT

Comando	Ação
CONTROL-E ou CONTROL-A	Volta um campo de dados
CONTROL-X ou CONTROL-F	Avança um campo de dados
CONTROL-S	Volta um caractere
CONTROL-D	Avança um caractere
CONTROL-Y	Limpa o campo com brancos
CONTROL-G	Elimina o caractere sob o cursor
RUBOUT	Elimina o caractere anterior ao cursor
CONTROL-Q	Termina a edição sem alterar o registro
CONTROL-W	Termina a edição e salva as alterações do registro
CONTROL-U	Marca o registro para ser eliminado
CONTROL-R	Grava o registro e volta um registro no arquivo
CONTROL-C	Grava o registro e avança um registro no arquivo
CONTROL-V	Marca para inserção

como por quem ou quando foram efetuadas. É preferível editar sob o controle de um programa, o que será discutido posteriormente neste capítulo.

O COMANDO BROWSE

Um outro editor poderoso é o comando BROWSE. Utilizando o comando BROWSE vários registros podem ser mostrados e editados por vez. Por exemplo, use o comando GOTO 1 para posicionar-se no primeiro registro e emita o comando indicando os campos a serem apresentados da seguinte maneira:

```
BROWSE FIELDS nopeca, desc, disp
```

Isto permite que você altere o arquivo a partir da tela (veja Figura 8-2). Se o número de caracteres de cada registro for maior que 80, este terá uma parte fora da tela. Use CONTROL-B e CONTROL-Z para movimentar-se à esquerda ou à direita.

Os campos podem ser editados utilizando-se os comandos da Tabela 8-2.

A tela pode ser considerada uma janela sobre o database e esta janela pode ser movimentada para cima, para baixo, para a esquerda ou para a direita, editando o registro ao ser movimentada.

Como o comando EDIT, o comando BROWSE é perigoso e raramente usado em programas. Você estará trabalhando diretamente no arquivo de dados. Não existe um acompanhamento das mudanças, bem como por quem, quando ou onde foram efetuadas. É muito prático para pesquisas rápidas, mas deve ser evitado em trabalhos de produção.


```
RECORD # :00001
NOPECA   DESC                          DISP
000001   CABO DA BATERIA                  4
000002   RELE                             0
000003   CORREIA DO VENTILADOR            2
000004   REGULADOR DE VOLTAGEM            2
000005   JUNTA UNIVERSAL                  1
000006   CONJ. PINO DO FREIO              6
000007   GUARNICAO DO FREIO               0
000008   MONTAGEM DO SILENCIOSO TRASEIRO  1
000009   LANTERNA LATERAL (E)             2
000010   LANTERNA LATERAL (D)             2
```

Figura 8-2. Tela do comando BROWSE

Tabela 8-2. Controles para o comando BROWSE

CONTROL-A ou CONTROL-E	Volta ao campo anterior
CONTROL-B	Movimenta horizontalmente um campo para a direita
CONTROL-C	Grava o registro atual e avança para o próximo registro
CONTROL-D	Avança um caractere
CONTROL-F ou CONTROL-X	Avança um campo
CONTROL-G	Elimina o caractere sob o cursor
CONTROL-Q	Termina sem efetuar alterações
CONTROL-R	Grava o registro atual e volta um registro
CONTROL-S	Volta um caractere
CONTROL-U	Marca o registro para ser eliminado
CONTROL-V	Liga ou desliga a chave do modo de inserção
CONTROL-W	Termina efetuando as mudanças
CONTROL-Y	Limpa o campo com brancos
CONTROL-Z	Movimenta horizontalmente um campo para a esquerda
RUBOUT	Elimina o caractere antes do cursor

EDITANDO SOB O CONTROLE DE UM PROGRAMA

A melhor maneira de alterar registros é sob o controle de um programa utilizando os seguintes procedimentos:

1. Abrir o arquivo
2. Entrar o valor do campo chave
3. Encontrar o registro pelo comando FIND e verificar se o registro foi encontrado
4. Guarde os valores atuais do registro em variáveis na memória
5. Mostre e edite estas variáveis
6. Altere o registro a partir dessas variáveis com o comando REPLACE
7. Feche o arquivo

A seguir temos o exemplo do programa de edição da Oficina do Fred:

```
* edita o arquivo de estoque
* por carl townsend
* 12/27/82
SELECT PRIMARY
USE mestre INDEX mestre
RESTORE FROM TITULO ADDITIVE
STORE t TO marca
DO WHILE t
        * obtem o numero da peca
        STORE '      ' TO mno
        ERASE
        @ 1,1 SAY '**********'+mtit+' **********'
        @ 2,1 SAY 'EDITANDO O ESTOQUE'
        @ 2,60 SAY DATE()
        @ 15,1 SAY 'ENTRE O NUMERO DA PECA' GET mno PICTURE '999999'
        @ 23,1 SAY 'ENTRE EM BRANCO PARA TERMINAR'
        READ
        IF $(mno,1,3)='   '
            RETURN
        ENDIF $(mno,1,3)='   '
        FIND '&mno'
        IF #=0
                @ 23,60 SAY CHR(7)
                @ 23,1 SAY 'NUMERO NAO CONSTA NO ARQUIVO'
                STORE 1 TO xx
                DO WHILE xx<35
                        STORE xx+1 TO xx
                ENDDO WHILE xx<35
                LOOP
        ENDIF # = 0
        ERASE
        @ 1,1 SAY '**********'+mtit+' **********'
        @ 2,1 SAY 'EDITANDO O ESTOQUE'
        @ 2,60 SAY DATE()
        * Coloca os valores anteriores em variáveis
        STORE desc TO mdesc
```



```
        STORE disp TO mdisp
        STORE custo TO mcusto
        STORE notas TO mnotas
        STORE preco TO mpreco
        * obtem os novos valores
        @ 5,1 say 'NUMERO DA PECA'+mno
        @ 6,1 say 'DESCRICAO 'GET mdesc
        @ 7,1 say 'DISP' GET mdisp
        @ 8,1 SAY 'CUSTO' GET mcusto
        @ 9,1 SAY 'PRECO' GET mpreco
        @ 10,1 SAY 'NOTAS' GET mnotas
        READ
        * substitua os valores
        REPLACE desc WITH mdesc,notas WITH mnotas
        REPLACE disp WITH mdisp,custo WITH mcusto,preco WITH mpreco
        ENDDO WHILE t
```

```
********** SISTEMA DE ESTOQUE **********
EDITANDO O ESTOQUE                    04/30/83

NUMERO DA PECA  000003
DESCRICAO       :CORREIA DO VENTILADOR   :
DISP            : 2:
CUSTO           :12.50:
PRECO           : 15.50:
NOTAS           :                        :
```

Figura 8-3. Edição sob o controle de um programa

Note que o conteúdo do arquivo não será visto diretamente pelo usuário, apenas as variáveis com aqueles dados. O usuário, também, nunca alterará os valores diretamente no arquivo. O arquivo é fechado pelo menu após o término do programa. Este programa é simples, mas ainda contém alguns elementos perigosos. Não existe um registro de cada alteração nem um arquivo de acompanhamento foi criado fazendo com que não se saiba quem efetuou qual mudança ou quando estas mudanças foram efetuadas.

ARQUIVOS DE ACOMPANHAMENTO E DICIONÁRIOS

Existem dois métodos de acompanhar ou controlar mudanças em um arquivo mestre:

1. Um arquivo secundário pode ser aberto como arquivo de acompanhamento. O menu principal solicita o nome do usuário que será mantido em uma variável na memória.

Para cada registro modificado no arquivo-mestre um registro é adicionado ao arquivo secundário de acompanhamento. O registro do arquivo de acompanhamento contém o nome de quem efetuou a mudança, a data, o valor antigo no arquivo-mestre, bem como o novo valor. Este arquivo pode ser impresso e eliminado quando necessário no acompanhamento das mudanças (veja Figura 8-4).

2. As informações do registro a ser editado são colocadas em um arquivo de lotes. Este arquivo pode ser impresso, testado e modificado tantas vezes quanto forem necessárias até que esteja correto. Então, o arquivo é "lançado" no arquivo-mestre. A cópia impressa do arquivo torna-se o registro de acompanhamento (veja a Figura 8-5).

Na página seguinte está um exemplo de um programa que usa um arquivo de lote para efetuar as edições para o sistema de custos. Este exemplo também mostra como o dicionário de dados pode ser usado para controlar a entrada dos dados durante a edição para uma maior exatidão.

Neste exemplo apenas os clientes, serviços, profissionais e tipos válidos serão aceitos como entradas para o arquivo de lotes.

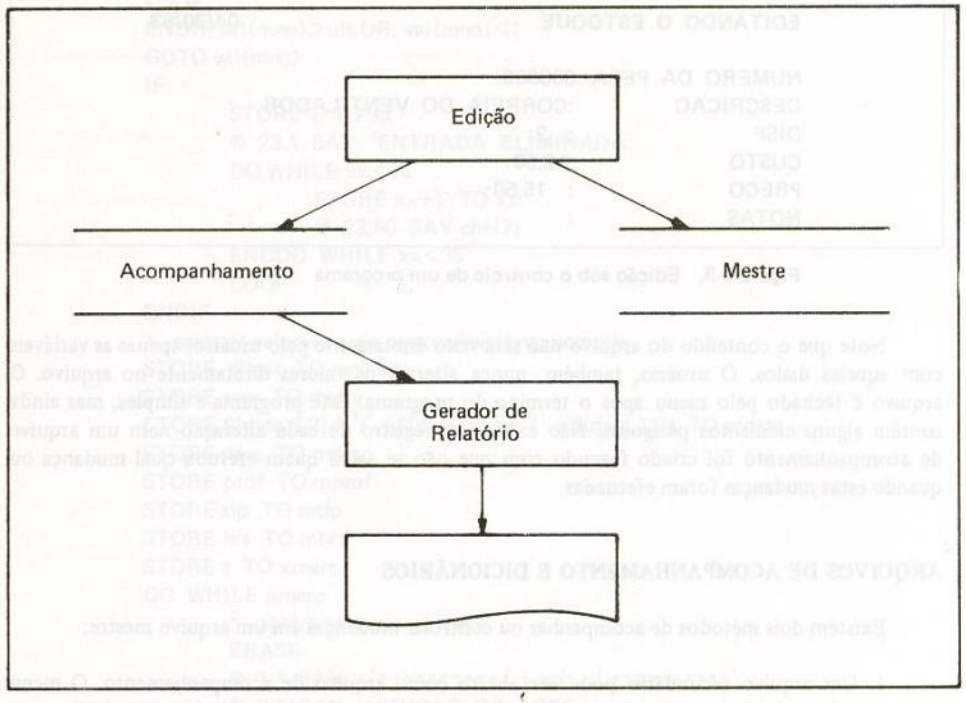


Figura 8-4. Edição com arquivo de acompanhamento

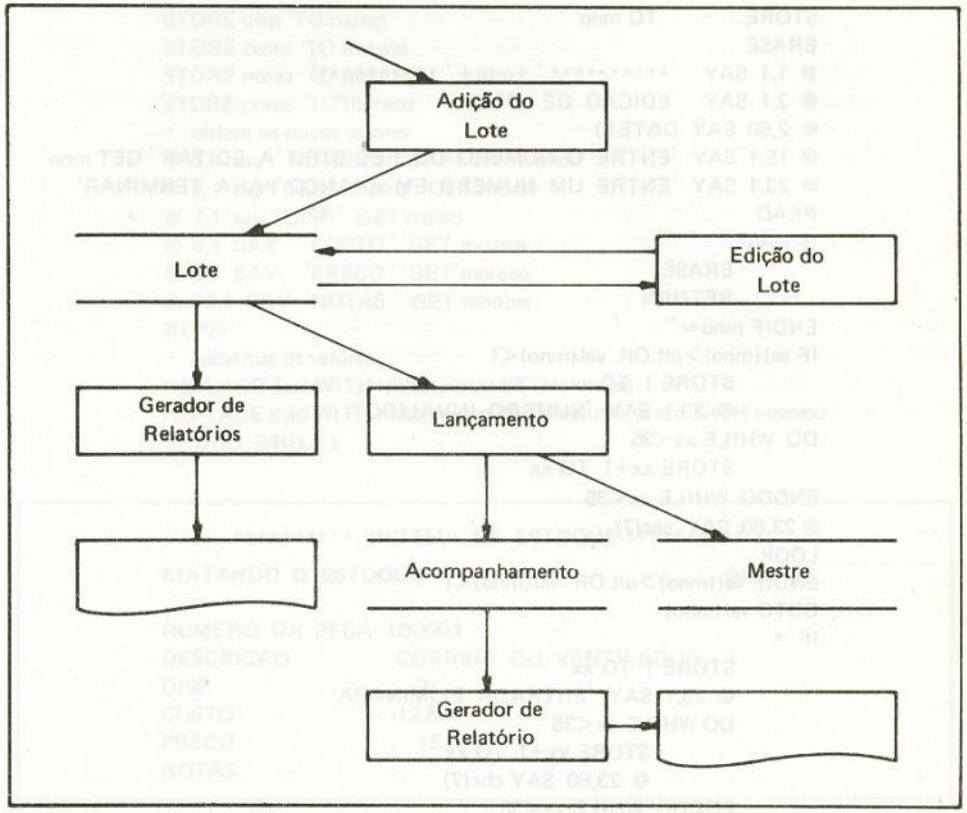


Figura 8-5. Edição com um arquivo de lote e de acompanhamento

```
* sistema de custos -- edicao do arquivo de lotes
* por carl townsend
* lotedit
* 12/29/82
SELECT PRIMARY
USE b: lote
GO BOTTOM
STORE #TO ult
SELECT SECONDARY
USE b:sumario INDEX b:sumario
SELECT PRIMARY
RESTORE FROM TITULO ADDITIVE
DO WHILE t
        * obtem o numero do registro a editar
```

```
STORE '      ' TO mno
ERASE
@ 1,1 SAY  '**********' +mtit+ ' **********'
@ 2,1 SAY  'EDICAO DE LOTES'
@ 2,60 SAY DATE( )
@ 15,1 SAY 'ENTRE O NUMERO DO REGISTRO A EDITAR ' GET mno
@ 23,1 SAY 'ENTRE UM NUMERO EM BRANCO PARA TERMINAR'
READ
IF mno='      '
        ERASE
        RETURN
ENDIF mno='      '
IF val(mno)>ult.OR. val(mno)<1
        STORE 1 TO xx
        @ 23,1 SAY 'NUMERO INVALIDO'
DO WHILE xx<35
        STORE xx+1 TO xx
ENDDO WHILE xx<35
@ 23,60 SAY chr(7)
LOOP
ENDIF val(mno)>ult.OR. val(mno)<1
GOTO val(mno)
IF *
        STORE 1 TO xx
        @ 23,1 SAY 'ENTRADA ELIMINADA'
        DO WHILE xx<35
                STORE xx+1 TO xx
                @ 23,60 SAY chr(7)
        ENDDO WHILE xx<35
        LOOP
ENDIF *
* passa os valores atuais para variaveis na memoria
STORE client TO mclient
STORE serv TO mserv
STORE $(date,3,2)+'/'+$(date,5,2)+'/'+$(date,1,2) TO mdata
STORE desc TO mdesc
STORE prof TO mprof
STORE tip TO mtip
STORE hrs TO mhrs
STORE t TO xmarc
DO WHILE xmarc
        * obtem as novas variaveis
        ERASE
        @ 1,1 SAY  '**********' +mtit+ ' **********'
        @ 2,1 SAY  'EDICAO DE LOTE'
        @ 2,60 SAY DATE( )
```



```
        @ 5,1 SAY  'CLIENTE ' GET mclient
        @ 6,1 SAY  'SERVICO ' GET mserv
        @ 7,1 SAY  'DATA (MM/DD/AA) ' GET mdata PICTURE '99/99/99'
        @ 8,1 SAY  'DESCRICAO ' GET mdesc
        @ 9,1 SAY  'PROFISSIONAL ' GET mprof
        @ 10,1 SAY ' TIPO ' GET mtip
        @ 11,1 SAY 'HORAS ' GET mhrs
        @ 23,1 SAY 'MODIFIQUE OS CAMPOS NECESSARIOS E COLOQUE O;
                                        CURSOR NO ULTIMO '
        READ
        IF mclient='      '
                ERASE
                RETURN
        ENDIF mclient='      '
        STORE STR(mhrs,14,2) TO y
        STORE $(y,12,3) TO y
        * verifica a validade dos registros
        SELECT SECONDARY
        @ 5,30 SAY '      '
        @ 6,30 SAY '      '
        @ 9,30 SAY '      '
        @ 10,30 SAY '      '
        @ 11,20 SAY '      '
        STORE 'C' +mclient TO x
        FIND '&x'
        STORE f TO errmarc
        IF #=0
                @ 5,30 SAY 'CLIENTE NAO CONSTA NO ARQUIVO'
                STORE t TO errmarc
        ENDIF #=0
        STORE 'J' +mserv TO x
        FIND '&x'
        STORE F TO errmac
        IF #=0
                @ 6,30 SAY 'SERVICO NAO CONSTA NO ARQUIVO'
                STORE t TO errmarc
ENDIF #=0
STORE 'E' + mprof TO x
FIND '&x'
STORE F TO errmarc
IF #=0
        @ 9,30 SAY 'PROFISSIONAL NAO CONSTA NO ARQUIVO'
        STORE t TO errmarc
ENDIF #=0
STORE taxa*mhrs TO mcusto
STORE 'A' +mtip TO x
FIND '&x'
STORE F TO errmarc
IF #=0
```

```
            @ 10,30 SAY 'TIPO NAO CONSTA NO ARQUIVO'
            STORE t TO errmarc
        ENDIF #=0
        IF y<>'.25' .AND. y<>'.00' .AND. y<>'.75' .AND. y<>'.50'
            @ 11,30 SAY 'HORARIO INCORRETO'
            STORE t TO errmarc
        ENDIF y<>'.25' .AND. y<>'.00' .AND. y<>'.75' .AND. y<>'.50'
        IF .NOT. errmarc
            STORE f TO xmarc
            LOOP
        ENDIF .NOT. errmarc
        STORE 1 TO xx
        DO WHILE xx<70
                    STORE xx+1 TO xx
            ENDDO WHILE xx<70
            @ 23,60 SAY chr(7)
        ENDDO WHILE xmarc
        * carrega os novos valores
        SELECT PRIMARY
        REPLACE client WITH mclient
        REPLACE serv WITH mserv
        REPLACE data WITH $(mdata,7,2)+$(mdata,1,2)+$(mdata,4,2)
        REPLACE desc WITH mdesc,prof WITH mprof,tip WITH mtip, hrs WITH
                                                                mhrs
        REPLACE custo WITH mcusto
        STORE '      ' TO mclient
        STORE '      ' TO mserv
        STORE '                ' TO mdesc
        STORE '      ' TO mprof
        STORE '      ' TO mtip
        STORE 0.00 TO mhrs
    ENDDO WHILE t
```

ELIMINANDO REGISTROS

O dBASE II oferece dois métodos para eliminar registros. Um método é usar a tecla CONTROL-U durante uma edição pelo comando EDIT. Uma mensagem aparecerá na tela informando que o registro foi eliminado e quaisquer listagens subseqüentes do arquivo mostrarão este registro com um asterisco. O registro não é fisicamente removido do arquivo é apenas *marcado* para ser removido. Todos os registros no arquivo mantêm o seu número de registro.

Se o registro for novamente editado e novamente o CONTROL-U for utilizado, a marca para ser eliminado será retirada e o registro não mais será eliminado.

A um determinado tempo o usuário poderá emitir o comando PACK para remover fisicamente todos os registros eliminados do arquivo. A numeração dos registros mudará e os registros

eliminados não mais estarão no arquivo. Se o comando PACK for executado sobre um arquivo indexado que tenha sido aberto com o índice, este será automaticamente ajustado.

Para os grandes arquivos você verá que é mais rápido abrir o arquivo sem o índice, compactar o arquivo e reindexá-lo. Por exemplo:

```
USE mestre
PACK
INDEX ON nopeca TO mestre
```

Esta maneira de eliminar registros tem os mesmos problemas que mencionamos anteriormente quanto ao uso dos comandos APPEND e EDIT; não existe um acompanhamento da eliminação, bem como não há registro de quem eliminou ou quando as eliminações foram efetuadas. É muito melhor eliminar registros sob controle de um programa usando o comando DELETE, como por exemplo:

```
* sistema de custos – elimina do arquivo sumario
* por carl townsend
* 12/27/82
SELECT PRIMARY
USE b:sumario INDEX b:sumario
STORE t TO marc
RESTORE FROM TITULO ADDITIVE
DO WHILE t
    * identifica o tipo de entrada
    DO WHILE marc
        ERASE
        @ 1,1 SAY  '********** ' +mtit+ ' **********'
        @ 2,1 SAY  'ELIMINANDO DO ARQUIVO SUMARIO'
        @ 2,60 SAY DATE()
        @ 5,1 SAY  'ESCOLHA O TIPO DE ENTRADA:  '
        @ 6,1 SAY  '    0 = TERMINA O PROGRAMA'
        @ 7,1 SAY  '    1 = CLIENTE'
        @ 8,1 SAY  '    2 = SERVICO'
        @ 9,1 SAY  '    3 =  PROFISSIONAL'
        @ 10,1 SAY '    4 = TIPO'
        STORE ' ' TO pesq
        @ 15,1 SAY 'ENTRE COM O TIPO ESCOLHIDO  '
    WAIT TO pesq
    DO CASE
        CASE pesq='0'
            ERASE
            RETURN
        CASE pesq='1'
            STORE 'C' TO mtipo
            STORE f TO marc
            STORE '          ' TO mtemp
            STORE 'CLIENTE' TO mnome
```

```
                CASE pesq='2'
                        STORE 'J' TO mtipo
                        STORE f TO marc
                        STORE '          ' TO mtemp
                        STORE 'SERVICO' TO mnome
                CASE pesq='3'
                        STORE 'E' TO mtipo
                        STORE f TO marc
                        STORE '          ' TO mtemp
                        STORE 'PROFISSIONAL' TO mnome
                CASE pesq='4'
                        STORE 'A', TO mtipo
                        STORE f TO marc
                        STORE '          ' TO mtemp
                        STORE 'TIPO' TO mnome
        ENDCASE
ENDDO WHILE marc
STORE t TO xmarc
DO WHILE xmarc
                STORE '          ' TO mtemp
        ERASE
        @ 1,1 SAY  '**********' +mtit+ ' **********'
        @ 2,1 SAY  'ELIMINANDO' +mnome+ 'DO ARQUIVO'
        @ 2,60 SAY DATE ( )
        @ 15,1 SAY 'ENTRE ' + 'mnome + '    ' GET mtemp
        @ 23,1 SAY 'ENTRE EM BRANCO PARA TERMINAR'
        READ
        IF $(mtemp,1,3)='      '
                ERASE
                @ 15,1 SAY 'INDEXANDO ' +mnome+ ' ...'
                PACK
                STORE f TO xmarca
                STORE t TO marca
                LOOP
        ENDIF $(mtemp,1,3)= '    '
        * decide e elimina
        IF pesq='1' .OR. pesq='2'
                STORE mtemp TO xnome
        ELSE
                STORE mtemp + '    ' TO xnome
        ENDIF pesq='1' .OR. pesq='2'
        STORE mtipo+nome TO x
        FIND '&x'
        IF #<>0 .AND. .not. *
                DELETE
                @ 23,1 SAY 'ELIMINANDO O REGISTRO'
```



```
                STORE 1 TO xx
                DO WHILE xx<70
                        STORE xx+1 TO xx
                ENDDO WHILE xx<70
                ELSE
                        @ 23,1 SAY 'REGISTRO NAO CONSTA NO ARQUIVO'
                        STORE 1 TO xx
                        DO WHILE xx<70
                                @ 23,60 SAY chr(7)
                                STORE xx+1 TO xx
                        ENDDO WHILE xx<70
                ENDIF #<>0
        ENDDO WHILE xmarc
ENDDO WHILE t
```

Neste exemplo, o valor da chave é passado e uma pesquisa é realizada com base neste valor por meio do índice. Se o registro não for encontrado, o usuário receberá uma mensagem de que o registro não está no arquivo. Se o registro for encontrado o seu valor será apresentado e, ao usuário, é solicitada a confirmação da eliminação. Se o usuário confirmar, um simples comando DELETE eliminará o registro. O registro pode ficar no arquivo e ser eliminado fisicamente a qualquer outro momento ou o comando PACK e uma reindexação podem ser feitos imediatamente.

Este processo dará um maior controle sobre a eliminação dos registros, mas ele não nos dá um registro de acompanhamento destas eliminações. Em uma edição normal existem dois métodos para controlar o arquivo mestre. Os registros podem ser eliminados imediatamente e ao mesmo tempo gravados em um arquivo secundário de acompanhamento, ou, um arquivo lote de eliminações pode ser criado e as eliminações serem feitas mais tarde (à noite talvez) guardando-se uma listagem do arquivo lote. Também é uma boa medida, ao se eliminar registros, salvar um deles no arquivo de acompanhamento do registro eliminado acompanhado da data.

NOVE

Processamento Baseado em Transações

Um dos mais importantes fatores em qualquer desenho de sistema é que o ciclo de entrada de dados seja mantido com o usuário. O processamento baseado em transações é um método de isolar uma rápida entrada de dados de qualquer processamento demorado. Uma vez que o dBASE II só pode manter dois arquivos abertos por vez e uma vez que muitos sistemas exigem três ou mais arquivos durante o processamento, um sensível retardamento poderá ser sentido se o processamento dos dados é feito ao mesmo tempo que a entrada dos dados.

Um segundo propósito do processamento baseado em transações é isolar a entrada dos dados do processamento e do ciclo de atualização do arquivo. Este isolamento do arquivo lhe dá uma proteção em caso de quedas acidentais do sistema.

A filosofia básica do processamento baseado em transações é colocar os dados de entrada em um arquivo de lote temporário não-indexado. Este arquivo pode ser listado, editado e comparado tantas vezes quantas forem necessárias até que esteja correto. Uma vez que muito pouco ou nenhum processamento é feito, a entrada dos dados é muito rápida.

Após a transação ou arquivo de lote estar correto, ele é lançado no arquivo ou arquivos de dados. O lançamento é o processo de atualizar os dados de um arquivo mestre pelo conteúdo do arquivo de lote. Se os relatórios são mensais, o lançamento pode ser feito várias vezes durante o mês ou mesmo diariamente. Qualquer processamento é executado durante o lançamento quando um ou mais arquivos mestres são atualizados.

O arquivo de transações normalmente não deverá conter mais do que um dia de processamento. A listagem do arquivo deve, se possível, incluir o total do lote. Isto permite ao usuário verificar se não houve perda de dados quando da sua entrada ou de qualquer outro erro de entrada.

O arquivo de transações não é indexado. Se o sistema cair, simplesmente adicione novas entradas a partir da última existente no arquivo. Não é necessário reconstruir os índices ou fazer uma recuperação elaborada.

MÉTODO 1 – PROCESSAMENTO ALTERNATIVO

Este método é ideal para sistemas de estoques (como o da Oficina do Fred) no qual as peças não são adicionadas ou eliminadas freqüentemente como é a quantidade disponível de cada uma. As peças acrescentadas ou eliminadas são processadas imediatamente sobre o arquivo mestre do estoque e um registro da mudança efetuada é feito no arquivo de acompanhamento. A mudança da quantidade disponível, no entanto, utiliza um arquivo de transações.

Ao modificar-se um registro, o arquivo mestre e o de transações estão abertos. Quando o valor da chave é passado, o registro no arquivo *mestre* é lido e as informações são passadas para variáveis na memória e apresentadas ao usuário.

O usuário, então, modificará um ou mais campos. Rotinas examinarão e compararão cada variável com o registro mestre. Para cada campo diferente será efetuada uma entrada no arquivo de transações com o nome do campo, data, nome da pessoa que efetuou a mudança e do novo

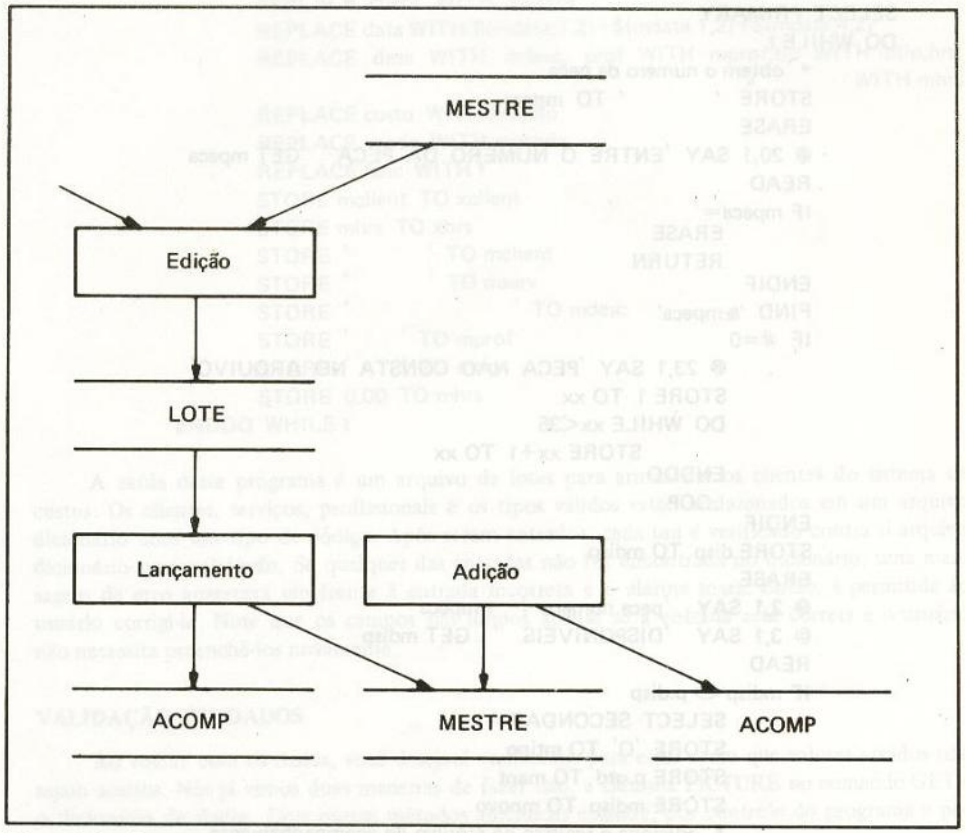


Figura 9-1. Processamento baseado em transações para o sistema de estoque

valor. Este arquivo de transações será processado posteriormente, criando no arquivo de acompanhamento suas próprias entradas que se juntarão às existentes de adições e eliminações feitas diretamente. Cada campo a ser atualizado no arquivo mestre tem um registro no arquivo de acompanhamento. Os registros serão pequenos, mas muitos registros podem existir para cada registro mestre alterado (veja a Figura 9-1). Este programa mostra como a Oficina do Fred altera a quantidade disponível pelo armazenamento da nova quantidade disponível em um arquivo de lote.

```
* programa para alterar a quantidade disponível de pecas usando arquivo de acompanha-
* mento
* por carl townsend
* 05/10/83
SET TALK OFF
SELECT PRIMARY
USE mestre INDEX mestre
SELECT SECONDARY
USE acomp
SELECT PRIMARY
DO WHILE t
        * obtem o numero da peca
        STORE '          ' TO mpeça
        ERASE
        @ 20,1 SAY 'ENTRE O NUMERO DA PECA ' GET mpeca
        READ
        IF mpeca='          '
                ERASE
                RETURN
        ENDIF
        FIND '&mpeca'
        IF #=0
                @ 23,1 SAY 'PECA NAO CONSTA NO ARQUIVO'
                STORE 1 TO xx
                DO WHILE xx<35
                        STORE xx+1 TO xx
                ENDDO
                LOOP
        ENDIF
        STORE disp TO mdisp
        ERASE
        @ 2,1 SAY  'peca numero ' +mpeca
        @ 3,1 SAY  'DISPONIVEIS '  GET mdisp
        READ
        IF mdisp<>p.disp
                SELECT SECONDARY
                STORE 'Q' TO mtipo
                STORE p.qtd TO mant
                STORE mdisp TO mnovo
                * adiciona o registro no arquivo do acompanhamento
                APPEND BLANK
                REPLACE peca WITH mpeca, tipo WITH mtipo
                REPLACE ant WITH mant, novo WITH mnovo
                REPLACE data WITH mdata, ini WITH mini
        ENDIF
ENDDO
```


MÉTODO 2 – PROCESSAMENTO COM ARQUIVO DE TRANSAÇÕES

Este método é ideal para sistemas de livros contábeis no qual cada registro é carregado no arquivo de transações e posteriormente adicionado em um arquivo de detalhes (Figura 9-2).

A estrutura dos arquivos de lote e de detalhes é idêntica. O arquivo de lotes é criado adicionando-se registros seqüencialmente; este arquivo pode ser listado e editado. O arquivo de lotes é então juntado ao arquivo de detalhes. Este método é prático quando vários registros são adicionados aos arquivos de produção e muito pouca edição será efetuada após o lançamento (como em um Diário Geral). Aqui está um programa para adicionar lotes em um sistema contábil utilizando este tipo de processamento do arquivo de transações:

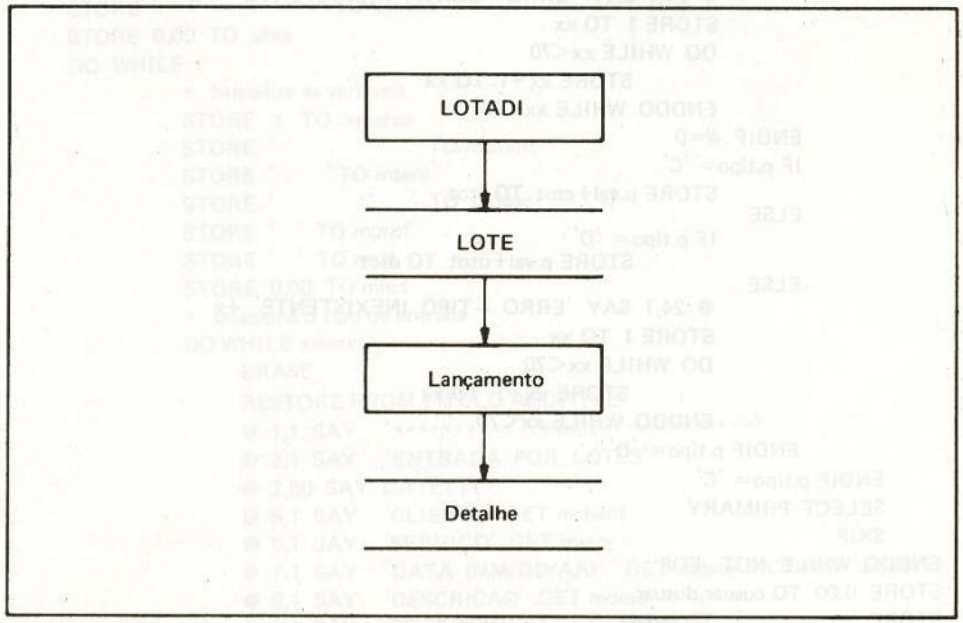


Figura 9-2. Processamento baseado em transações para um sistema de livros contábeis

```
* adiciona lotes ao livro diario
* por carl townsend
*  12/16/82
*  lotadi
ERASE
RESTORE FROM TITULO ADDITIVE
@ 1,1 SAY  '**********' +mtit+  ' **********'
@ 2,1 SAY  'ADICAO DE LOTE'
@ 2,60 SAY DATE( )
@ 15,1 SAY 'ACUMULANDO O TOTAL ATUAL DOS LOTES...'
USE dilote
SELECT SECONDARY
USE conta INDEX conta
STORE 0.00 TO ctot,dtot
SELECT PRIMARY
DO WHILE .NOT. EOF
        IF *
                SKIP
        ENDIF * LOOP
        STORE $(cont,1,1)+ '000' TO x
        SELECT SECONDARY
        FIND '&x'
        IF #=0
                @ 24,1 SAY 'ERRO- CONTA INEXISTENTE ' +x
                STORE 1 TO xx
                DO WHILE xx<70
                        STORE xx+1 TO xx
                ENDDO WHILE xx<70
        ENDIF #=0
        IF p.tipo= 'C'
                STORE p.val+ctot TO ctot
        ELSE
                IF p.tipo= 'D'
                        STORE p.val+dtot TO dtot
        ELSE
                @ 24,1 SAY 'ERRO -TIPO INEXISTENTE' +x
                STORE 1 TO xx
                DO WHILE xx<70
                        STORE xx+1 TO xx
                ENDDO WHILE xx<70
        ENDIF p.tipo= 'D'
    ENDIF p.tipo= 'C'
    SELECT PRIMARY
    SKIP
ENDDO WHILE .NOT. EOF
STORE 0.00 TO cusuar,dusuar
STORE '        ' TO mdata
DO WHILE t
    * obtem numero da conta
```



```
STORE '0000' TO mcont
ERASE
@ 1,1 SAY  '**********'+mtit+ ' **********'
@ 2,1 SAY  'ADICAO DE LOTE'
@ 2,60 SAY DATE( )
@ 4,1 SAY  'CONTA NUMERO ' GET mcont PICTURE '9999'
STORE cusuar-dusuar TO susuar
STORE ctot-dtot TO stot
@ 15,1 SAY 'CREDITO TOTAL DO USUARIO '   +STR(cusuar,14,2)
@ 16,1 SAY 'DEBITO TOTAL DO USUARIO '+STR(dusuar,14,2)
@ 17,1 SAY 'SALDO DO USUARIO         '+STR(susuar,14,2)
@ 19,1 SAY 'CREDITO TOTAL DO LOTE    '+STR(ctot,14,2)
@ 20,1 SAY 'DEBITO TOTAL DO LOTE     '+STR(dtot,14,2)
@ 21,1 SAY 'SALDO DO LOTE            '+STR(stot,14,2)
@ 23,1  SAY 'ENTRE UM NUMERO DE CONTA EM BRANCO OU COM ZEROS PARA;
       TERMINAR'
READ
IF VAL(mcont)=0
        RETURN
        ENDIF VAL(mcont)=0
        SELECT SECONDARY
        FIND &mcont
        IF #=0
                @ 6,1 SAY  'NUMERO DE CONTA NAO CONSTA NO ARQUIVO '
                STORE 1 TO xx
        DO WHILE xx<70
                STORE xx+1 TO xx
        ENDDO WHILE xx<70
        SKIP
        LOOP
ENDIF #=0
STORE desc TO mdesc
STORE $(mcont,1,1)+ '000' TO x
FIND &x
IF #=0
        @ 6,1 SAY  'GRUPO DE CONTA INEXISTENTE '
        STORE 1 TO xx
        DO WHILE xx<70
                STORE xx+1 TO xx
        ENDDO WHILE xx<70
        SKIP
        LOOP
ENDIF #=0
IF tipo<> 'C' .AND.  tipo<> 'D'
        @ 6,1 SAY  'TIPO DE CONTA INEXISTENTE'
```

```
    SKIP
    LOOP
ELSE
    STORE tipo TO mtipo
ENDIF tipo<> 'C' .AND. tipo<> 'D'
STORE '                              ' TO mref
STORE 0.00 TO mval
SELECT PRIMARY
    ERASE
    @ 1,1 SAY  '********** ' +mtit+ '**********'
    @ 2,1 SAY  'ADICAO DE LOTE                '
    @ 2,60 SAY DATE( )
    @ 4,1 SAY  'CONTA NUMERO '  +mcont
    IF mtipo= 'C'
        @ 4,40 SAY 'ENTRADA NORMAL DE CREDITO'
    ELSE
        @ 4,40 SAY 'ENTRADA NORMAL DE DEBITO'
    ENDIF mtipo= 'C'
    @ 5,1 SAY  'DESCRICAO DA CONTA ' +mdesc
    @ 6,1 SAY  'DATA '  GET mdata PICTURE '99/99/99'
    @ 7,1 SAY  'VALOR  ' GET mval
    @ 8,1 SAY  'REFERENCIA '  GET  mref
    STORE cusuar-dusuar TO susuar
    STORE ctot-dtot TO stot
    @ 15,1 SAY 'CREDITO TOTAL DO USUARIO '  +STR(cusuar,14,2)
    @ 16,1 SAY 'DEBITO TOTAL DO USUARIO  '  +STR(dusuar,14,2)
    @ 17,1 SAY 'SALDO DO USUARIO         '  +STR(susuar,14,2)
    @ 19,1 SAY 'CREDITO TOTAL DO LOTE    '  +STR(ctot,14,2)
    @ 20,1 SAY 'DEBITO TOTAL DO LOTE     '  +STR(dtot,14,2)
    @ 21,1 SAY 'SALDO DO LOTE            '  +STR(stot,14,2)
    @ 23,1 SAY 'ENTRE O CAMPO DE REFERENCIA EM BRANCO PARA TERMINAR'
    READ
    STORE $(mdata,1,2) +$(mdata,4,2)+$(mdata,7,2) TO mmdata
    IF $(mref,1,2)='  '
            LOOP
        ENDIF $(mref,1,2)='  '
        STORE mtipo TO xtipo
        IF mval<0 .AND. mtipo= 'C'
            STORE 'D' TO xtipo
            STORE -mval TO mval
        ELSE
            IF mval<0 .AND. mtipo= 'D'
                STORE 'C' TO xtipo
                STORE -mval TO mval
        ENDIF mval<0 .AND. mtipo= 'D'
```



```
    ENDIF mval<0 .AND. mtipo= 'C'
    * adiciona o registro
    APPEND BLANK
    REPLACE cont WITH mcont
    REPLACE data WITH mdata
    REPLACE tipo WITH xtipo
    REPLACE val WITH mval
    REPLACE ref WITH mref
    IF xtipo= 'C'
            STORE ctot+val TO ctot
            STORE cusuar+val TO cusuar
    ELSE
            STORE dtot+val TO dtot
            STORE dusuar+val TO dusuar
    ENDIF xtipo= 'C'
    STORE cusuar-dusuar TO susuar
    STORE ctot-dtot TO stot
    @ 15,1 SAY 'CREDITO TOTAL DO USUARIO '  +STR(cusuar,14,2)
    @ 16,1 SAY 'DEBITO TOTAL DO USUARIO  '  +STR(dusuar,14,2)
    @ 17,1 SAY 'SALDO DO USUARIO         '  +STR(susuar,14,2)
    @ 19,1 SAY 'CREDITO TOTAL DO LOTE    '  +STR(ctot,14,2)
    @ 20,1 SAY 'DEBITO TOTAL DO LOTE     '  +STR(dtot,14,2)
    @ 21,1 SAY 'SALDO DO LOTE            '  +STR(stot,14,2)
ENDDO WHILE t
```

O CICLO DOS LANÇAMENTOS

Após o arquivo de transações estar pronto ele representa as mudanças a serem efetuadas em um outro arquivo. O processo de utilizar o arquivo de transações para atualizar um outro arquivo é chamado de *lançamento.*

A formação e as alterações em um arquivo de transações não alteram nenhum dos arquivos de produção. Outros arquivos de dados também permanecem inalterados e podem ser o arquivo de estoque de peças ou o arquivo de detalhes em um sistema de custos ou contábil. Por isso os arquivos de dados devem ser copiados antes do processamento de qualquer registro de transação. Isto pode ser feito pelo dBASE II por meio do comando COPY ou saindo do dBASE II (pelo comando QUIT) e usando um utilitário para cópia de arquivos do seu sistema operacional.

Por segurança, para prevenir erros no lançamento, use o comando SET ESCAPE OFF antes de iniciar os lançamentos e SET ESCAPE ON ao final dos lançamentos. Isto previnirá interrupções acidentais ao processo de lançamento. Se ESCAPE está colocado em OFF não é possível ao usuário interromper ou parar o processo de lançamento a partir do teclado. Você deverá enviar mensagens ao usuário informando que o processo não deve ser interrompido e o que ocorrerá, caso o seja. Avise-o, também, que o sistema está efetuando o ciclo de lançamentos.

Se você tem espaço para manter uma cópia dos arquivos de dados em disco, você poderá escrever uma rotina de recuperação. Isto permitirá que qualquer pessoa volte o sistema à posição anterior aos lançamentos, se necessário. Isto deverá ser rápido e sem chances de interrupção ou

o usuário estará realmente em apuros. A maneira mais simples de recuperar o arquivo é eliminá-lo e redenominar a cópia como:

```
DELETE FILE mestre.dbf
RENAME mestre.cop TO mestre.dbf
```

Isto se processará instantaneamente, mas, apenas funcionará se a cópia estiver no mesmo disco que o arquivo mestre. Se você utiliza discos rígidos, faça cópias em disco rígido e em disco flexível e inclua rotinas de recuperação para ambos.

FECHAMENTO

O arquivo de transações pode ser atualizado de hora em hora, diariamente, semanalmente ou quando necessário para tornar a acumulação significativa. O lançamento deve ser efetuado a um intervalo periódico pré-definido. Uma vez por dia, por semana ou por mês. Para databases contábeis, o processo final, ou fechamento, é efetuado todas as vezes que desejamos relatar ou sumarizar os arquivos de dados atuais. O processo de fechamento limpa os totais parciais, os registros de detalhe e salva apenas os totais atuais em um arquivo separado para relatórios de sumários. Isto é feito normalmente, a cada mês, com um programa que será executado após todos os relatórios serem impressos e os dados aceitos como corretos.

No sistema de custos, por exemplo, as transações sobre tempo devem ser lançadas em um arquivo de detalhe que é essencialmente uma lista de todas as transações do mês. Ao final do mês este arquivo estará muito grande e as informações de detalhe não serão mais necessárias. Os totais anuais parciais são transferidos para um arquivo sumário e o arquivo de detalhes é limpo e compactado.

Como no lançamento, o teclado deverá estar travado durante o fechamento por meio do ESCAPE OFF. Assegure-se que ESCAPE será colocado em ON novamente ao término do fechamento. Envie mensagens ao usuário durante o fechamento indicando onde o sistema está dentro do ciclo de fechamento.

QUANDO NÃO UTILIZAR O PROCESSAMENTO BASEADO EM TRANSAÇÕES

O principal problema com o processamento baseado em transações é que os arquivos principais não refletem a realidade. Se as transações são lançadas diariamente ao final do dia, os arquivos só estarão exatos imediatamente após o lançamento. Por exemplo, se a Oficina do Fred está alterando a quantidade disponível de peças no arquivo de transações e alguém na loja examinar o arquivo mestre do estoque, a quantidade não refletirá as peças que foram adicionadas ou retiradas recentemente.

Para muitas empresas a diferença de um dia na entrada de dados não é um problema sério. A Oficina do Fred imprime o relatório de seu estoque após o lançamento diário que é utilizado durante o dia para referências. Uma outra alternativa é criar um arquivo para apresentação que é utilizado em pesquisa sobre o database durante o dia. O arquivo de apresentação é atualizado automaticamente ao serem nele adicionadas entradas. Ao final do dia, o arquivo mestre do estoque é atualizado e o arquivo de apresentação é completamente reconstruído a partir do arquivo mestre.

DEZ

Criando Relatórios com o dBASE II

O dBASE II dá ao usuário quatro métodos para gerar relatórios. O usuário pode usar qualquer um deles ou uma combinação deles na criação de relatórios. Os quatro métodos são o comando LIST, o gerador de relatórios, o controle de um programa e os relatórios formatados.

O COMANDO LIST

Os comandos LIST e DISPLAY são úteis para se examinar um arquivo de dados. Os dois comandos mostrarão variáveis, valores de campos e expressões como:

```
LIST nopeca,desc,disp,custo,custo*disp
```

Se o comando SET PRINT ON for utilizado previamente, a saída será direcionada para a impressora. O comando LIST pode ser seletivo. A sintaxe do comando LIST é

```
LIST [<extensão>] [FOR <expressão>] [<lista de expressões>] [OFF] [WHILE <expressão>]
                                                    [FIELDS <lista>]
```

Por exemplo:

```
LIST nopeca,disp FOR disp=0
```

(A extensão padrão é ALL.)

Se a opção OFF é omitida, o número de registro para cada registro é impresso no início de cada linha. Se a opção OFF for incluída, o número do registro não será impresso.

O comando DISPLAY é similar, exceto que o dBASE II fará uma pausa a cada 15 registros e perguntará se você deseja continuar. Com o comando DISPLAY, contudo, a extensão padrão não é ALL.

Tanto o comando DISPLAY como o comando LIST podem ser usados para apresentar a estrutura de um arquivo, uma lista de variáveis ou arquivos em disco (veja Capítulos 3 e 12).

O GERADOR DE RELATÓRIOS

O segundo método simples de gerar relatórios é utilizar o gerador de relatórios do dBASE II. Ele será muito prático se o relatório for extraído de um único arquivo e com pouco controle lógico. O relatório pode ser extraído de qualquer parte do arquivo baseado em qualquer critério definido pelo usuário, e as colunas do relatório podem conter campos variáveis, memórias variáveis, conjunto de textos ou expressões aritméticas. O relatório é gerado com um simples comando e utiliza uma definição de formato (arquivo FRM) que pode ter sido preparado anteriormente ou na hora em que o relatório é solicitado. O relatório pode ser direcionado para a console ou para a impressora.

A sintaxe do comando REPORT é

```
REPORT [FORM <nome do arquivo>] [<extensão>] [TO PRINT] [FOR <expressão>]
                                [WHILE <expressão>]
```

NOTA: A extensão WHILE é disponível apenas na versão 2.4 e posteriores.

O comando requer um arquivo de formato para definir o relatório. O arquivo de dados a ser relatado deve ter sido selecionado anteriormente com o comando USE antes de você entrar com o comando REPORT:

```
USE mestre
REPORT FORM temp TO PRINT
```

Uma vez que a cláusula FORM é opcional, se você omiti-la o sistema solicitará que você entre com o nome do arquivo. Se o arquivo já existir, o dBASE II o utilizará como o formato para criação do relatório. Se o arquivo não existir, você responderá a algumas questões necessárias para a criação do formato e posterior armazenamento em um arquivo. A seguir, um exemplo de geração de um relatório para a Oficina do Fred.

```
ENTER OPTIONS, M=LEFT MARGIN, L=LINES/PAGE, W=PAGE WIDTH
PAGE HEADING? (Y/N) Y
ENTER PAGE HEADING: Listagem do Estoque
DOUBLE SPACE REPORT?  (Y/N)  N
ARE TOTALS REQUIRED?  (Y/N)  Y
SUBTOTAIS IN REPORT?  (Y/N)  N
COL      WIDHT,CONTENTS
001      7,NOPECA
ENTER HEADING: No. peca
002      32,DESC
ENTER HEADING: Descricao
003      6,DISP
ENTER HEADING: Disp
ARE TOTALS REQUIRED?  (Y/N)  N
004      8,CUSTO
ENTER HEADING: Custo
ARE TOTALS REQUIRED?  (Y/N)  N
005      10,CUSTO*DISP
ENTER HEADING: Custo total
ARE TOTALS REQUIRED?  (Y/N)  Y
006

PAGE NO. 00001
```

No. PECA	DESCRICAO	DISP	CUSTO	CUSTO TOTAL
0001	CABO DA BATERIA	4	1.98	7.92
0002	RELE	0	8.50	0.00
0003	CORREIA DO VENTILADOR	2	6.95	13.90
0004	REGULADOR DE VOLTAGEM	2	11.50	23.00
0005	JUNTA UNIVERSAL	1	44.95	44.95
0006	CONJ. PINO DO FREIO	6	1.50	9.00
0007	GUARNICAO DO FREIO	0	15.00	0.00
0008	MONTAGEM DO SILENCIOSO TRASEIRO	1	15.00	15.00
0009	LANTERNA LATERAL (E)	2	4.95	9.00
0010	LANTERNA LATERAL (D)	2	4.95	9.00
TOTAL				133.57

Estas perguntas são feitas somente na primeira vez que o relatório é solicitado. Todas as chamadas subseqüentes para este relatório utilizarão o formato previamente criado sem nenhuma solicitação ao operador.

Você pode definir o tamanho da margem esquerda, as linhas por página, o tamanho do cabeçalho do relatório ou aceitar os valores pré-definidos (8, 57 e 80, respectivamente) teclando RETURN. Se deseja-se um cabeçalho para a página, ele poderá ser colocado e um espacejamento simples ou duplo poderá ser selecionado. Cabeçalhos em múltiplas linhas poderão ser feitos pelo uso de ponto-e-vírgulas na definição para indicar a quebra da linha. Então você será solicitado a responder se deseja totais. Se você responder Y, poderá obter totais em qualquer coluna numérica. Por exemplo, você pode definir uma coluna de custo total (a quantidade disponível multiplicada pelo custo). O total desta coluna lhe dirá o valor total do seu estoque.

Então, você responderá se deseja subtotais. Se você responder N, o gerador de relatórios saltará as perguntas sobre a definição de subtotais. Se você responder Y, três questões serão solicitadas. A primeira é sobre qual campo (você obterá o valor do subtotal todas as vezes que o valor do campo for modificado). A segunda é se você deseja todas as informações detalhadas ou apenas um sumário de subtotais. Se o Fred adicionasse um campo no seu estoque para indicar o tipo de

carro (Volvo, Chevrolet etc.) ele poderia obter subtotais indicando o valor do estoque para cada tipo de carro. A terceira questão é se você deseja um salto de página após cada subtotal impresso.

A seguir as colunas são definidas. Para cada coluna entre o tamanho da coluna, o conteúdo (nome do campo, nome da variável, conjunto de caracteres ou expressão) e o título da coluna. Através do controle do comprimento da coluna você pode colocar informações em qualquer parte de uma linha horizontal. Como no cabeçalho, ponto-e-vírgulas são utilizados como indicadores de quebra para os cabeçalhos das colunas ou conjuntos de textos. Você também poderá ajustar à esquerda o título de uma coluna ao preceder o título da coluna com um "<" e ajustá-lo à direita precedendo-o com ">". No exemplo da Figura 10-1, uma coluna de custo total foi criada através de uma expressão aritmética com dois campos. Esta coluna também é totalizada dando o valor total do estoque neste relatório.

Normalmente o tamanho do campo deve ser um caractere maior que o tamanho da variável. Variáveis numéricas ou de caracteres podem ser utilizadas sem a conversão necessária e até 24 colunas por linha poderão ser definidas. Após todas as colunas estarem definidas, tecle o retorno de carro ao ser solicitado o sinal de pronto para a próxima coluna. Isto iniciará a apresentação do relatório ou, se a opção TO PRINT foi utilizada, o relatório será enviado para a impressora.

Várias outras opções estão, também, disponíveis ao usuário:

PAGE NO. 00001
05/10/83

LISTAGEM DO ESTOQUE

Nº PEÇA	DESCRIÇÃO	DISP	CUSTO	CUSTO TOTAL
0001	CABO DA BATERIA	4	1.98	7.92
0002	RELÉ	0	8.50	0.00
0003	CORREIA DO VENTILADOR	2	6.95	13.90
0004	REGULADOR DE VOLTAGEM	0	12.50	0.00
0005	JUNTA UNIVERSAL	1	44.95	44.95
0006	CONJ. PINO DO FREIO	2	3.50	7.00
0007	GUARNIÇÃO DO FREIO	0	15.00	0.00
0008	MONTAGEM DO SILENCIOSO	1	15.00	15.00
0009	LANTERNA LATERAL (E)	2	4.95	9.90
0010	LANTERNA LATERAL (D)	2	4.95	9.90
**TOTAL				108.57

Figura 10-1. Um relatório do estoque utilizando o gerador de relatórios

1. A opção <extensão> pode ser utilizada para limitar o relatório a uma dada parte do arquivo. Se for omitida, o arquivo inteiro será impresso. Por exemplo:

   ```
   REPORT FORM temp NEXT 10
   ```


2. A opção FOR<expressão> pode ser utilizada para selecionar registros a serem impressos. Por exemplo:

   ```
   REPORT FORM temp FOR disp=0
   ```

3. A opção PLAIN pode ser usada para omitir a numeração das páginas e a data que são normalmente impressas pelo gerador de relatórios. Isto será útil se o relatório for editado posteriormente com um processador de textos.
4. A opção WHILE<expressão> pode ser utilizada para especificar uma condição de término para a geração do relatório.
5. O formato do relatório é salvo como .FRM. Este arquivo pode ser lido ou editado com um processador de textos caso se deseje mudanças.
6. O relatório pode ser salvo em um arquivo se a chave SET ALTERNATE for utilizada. Por exemplo:

   ```
   SET ALTERNATE TO arqsum.txt
   SET ALTERNATE ON
   REPORT FORM temp TO PRINT
   SET ALTERNATE OFF
   ```

7. O cabeçário pode ser definido ao tempo de impressão pelo comando SET. Por exemplo:

   ```
   SET HEADING TO LISTA DE EXCECAO
   REPORT FORM temp TO PRINT
   ```

   Note que não é necessário o uso de apóstrofos com um texto.
8. A data do relatório pode ser definida ao tempo de impressão se você desejar uma data diferente de data do sistema. Por exemplo:

   ```
   SET DATE TO 01/30/83
   ```

9. Normalmente o gerador de relatórios saltará uma página antes de iniciar a impressão do relatório. Para evitar isto,

   ```
   SET EJECT OFF
   ```

10. Normalmente ao ser impresso o relatório também é apresentado na console. Para evitar isto entre

    ```
    SET CONSOLE OFF
    ```

    antes de enviar o comando de impressão. Após a impressão entre

    ```
    SET CONSOLE ON
    ```

    para voltar a console ao estado ativo.

O gerador de relatórios não altera nem modifica nenhum arquivo de dados.

RELATÓRIOS SOB O CONTROLE DE UM PROGRAMA

Muitas vezes não é possível criar o relatório desejado com o gerador de relatórios. Se o relatório envolve mais de um arquivo, uma complexa seleção de registros, ou formatação especial, o gerador de relatório não poderá ser utilizado. Se o gerador de relatórios não pode ser usado,

será necessário escrever algum tipo de programa sob o dBASE II para criar o relatório. Se o programa é necessário, você tem duas alternativas, o controle por programa e o controle por formato.

Os comandos "?" e "??", sob controle por programa, são utilizados para formatar e imprimir uma linha por vez. A saída é enviada para a impressora com o comando SET PRINT ON e a impressora é liberada ao final do relatório com o comando SET PRINT OFF. Se o comando SET CONSOLE OFF for utilizado, o relatório irá apenas para a impressora e não será visto na tela.

O controle por formato utiliza o operador "@" para direcionar linhas impressas para qualquer posição da página impressa. Imprimir com controle por formato dá um maior controle sobre a colocação da informação no relatório. A saída formatada (saída utilizando o comando "@") é dirigida para a impressora com o comando SET FORMAT TO PRINT. O controle de formato é devolvido para a tela com o comando SET FORMAT TO SCREEN. A apresentação na console é automaticamente desativada se o controle de formato é dirigido para a impressora.

Ambos os tipos de controle de impressão podem ser utilizados no mesmo relatório se você desejar. Formulários especiais e relatórios formatados devem utilizar o controle por formato. A maioria dos relatórios podem ser produzidos satisfatoriamente com controle por programa.

A seguir um exemplo de um programa para o sistema de custos utilizando controle por programa.

```
* sistema de custos — imprime relatorio parcial
* por carl townsend
* 12/31/82
* parcial
SET TALK OFF
ERASE
RESTORE FROM TITULO ADDITIVE
@ 1,1 SAY '********** ' +mtit+ ' **********'
@ 2,1 SAY 'RELATORIO PARCIAL'
@ 2,60 SAY DATE ( )
@ 15,1 SAY 'INDEXANDO . . . . . . . '
USE b:sumario
INDEX ON tipo+nome TO temp
@ 15,1 SAY 'CALCULANDO AS SOMAS . . . . '
SUM taxa,TOTAL,thrs TO ttaxa,ttotal,tthrs FOR tipo= 'C' .AND. venda= 'T'
GO TOP
@ 15,1 SAY 'IMPRIMINDO . . . . . . . . . . '
* encontra o primeiro cliente de vendas
STORE 'CT' TO x
FIND &x
STORE 0.00 TO xxtaxa,xxtotal
STORE 0 TO pagctr
STORE 90 TO linctr
STORE t TO xmarc
SET PRINT ON
SET CONSOLE OFF
        DO WHILE .NOT. EOF.AND. tipo= 'C' .AND. xmarc .AND. venda= 'T'
```



```
IF #=0
        @ 23,1 SAY 'NAO EXISTEM CLIENTES DE VENDAS NO ARQUIVO;
           SUMARIO'
        STORE 1 TO xx
        @ 23,60 SAY chr(7)
        DO WHITE xx<35
                STORE xx+1 TO xx
        ENDDO WHILE xx<35
        STORE f TO xmarc
        LOOP
ENDIF #=0
IF *
        SKIP
        LOOP
ENDIF *
If linctr>60
        EJECT
        STORE pagctr+1 TO pagctr
        ? 'PAG. ' +STR (pagctr,3) + '
        ?? 'RELATORIO PARCIAL'
        ?? '                          ' +DATE( )
        ?
        ? 'CLIENTE                TAXA           %'
        ?? '              CUSTO            %              DIFERENCA'
        ? '---------           ------                          -'
        ?? '            -------          -          ------------'
        ?
        STORE 5 TO linctr
ENDIF linctr>60
STORE (taxa/ttaxa)*100 TO q
STORE STR(q,8,4) TO x
STORE $(x,LEN(x)-1,2) TO y
IF val(y)>49
        STORE STR(q+.01,6,2) TO xtaxa
ELSE
        STORE STR(q,6,2) TO xtaxa
ENDIF val(y)>49
STORE val(xtaxa)+xxtaxa TO xxtaxa
? '     ' +nome+ '              ' +STR(taxa,10,2)+ '
?? xtaxa+ '        ' +STR(total,10,2)+ '   '
STORE (total/ttotal) * 100 TO q
STORE STR(q,8,4) TO x
STORE $(x,LEN(x)-1,2) TO y
IF VAL(y)>49
        STORE STR(q+.01,6,2) TO xtotal
ELSE
```

```
            STORE STR(q,6,2) TO xtotal
        ENDIF val(y)>49
        ?? xtotal+'          '+STR(taxa-total,10,2)

        STORE VAL(xtotal)+xxtotal TO xxtotal
        SKIP
ENDDO WHILE .NOT. EOF .AND. tipo='C' .AND. XMARC.AND. VENDA='T'
?
? 'TOTAIS FINAIS     '+str(ttaxa,10,2)+'          '+STR(xxtaxa,6,2)
??'          '+STR(ttotal,8,2)+'   '+STR(xxtotal,8,2)+'          '
?? STR(ttaxa-ttotal,10,2)
?
STORE linctr+3 TO linctr
* encontra o primeiro cliente que nao seja vendas
STORE 'CF' TO x
FIND &x
STORE t TO xmarc
DO WHILE .NOT. EOF .AND. tipo='C' .AND. xmarc .AND. venda='F'
        IF #=0
                @ 23,1 SAY 'NAO EXISTEM CLIENTES NAO-VENDAS NO ARQUIVO'
                STORE 1 TO xx
                DO WHILE xx<35
                        STORE xx+1 TO xx
                        @ 23,60 SAY chr(7)
                ENDDO WHILE xx<35
                STORE f TO xmarc
                LOOP
        ENDIF #=0
        IF *
                SKIP
                LOOP
        ENDIF *
        IF linctr>60
                EJECT
                STORE pagctr+1 TO pagctr
                ? '          PAG. '+STR(pagctr,3)+'
                ?? 'RELATORIO PARCIAL'
                ??'                    '+DATE( )
                ?
                ? '     CLIENTE                TAXA          %'
                ??'                CUSTO        %          DIFERENCA'
                ? '     ---------              ------        -'
                ??'               -------       -          ----------'
                ?
                STORE 5 TO linctr
```



```
        ENDIF linctr>60
        ?'          '+nome+'          '+STR(taxa,10,2)+'          '
        ??'          '+STR(TOTAL,10,2)+'          '
        ??'          '+STR(taxa-total,10,2)
        SKIP
ENDDO WHILE .NOT. EOF .AND. tipo='C' .AND. XMARC.AND. VENDA='F'
SET PRINT OFF
SET CONSOLE ON
```

O procedimento geral é o seguinte:

1. Indexar o arquivo na ordem desejada antes de iniciar.
2. Inicializar o contador de páginas com 0 e o contador de linhas com um valor alto. Coloque os acumuladores de totais em 0:

```
STORE 1 TO pagctr
STORE 90 TO linctr
STORE 0.00 TO extcst
```

3. Se o relatório destina-se a impressora, remeta-o para a impressora e desligue-o da console.

```
SET PRINT ON
SET CONSOLE OFF
```

4. Abra os arquivos e inicie a procura dos registros com um loop DO. Use o comando SKIP para evitar os registros eliminados.
5. Antes de imprimir uma linha, verifique se o contador de linhas não "estourou". Se necessário salte uma página e imprima o cabeçalho. Ajuste o contador de páginas e zere o contador de linhas. Por exemplo,

```
IF linctr>60
    EJECT
    ? "ISTO E O CABECALHO"
    ?
    STORE 2 TO linctr
    STORE pagctr+1 TO pagctr
ENDIF
```

6. Imprima as linhas com os comandos "?" e "??". Acumule os totais.

```
? no+'  '+desc+'  '+STR(custo,6,1)
?? '  '+STR(disp,4)+'  '+STR(disp*custo,10,1)
STORE totcusto+(disp*custo) TO totcust
```

7. Avance no arquivo com o comando SKIP.
8. Ao terminar, imprima os totais.
9. Durante a impressão informe ao usuário que a impressão está acontecendo.
10. Dirija a saída para a console e desligue-a da impressora.

Note que as mensagens de erro são construídas e enviadas para a tela como mensagens *formatadas*. O controle por formatos permanece dirigido para a tela. Apenas as saídas não forma-

tadas dos operadores "?" e "??" são enviadas para a impressora. A console é "desligada" e não mostra o relatório – apenas as mensagens de erro.

Você deve notar que o dBASE II *trunca* valores e não os arredonda. Se você necessita calcular porcentagens, você precisará de uma rotina de arredondamento, caso contrário as porcentagens em seus relatórios não totalizarão 100.

Observe as vantagens do controle por programa sobre o gerador de relatórios. Você pode criar mensagens de erro, arredondar cálculos em vez de tê-los truncados e ter o domínio sobre o formato do relatório. Em relatórios mais elaborados você também usará vários arquivos e os operadores do dBASE II.

FORMATANDO RELATÓRIOS

A quarta maneira de gerar relatórios é conveniente se você precisa formatá-lo para um formulário específico. É prática para imprimir faturas, cheques e etiquetas, dentre uma variedade de formulários especiais.

Uma saída formatada é uma saída na qual a posição da variável é definida. Nos capítulos anteriores nós tivemos exemplos como:

```
@ 2,1 SAY  'ADICAO DE LOTE'
@ 2,60 SAY DATE( )
```

Isto mostrará o texto ou a variável na posição indicada.

Isto poderá ser formatado na impressora pelo USO do comando SET FORMAT TO PRINT como se vê a seguir:

```
SET FORMAT TO PRINT
EJECT
@ 1,1 SAY  'LISTAGEM DO LOTE'
@ 1,60 SAY DATE( )
SET FORMAT TO SCREEN
```

Os comandos de formatação para a *impressora* devem indicar posições da esquerda para a direita e de cima para baixo. O exemplo subseqüente não estará correto:

```
@ 1,60 SAY DATE( )
@ 1,1 SAY  'LISTAGEM DO LOTE'
```

O programa a seguir imprime duas etiquetas de endereçamento por vez. (Para simplificar este exemplo, o programa trabalhará apenas com um arquivo e um número par de etiquetas. Veja se você adivinha o porquê). Dois endereços são carregados por vez em um conjunto virtual. O conjunto, então, é impresso com a posição de cada linha calculada antes da impressão.

```
* imprime duas etiquetas de enderecamento por vez
* por carl townsend
* 1/31/83
```



```
SELECT PRIMARY
SET TALK OFF
* da o deslocamento horizontal da primeira etiqueta
STORE 5 TO horiz
* da o deslocamento vertical da primeira etiqueta
STORE 1 TO v
* da o numero de etiquetas horizontais
STORE 2 TO col
* da a distancia entre as etiquetas
STORE 35 TO dist
USE mala INDEX cep
SET FORMAT TO PRINT
DO WHILE .NOT. EOF
      * Carrega o conjunto
     STORE 1 TO no
     DO WHILE no>=col
          STORE 1 TO ctr
          STORE 'V'+STR(ctr,1) TO var
          STORE TRIM(titulo)+'. '+TRIM(prenome)+' 'nome;
          TO &var
          STORE ctr+1 TO ctr
          STORE 'V'+STR(ctr,1) TO var
          STORE endereco TO &var
          STORE ctr+1 TO ctr
          STORE 'V'+STR(no,1)+STR(CTR,1) TO var
          STORE TRIM(cidade)+', 'estado+' '+cep TO &var
          STORE no+1 TO no
          SKIP
     ENDDO
     STORE 1 TO ctr
     * imprime todas as etiquetas
     DO WHILE ctr<=3
     STORE 1 TO no
     STORE horiz TO h
     DO WHILE no<=col
          STORE 'V'+STR(no,1)+STR(ctr,1) TO var
          @ v,h SAY &var
          STORE h+dist TO h
          STORE no+1 TO no
   ENDDO
   STORE ctr+1 TO ctr
   STORE v+1 TO v
 ENDDO
 STORE v+2 TO v
ENDDO
SET FORMAT TO SCREEN
```

Agora podemos imprimir etiquetas ou formatar qualquer relatório que desejamos. Há uma outra vantagem distinta sobre o controle por programa – podemos calcular, ao tempo de impressão, aonde imprimir qualquer linha ou texto no relatório. Neste exemplo os valores de "v" e "h" utilizados juntamente com o operador "@" não são calculados até o momento que a linha é impressa. O comando TRIM é utilizado para comprimir a linha do nome reduzindo espaço entre o nome e o prenome e também para reduzir os espaços vazios nas linhas da cidade, do estado e do CEP.

Se você precisar apresentar mensagens de erro na console, necessitará mudar o destino dos formatos para a tela, mostrar a mensagem e retornar o destino dos formatos para a impressora.

Tanto com controle por formato como por programa você pode obter o relatório em um arquivo pelo comando SET ALTERNATE ON. Isto é conveniente para a impressão de relatórios longos e para apanhar mensagens de erro. Inicie o programa e estabeleça o nome do arquivo alternado.

```
SET ALTERNATE TO ERRO.TXT
```

Se você detectar um erro, use o comando SET ALTERNATE ON para apanhar a mensagem e dirigi-la para o arquivo alternado. Após isto, emita o comando SET ALTERNATE OFF e volte a gerar o relatório.

ONZE

Geradores de Telas

Com o dBASE II é possível desenhar telas rapidamente para as mais variadas necessidades. Menus, telas especiais e inusitadas poderão ser facilmente criadas. Também é possível comprar um programa utilitário especial que escreverá programas sob o dBASE II que criam telas.

As telas podem ser usadas para saída, para entrada ou para uma combinação das duas. A tela pode ser apresentada ao usuário que apenas preencherá os vazios. Uma fatura pode ser mostrada na tela exatamente como irá se apresentar na impressora. Por meio de um desenho criativo você poderá desenhar menus que tornem o sistema bem simples para o usuário.

COMANDOS DISPLAY E LIST

Como nos relatórios, os comandos DISPLAY e LIST podem ser utilizados para pesquisar rapidamente um arquivo. O comando DISPLAY ALL fará uma pausa após 15 registros e perguntará se você deseja continuar; o comando LIST ALL não fará pausas. Os registros serão apresentados com um espaço entre cada campo e você pode selecionar os campos que deseja apresentar:

```
DISPLAY ALL nopeca,desc
```

Isto mostrará o número dos registros, das peças e as descrições, e parará a cada 15 registros. Se o número do registro não é necessário use a opção OFF:

```
DISPLAY ALL nopeca,desc OFF
```

Se você não deseja o espaço entre os campos use a concatenação de conjuntos:

```
DISPLAY ALL nopeca+desc
```

SAÍDA EM TELAS FORMATADAS

Com telas não formatadas o programa irá apenas dirigir textos para linhas seqüenciais. Em muitas aplicações você pode desejar dirigir textos para determinadas áreas da tela; isto é particularmente importante se desejar colocar uma tela para o usuário preencher.

Assim como nos relatórios formatados, a saída é feita com o comando "@". Linhas e colunas são especificadas para as coordenadas de saída, sendo as linhas especificadas primeiro. O formato geral é

@<coordenadas>[SAY <expressão> [USING<formato>]]

A saída formatada deve ser dirigida para a tela com

```
SET FORMAT TO SCREEN
```

Normalmente, esta é a condição padrão. Este comando não é necessário a menos que você tenha formatado um relatório para a impressora. Vários exemplos deste tipo de programação já foram vistos nos capítulos anteriores.

Variáveis também podem ser usadas como coordenadas e o posicionamento é controlado em tempo de execução com

```
ERASE
STORE 1 TO ctr
@ ctr, 1 SAY 'Sistema de Controle de Estoques'
```

A cláusula USING permite ao usuário formatar variáveis segundo seu controle. Os caracteres de formatação que podem ser usados na cláusula USING são apresentados na Tabela 11-1.

As coordenadas podem ser especificadas com cifrões ($) para indicar uma posição *relativa* a partir do último comando SAY. Por exemplo,

```
@ 10,10 SAY titulo1
@ $,$+2 SAY titulo2
```

colocará o texto de título1 seguido pelo texto de título2 deslocado em duas posições.

A saída poderá ser dirigida para a tela em qualquer ordem de linhas ou colunas. Ao contrário da geração de relatórios (na qual a impressão podia mover-se apenas da esquerda para a direita e de cima para baixo de página), o cursor pode mover-se rapidamente para qualquer posição da tela. Embora o dBASE II permita ao usuário criar a tela em qualquer ordem da linha e coluna, tente manter a ordem da esquerda para a direita e de cima para baixo. Isto tornará mais simples e rápida a correção de programas e permitirá o uso de partes do programa em outro programa no qual a ordem seja importante – como entrada de valores da tela e impressão de relatórios.

Quantas variáveis podem ser mostradas em uma tela? Não existe uma limitação teórica. Por exemplo, você pode apresentar uma tabela bidimensional do arquivo de estoque. Um método consistiria em organizar uma linha de variáveis, apresentar a linha, limpar e recarregar as variáveis para a próxima linha. Alternativamente um arquivo para apresentação pode ser criado no qual cada registro representa uma linha de dados de tela. Uma rápida apresentação de dados pode ser criada

Tabela 11-1. Caracteres de formatação na cláusula USING

# ou 9	O próximo número deve ser apresentado
X	Apresente o próximo caractere
A	Apresente o próximo caractere
$ ou *	Apresente "$" ou "*" ao invés de zeros à esquerda

por uma leitura seqüencial de registros e apresentá-los em cada linha da tela. O arquivo de apresentação pode ser recriado completamente todos os dias e ajustado dinamicamente com o estoque durante o dia; pode também ser usado para uma rápida criação de relatórios. O exemplo de uma parte de um programa para uma tela bidimensional é visto aqui:

```
* mostra os valores atuais
STORE 0 TO V
DO WHILE v<19
     STORE 0 TO ctr
     @ v, 1 SAY chave
     STORE 10 TO h
     DO WHILE ctr<11
          STORE 'N' +str(ctr,2) TO var
          IF $(var,2,1)=' '
               STORE $(var,1,1)+ 'O' +$(var,3,1) TO var
          ENDIF $(var,2,1)=' '
          STORE $var To z
          @ v,h SAY str(z,4)
          STORE ctr+1 To ctr
          STORE h+5 To h
     ENDDO WHILE ctr < 11
     STORE v+2 To v
     SKIP
ENDDO WHILE V<19
```

ENTRADAS PELAS TELAS FORMATADAS

Muitas telas podem ser uma combinação de entradas e saídas. Utilizando o comando GET um valor pode ser carregado a partir de qualquer posição da tela. O formato completo do comando é

@<coordenadas>[SAY<expressão>][GET<variável>[PICTURE<formato>]]

Como explicado anteriormente as coordenadas podem ser constantes ou calculadas dinamicamente. O comando GET deve ser processado da esquerda para a direita e de cima para baixo. Os comandos GET não são processados até que uma instrução READ seja encontrada como

```
STORE '        ' TO mno
@ 5,1 SAY  'ENTRE O NUMERO DA PECA' GET mno
READ
```

A variável deve ser declarada antes de ser usada pelo armazenamento de um valor vazio nesta variável. A instrução READ processa todos os comandos GET ainda não processados. O programa parará e aguardará que o usuário entre com os valores.

As instruções de um programa podem ser usadas para criar qualquer tipo de tela de uma forma dinâmica com o formato da tela sendo determinado em tempo de execução. A tela é desenhada ao serem processadas as instruções de formato do programa até que um comando READ seja encontrado.

O exemplo seguinte é um programa que utiliza uma tabela bidimensional como entrada. Os comandos GET são, cada um, dirigidos para uma posição calculada.

```
* usando gets bidimensionais
STORE 10 TO v
SET COLON OFF
STORE 0 TO no
* Carrega o conjunto de entrada a partir da tela
DO WHILE v<19
      @ v,1 SAY mtam
      STORE 10 TO h
      STORE 0 TO ctr
      DO WHILE ctr<11
            STORE STR(no,3) TO var
            * remove brancos
            IF $(var,1,1)=' '
                  STORE '0'+$(var,2,2) TO var
            ENDIF
            IF $(var,2,1)=' '
                  STORE $(var,1,1)+'0'+$(var,3,1) TO var
            ENDIF
            STORE 'X'+var TO var
            STORE '    ' TO &var
            @ v,h GET &var
            STORE h+5 TO h
            STORE no+1 TO no
            STORE ctr+1 TO ctr
      ENDDO WHILE ctr<11
      STORE v+2 TO v
ENDDO WHILE v<19
READ
SET COLON ON
```


Normalmente, aproximadamente 55 variáveis é o máximo que pode ser carregado a partir de uma tela utilizando-se este tipo de entrada bidimensional. Isto deixa poucas variáveis (9) para o restante do programa.

Também é possível criar-se uma máscara de certos valores a serem apresentados de um determinado arquivo. Esta máscara pode existir como uma coleção de valores lógicos, como campos de um arquivo de um só registro ou como variáveis na memória. Estes valores podem então controlar, em uma rotina de edição, quais variáveis devem ser apresentadas para edição. Todos os valores de um registro a ser editado são movimentados para variáveis. Um contador é colocado em 1 (para controlar o posicionamento vertical de tela) e cada valor de máscara é testado. Se o valor é verdadeiro, o valor para edição é apresentado e o contador incrementado. Ao final, um comando READ é executado e todos os valores gravados no arquivo com o comando REPLACE.

UTILIZANDO TERMINAIS COM CARACTERÍSTICAS ESPECIAIS

Se o seu terminal tem características especiais como imagem inversa e chaves de funções programadas, estas características podem ser utilizadas no dBASE II usando-se a função CHR. Esta função torna possível ao usuário o controle de imagem reversa do vídeo a partir de um programa sob o dBASE II ou definir chaves de função no teclado para entrada de dados.

Por exemplo, no terminal TeleVídeo 950 as chaves de função podem ser programadas com a seguinte seqüência:

```
ESCAPE | <f1> <f2> <texto> Control-Y
```

onde "f1" é o número da chave da função e "f2" é o código para indicar duplicidade. Por exemplo:

```
?? CHR(27)+CHR(124)+CHR(49)+CHR(49)+'AGENCIA'+CHR(25)
```

Esta linha poderia ser uma parte de um menu principal e definiria a chave de função 1. Subseqüentemente a cada vez que f1 for teclada, a palavra "AGÊNCIA" será enviada ao dBASE II como um conjunto de entrada; isto reduz erros de digitação. Similarmente, a execução do próprio menu (DO MENU) pode ser definida e executada a partir de uma chave da função.

Os conjuntos de comandos de um terminal podem ser armazenados em variáveis. Normalmente, poucas variáveis são necessárias. Aqui está um exemplo de salvamento de chaves de meia-intensidade e de vídeo-reverso como variáveis em um terminal TeleVídeo 925:

```
CLEAR
SET TALK OFF
SET INTENSITY OFF
ERASE
SET COLON OFF
* define uma variavel ligando reverse video
STORE CHR(27)+CHR(106) TO ron
* define uma variavel desligando reverse video
STORE CHR(27)+CHR(107) TO roff
* define uma variavel para ligar meia intensidade
```

```
STORE CHR(27)+CHR(41) TO hon
* define uma variavel para desligar meia intensidade
STORE CHR(27)+CHR(49) TO hoff
* mostra a tela enviando os codigos necessarios
? ' ' +hon+ '          ' +hoff
?? '          ' +hon+ '          ' +hoff
? ' ' +hon+ 'Data do cartao de ponto: ' +hoff+' ;
     ' +hon+ '          Dia:'+hoff
?? '          ' +hon+ ' Identificacao do Empregado:' +hoff+ ' ;
                                    ' +hon+ ' ' +hoff
? ' ' +hon+ ' Nome: ' +hoff+ '                    ;
     ' +hon+ ' Departamento:' +hoff
?? '          ' +hon+ 'Periodo de trabalho:' +hoff+ '          ' ;
                                    +hon+ ' ' +hoff
? ' ' +hon+ 'Inicio do periodo:' +hoff+ '
     ' +hon+ 'Termino do periodo:' +hoff
?? '          ' +hon+ 'Duracao do periodo:' +hoff+ '          ' ;
                                    +hon +' ' +hoff
? ' ' +hon
```

A seguir, linhas gerais para utilização de características especiais dos terminais:

- Use códigos de controles quando a tela não for desenhada com uma sucessão de comandos GET; isto significa, normalmente, armazenar formatos separados para telas de entrada e de saída. Os formatos podem sobrepor-se na tela; isto significa que você pode apresentar um formato e então (sem limpar a tela com o comando ERASE) mostrar um segundo formato.
- Para evitar a criação de muitos arquivos de formatos para entradas e saídas você pode usar variáveis no programa chamador.
- Alguns terminais usam um branco quando você coloca algum atributo especial. Certifique-se verificando o manual do seu terminal.
- Não é necessário trabalhar de cima para baixo e de esquerda para a direita na criação de formatos, mas cuidado deve ser tomado quando códigos de controle são utilizados. Um código de controle, uma vez enviado à tela, torna a tela indiferente ao posicionamento do cursor.

Com um pouco de experiência, você verá que pode criar telas extremamente adequadas aos usuários com mensagens de erro brilhantes e formatos de telas complexos.

SALVANDO TELAS EM PROGRAMAS

Freqüentemente se deseja criar uma tela que seja utilizada em diversos programas. Isto pode ser feito de duas maneiras.

A primeira maneira é criar todas as instruções de formatação, incluindo as instruções GET e salvá-las em um arquivo do tipo FMT. Isto poderá ser chamado em um programa com o comando SET FORMAT e será executado no READ subseqüente. Por exemplo,

```
SET FORMAT TO test
READ
```

executará as instruções de formato do arquivo TESTE.FMT e terá qualquer variável pendente criada pelo formato enviada à tela. Se o arquivo de formato é usado apenas para saída, este método ainda pode ser utilizado. O comando READ não obterá nenhum dado na execução.

Alternativamente, o usuário pode definir um arquivo de programa para a formatação e chamá-lo como chamaria um programa a partir do menu. Podem ser salvas duas telas e serem usadas sobrepondo-se uma a outra. Uma pode ser a tela de saída que mostre um formato para ser preenchido.

A segunda maneira pode conter o GET e as instruções para entrada. Você primeiramente formatará uma tela de uso geral que possa ser usada em vários programas ou várias partes em um mesmo programa. Então, sem limpar a tela, uma segunda série de comandos de formatação é executada com as instruções GET e os READ associados.

DOZE

Gerenciamento de Arquivos

Com o dBASE o usuário dispõe de uma grande variedade de comandos para gerenciar arquivos. Listagem de diretórios, eliminação de arquivos, redenominação de arquivos e comandos mais complexos como cópias e classificações podem ser feitos pelo dBASE II.

APRESENTAÇÃO DE DIRETÓRIOS

Os arquivos de dados de uma determinada unidade de disco podem ser apresentados utilizando-se o comando

```
DISPLAY FILES
```

ou

```
DISPLAY FILES ON B
```

O comando DISPLAY também pode ser usado com especificações e com a opção LIKE. As especificações utilizam asteriscos ou pontos de interrogação para indicar igualdade como no comando DIR do CP/M. Um exemplo seria

```
DISPLAY FILES LIKE *.*
```

Isto mostrará todos os arquivos no disco padrão e emitirá uma lista muito parecida com a listagem de diretório que você obtém utilizando o comando DIR fora do dBASE II.

ELIMINANDO E REDENOMINANDO ARQUIVOS

Os arquivos podem ser eliminados com o comando.

```
DELETE FILE mestre
```

No comando DELETE, o tipo de arquivo assumido é "DBF". A unidade de disco pode ser indicada, se desejado, com

```
DELETE FILE b:mestre
```

Se não for especificada nenhuma unidade de disco, a unidade padrão será assumida.

Os arquivos podem ser redenominados com o comando RENAME. Utilizando-se DELETE FILE e RENAME conjuntamente, uma rotina simples de recuperação, a partir de um arquivo anteriormente copiado, poderá ser escrita da seguinte maneira:

```
DELETE FILE mestre
RENAME mestre.bak TO mestre.dbf
```

O COMANDO COPY

O comando COPY é útil para copiar total ou parcialmente um arquivo para outro. Se você efetuar uma cópia para um arquivo que já existe, este será destruído pelo conteúdo proveniente do arquivo copiado pelo comando COPY. O comando COPY é muito rápido (mais do que o programa PIP) e pode ser utilizado para obter-se cópias de segurança. Por exemplo:

```
USE mestre
COPY TO mestre.bak
```

O comando COPY também pode ser usado para copiar apenas campos específicos para um outro arquivo ou para copiar em condições específicas. O formato completo do comando COPY é

```
COPY TO <arquivo>[<extensão] [FIELD <campos>] [FOR<expressão>] [WHILE <expressão>]
```

NOTA: A opção WHILE só é disponível com a versão 2.4 ou posteriores.

Lembre-se que você não pode *adicionar* registros em um arquivo com o comando COPY, uma vez que o conteúdo do original do arquivo de destino (incluindo a sua estrutura) será destruído.

É possível copiar-se apenas a estrutura de um arquivo para um outro. Por exemplo:

```
USE mestre
COPY TO temp STRUCTURE
```

Isto apenas copiará a estrutura do arquivo MESTRE para o arquivo TEMP. Isto é especialmente útil para modificar databases como veremos neste capítulo.

ACRESCENTANDO AO DATABASE

Também é possível adicionar registros a um arquivo sob o dBASE II a partir de um outro arquivo sob o dBASE II. Isto é feito pelo comando APPEND. A forma geral do comando é

```
APPEND FROM <nome do arquivo> [FOR <expressão>]
```

Isto é muito útil para modificar-se a estrutura de um database sem destruir os dados existentes. O comando MODIFY STRUCTURE não pode ser utilizado em um database com dados, uma vez que destruiria todos os dados. Ao invés dele pode ser utilizado o comando APPEND.

```
USE mestre
COPY STRUCTURE TO temp
USE temp
MODIFY STRUCTURE
        (modificações)
APPEND FROM mestre
CLEAR
DELETE FILE mestre
RENAME temp.dbf TO mestre.dbf
```

Isto copiará todas as informações do arquivo MESTRE para um arquivo temporário (TEMP). Após as modificações, todos os campos que ainda existem na nova estrutura são copiados e o arquivo temporário é redenominado. Os novos campos que foram criados permanecerão vazios.

CLASSIFICANDO ARQUIVOS

É rara a necessidade de classificar-se arquivos no dBASE II. Para a impressão de relatórios ou a apresentação de telas, geralmente é bem melhor criar-se um índice do que classificar os registros. Se é necessária a classificação de um arquivo, a forma do comando é

```
SORT ON <campo> TO <nome do arquivo> [ {ASCENDING | DESCENDING }]
```

Ao contrário de indexação, você não pode classificar parte de campos. Se você quiser classificar vários campos, classifique primeiro o de menor campo chave e, progressivamente, sucessivas classificações em direção a chave maior. Como o SORT regrava, o arquivo de dados (o arquivo não classificado) não deve ser o mesmo que o arquivo de destino (o arquivo classificado).

Classificações sob o dBASE II são raramente necessárias e muito demoradas. Se você necessita usar classificação é melhor fazê-la fora do dBASE II e utilizar um programa de classificação comercial como o QSORT da Structured Systems.

ATUALIZANDO UM ARQUIVO

O comando UPDATE permite ao usuário atualizar o arquivo MESTRE a partir de um arquivo secundário que contenha as informações de atualização e de chave. Na versão 2.3 do dBASE II este comando era um tanto limitado. Na versão 2.4 ou posteriores do dBASE II o formato completo do comando tornou-se

```
UPDATE FROM <nome do arquivo> ON <chave> [ADD <campos>] [RANDOM] [REPLACE
        [<campos>]] [<campos> WITH <campo>]]
```

Como exemplo, a Figura 12-1 mostra uma listagem parcial do estoque da Oficina do Fred. A estrutura do arquivo é a seguinte:

```
STRUCTURE FOR FILE: MESTRE.DBF
NUMBER OF RECORDS: 00000
DATA OF LAST UPDATE: 04/29/83
PRIMARY USE DATABASE
FLD     NAME       TYPE   WIDTH     DEC
001     NOPECA     C      006
002     DESC       C      032
003     NOTAS      C      040
004     DISP       N      005
005     CUSTO      N      007       002
006     PRECO      N      008       002
** TOTAL **                00099
```

```
PAGE NO.00001

                    LISTAGEM DO ESTOQUE

NO PECA  DESCRICAO                          DISP   CUSTO   CUSTO TOTAL
0001     CABO DA BATERIA                      4    1.98      7.92
0002     RELE                                 0    8.50      0.00
0003     CORREIA DO VENTILADOR                2    6.95     13.90
0004     REGULADOR DE VOLTAGEM                0   12.50      0.00
0005     JUNTA UNIVERSAL (F)                  1   44.95     44.95
0006     CONJ. PINO DO FREIO                  2    3.50      7.00
0007     GUARNICAO DO FREIO                   0   15.00      0.00
0008     MONTAGEM DO SILENCIOSO TRASEIRO      1   15.00     15.00
0009     LANTERNA LATERAL (E)                 2    4.95      9.90
0010     LANTERNA LATERAL (D)                 2    4.95      9.90
**TOTAL**                                                   108.57
```

Figura 12-1. O arquivo MESTRE antes da atualização

Agora criamos o arquivo de transações (TRANSAC) com a seguinte estrutura:

```
STRUCTURE FOR FILE: B:TRANSAC.DBF.
NUMBER OF RECORDS: 00000
DATE OF LAST UPDATE: 05/13/83
PRIMARY USE DATABASE
FLD     NAME        TYPE    WIDTH     DEC
001     NOPECA      C       006
002     DISP        N       005
003     CUSTO       N       007       002
004     PRECO       N       008       002
**TOTAL**                   00027
```

Adicionamos agora uma mudança na quantidade disponível, no custo e no preço de duas peças com

```
00001    0004    2    11.50    15.29
00002    0006    4     1.50     1.99
```

Referenciando-se à Figura 12-1 veremos tratar-se do regulador de voltagem e do conjunto do pino do freio. Agora podemos atualizar o arquivo MESTRE com

```
USE mestre
UPDATE ON nopeca FROM transac ADD disp REPLACE custo, preco
```

```
PACE NO. 00001
05/13/83

                    LISTAGEM DO ESTOQUE

NO PECA  DESCRICAO                          DISP   CUSTO   CUSTO TOTAL
0001     CABO DA BATERIA                      4    1.98        7.92
0002     RELE                                 0    8.50        0.00
0003     CORREIA DO VENTILADOR                2    6.95       13.90
0004     REGULADOR DE VOLTAGEM                2   11.50       23.00
0005     JUNTA UNIVERSAL                      1   44.95       44.95
0006     CONJ. DO PINO DO FREIO               6    1.50        9.00
0007     GUARNICAO DO FREIO                   0   15.00        0.00
0008     MONTAGEM DO SILENCIOSO TRASEIRO      1   15.00       15.00
0009     LANTERNA LATERAL (E)                 2    4.95        9.90
0010     LANTERNA LATERAL (D)                 2    4.95        9.90
**TOTAL**                                                     133.57
```

Figura 12-2. O arquivo MESTRE após a atualização

O arquivo MESTRE resultante é visto na Figura 12-2. Note que "disp", "custo" e "preço" foram modificados para essas duas peças.

O COMANDO JOIN

O comando JOIN permite ao usuário juntar dois databases para formar um terceiro database baseando-se em critérios específicos. O formato do comando é

```
JOIN TO <nome do arquivo> FOR <expressão> [FIELDS<campos>]
```

Como um exemplo, vamos assumir que nós temos o arquivo FATURA com a seguinte estrutura:

```
STRUCTURE FOR FILE: B:FATURA.DBF
NUMBER OF RECORDS: 00005
DATE OF LAST UPDATE: 05/13/83
PRIMARY USE DATABASE
FLD     NAME          TYPE    WIDTH      DEC
001     noclient      C       006
002     nopeca        C       006
002     no            N       005
**TOTAL**                     00018
```

Nós adicionaremos a ele alguns números de clientes, as peças que compraram e a quantidade de cada peça.

```
CLIENTE  NUMERO  NOPECA   NO
         000010  0004     1
         000012  0007     2
         000015  0003     1
         000022  0003     1
         000030  0004     1
```

Nós podemos agora processar este arquivo com o comando JOIN como se vê:

```
SELECT PRIMARY
USE mestre INDEX mestre
SELECT SECONDARY
USE fatura
JOIN TO temp FOR s.nopeca=p.nopeca FIELD noclient, nopeca, no, custo
```

Isto criará um novo arquivo, o arquivo TEMP, com a seguinte estrutura:

```
STRUCTURE FOR FILE: B:TEMP.DBF
NUMBER OF RECORDS: 00005
DATE OF LAST UPDATE: 05/13/83
PRIMARY USE DATABASE
FLD     NAME          TYPE   WIDTH     DEC
001     noclient        C      006
002     NOPECA          C      006
003     no              N      005
004     CUSTO           N      007      002
**TOTAL**                      00025
```

O conteúdo do arquivo TEMP será o seguinte:

```
00001    0015    000003    1      6.95
00002    0022    000003    1      6.95
00003    0010    000005    1     11.50
00004    0030    000004    1     11.50
00005    0012    000007    2     15.00
```

Agora será possível calcular o valor das mudanças do estoque usando-se o arquivo TEMP

```
USE temp
SUM NO *CUSTO  TO x
```

Neste exemplo, nós utilizamos propositadamente letras minúsculas nos nomes dos campos do arquivo FATURA para mostrar de onde foram derivados os campos do arquivo TEMP. Isto é opcional nas suas aplicações. Note que o comando JOIN usa dois arquivos como entrada para criar um terceiro arquivo.

RECUPERANDO ARQUIVOS DANIFICADOS

Existirão ocasiões em que o seu arquivo favorito poderá desaparecer. Se os programas são desenvolvidos e testados corretamente, você nunca verá que isso aconteça com arquivos de produção. Isto será possível de ocorrer durante o desenvolvimento dos programas, particularmente com comandos diretos. As últimas versões do dBASE II têm muito menos erros do que as versões anteriores que realmente faziam coisas estranhas. Mesmo com as novas versões, o dBASE II poderá, ocasionalmente, testar a sua paciência.

Lembre-se de que o dBASE II é um grande sistema de gerenciamento de databases e programas para microcomputadores. O dBASE II não reside totalmente na memória a um determinado tempo e várias partes do programa vão e vêem do disco conforme o necessário. Para obter todas as características necessárias no dBASE II, com as limitações de memórias impostas (48K), por vezes as rotinas de recuperação de erros não são tão boas quanto deveriam ser. O programador deve assumir a responsabilidade maior na detecção de condições de erro. Por exemplo, fechar os arquivos que não está utilizando, declarar variáveis antes de usá-las, pelo armazenamento de valores vazios nelas, e evitar a utilização de técnicas de programação não testadas. Em qualquer

versão você normalmente encontrará erros nas novas características adicionadas desde a última versão.

Em um arquivo de dados do dBASE II, o número de registros existentes no database é armazenado no início do arquivo em um registro cabeçalho. Este cabeçalho é atualizado toda vez que o arquivo é fechado. Se o sistema terminou de maneira acidental ou algum erro incomum acontecer (como retirar o disco da unidade antes do arquivo ser fechado), o cabeçalho não será atualizado e o arquivo não funcionará corretamente com nenhum dos programas sob dBASE II.

Os índices são razoavelmente fáceis de recuperar sob certas condições. Você sempre pode adicionar pequenas rotinas para reconstruir todos os índices para o usuário. Isto lhe evitará um grande número de chamados se você faz consultoria. Na maioria dos programas que correm durante a noite, inclua uma rotina para reconstrução de índices.

A reconstrução de arquivos de dados é um outro problema. Felizmente, se você fechar os arquivos corretamente provavelmente nunca verá um arquivo de dados danificado. Mesmo com problemas causados por falhas de equipamento, você geralmente encontrará os arquivos intactos com a perda, apenas, dos últimos registros.

Erros nos arquivos de dados podem, todavia, ocorrer durante o desenvolvimento do sistema. Nunca faça o desenvolvimento de sistemas e o trabalho de produção no mesmo sistema ao mesmo tempo. Se um arquivo de dados estragar-se, primeiramente tente copiá-lo para um outro arquivo ou juntá-lo a outro arquivo temporário com a mesma estrutura. Se isto não der certo (apenas alguns registros forem copiados ou mesmo nenhum), você poderá usar o comando GOTO para obter qualquer registro do arquivo corretamente, mas se você tentar listar o arquivo, não verá todos os registros. Se isto for verdade, um programa similar a este poderá copiar o arquivo corretamente (mas vagarosamente) pelo uso do comando GOTO.

```
SET TALK OFF
SELECT PRIMARY
USE SUMARIO
SELECT SECONDARY
USE temp
SELECT PRIMARY
STORE 1 TO ctr
DO WHILE ctr<60
    GOTO ctr
    SELECT SECONDARY
    APPEND BLANK
    REPLACE s.tipo WITH p.tipo, s.nome WITH p.nome, s.total WITH p. total
    REPLACE s.renda WITH p.renda, s.thrs WITH p.thrs, s. desc WITH p.desc
    REPLACE s.vendas WITH p.vendas, s.total2 WITH p.total2
    SELECT PRIMARY
    STORE ctr+1 TO ctr
ENDDO
```

Você deverá definir um loop para o número de registros que estima existirem no arquivo. Se tudo isso falhar, leia o Apêndice E e estude a estrutura de alguns poucos registros do arquivo de dados. Você estará apto a consertá-los e trazê-los à vida com um pouco de magia técnica.

TREZE

Usando o dBASE II com Arquivos Externos e Outros Programas

O dBASE II mantém os dados e as chaves dos arquivos em um formato especial (veja Apêndice E). Estes arquivos têm um registro cabeçalho que descreve a estrutura do arquivo assim como os dados. Todavia, é possível para o usuário ler arquivos criados por outras linguagens, como BASIC ou PASCAL, ou gravar arquivos que serão lidos por outros programas, por meio do dBASE II. Como resultado disto, caso você esteja atualmente utilizando um arquivo de dados gerenciado por um programa BASIC, será um tanto simples escrever um programa sob o dBASE II que leia este arquivo e armazene-o em um arquivo de dados sob o dBASE II.

LENDO ARQUIVOS EXTERNOS

Arquivos que contenham dados que não estejam nos padrões do dBASE II são chamados arquivos externos. Estes arquivos podem ser lidos pelo dBASE II usando-se o comando APPEND e podem ser gravados pelo comando COPY. Os dois comandos têm uma variedade de opções que permitem ao usuário criar vários tipos de outros arquivos.

Imagine que você tenha um arquivo de entrada utilizado por um programa em BASIC, com um formato predeterminado e deseja lê-lo com o dBASE II. Este arquivo deverá ser convertido em um outro na mesma linguagem (BASIC). Os números são convertidos em conjunto de caractetes, os campos podem precisar ser extraídos e alguma formatação especial pode ser necessária. Este arquivo externo é então lido para um arquivo dBASE II (veja Figura 13-1). Após esta conversão, o arquivo será convertido com o comando COPY para a nova estrutura desejada. Como linha geral, tente fazer o maior trabalho de reformatação do arquivo dentro do dBASE II. Em outras palavras, se você planeja expandir ou adicionar um campo, faça-o dentro do dBASE II após a conversão do arquivo. É muito mais fácil do que o usar o BASIC para o mesmo trabalho.

Comece criando um arquivo vazio sob o dBASE, com um formato o mais próximo possível do arquivo que você está utilizando com a linguagem de alto nível. Assegure-se de que os campos estão na mesma ordem e tamanho do que no outro arquivo. Todos os campos devem ser compostos por caracteres para facilitar o trabalho de conversão.

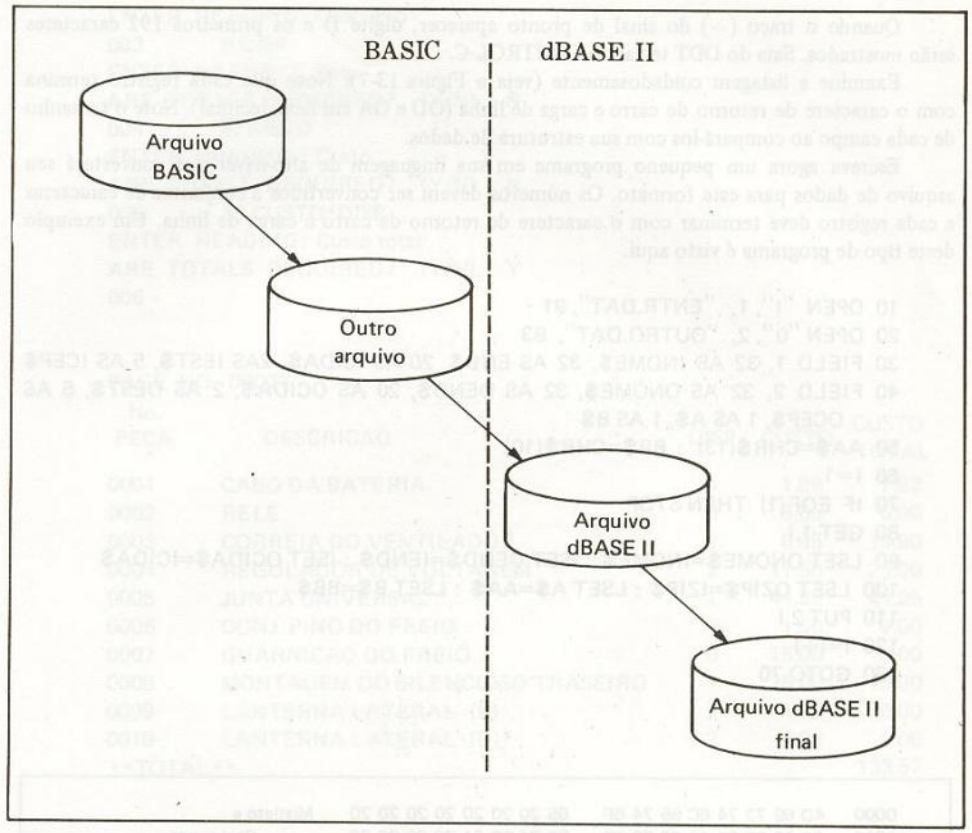


Figura 13-1. Convertendo um arquivo BASIC para dBASE II.

Agora use o comando APPEND para carregar uns poucos registros vazios no arquivo. Use o comando COPY com a opção SDF para copiar estes registros vazios para o arquivo externo ao dBASE II.

```
COPY TO temp.dat SDF
```

A opção SDF diz ao dBASE II para criar o arquivo TEMP.DAT como um arquivo externo ao dBASE II sem nenhum registro cabeçalho. Agora, saia do dBASE II e use algum tipo de utilitário DUMP para listar o arquivo. Por exemplo, se você está sob CP/M, pode usar o programa DDT. Entre com o comando

```
DDT temp.dat
```

Quando o traço (–) do sinal de pronto aparecer, digite D e os primeiros 192 caracteres serão mostrados. Saia do DDT teclando CONTROL-C.

Examine a listagem cuidadosamente (veja a Figura 13-7). Note que cada registro termina com o caractere de retorno de carro e carga de linha (OD e OA em hexadecimal). Note o tamanho de cada campo ao compará-los com sua estrutura de dados.

Escreva agora um pequeno programa em sua linguagem de alto-nível que converterá seu arquivo de dados para este formato. Os números devem ser convertidos a conjuntos de caracteres e cada registro deve terminar com o caractere de retorno de carro e carga de linha. Um exemplo deste tipo de programa é visto aqui.

```
10 OPEN "I", 1, "ENTR.DAT", 91
20 OPEN "O", 2, "OUTRO.DAT", 93
30 FIELD 1, 32 AS INOME$, 32 AS END$, 20 AS ICIDA$, 2AS IEST$, 5 AS ICEP$
40 FIELD 2, 32 AS ONOME$, 32 AS OEND$, 20 AS OCIDA$, 2 AS OEST$, 5 AS
     OCEP$, 1 AS A$, 1 AS B$
50 AA$=CHR$(13) : BB$=CHR$(10)
60 I=1
70 IF EOF(1) THEN STOP
80 GET 1,I
90 LSET ONOME$=INOME$ : ISET OEND$=IEND$ : ISET OCIDA$=ICIDA$
100 LSET OZIP$=IZIP$ : LSET A$=AA$ : LSET B$=BB$
110 PUT 2,I
120 I=I+1
130 GOTO 70
```

```
0000   4D 69 73 74 6C 65 74 6F   65 20 20 20 20 20 20 20   Mistleto e
0010   20 20 20 20 43 68 72 69   73 74 6D 61 73 20 20 20       Chri stmas
0020   20 20 20 20 20 20 20 20   32 33 34 20 46 6F 78 20             234 Fox
0030   52 6F 61 64 20 20 20 20   20 20 20 20 20 20 20 20   Road
0040   20 20 20 20 20 20 20 20   50 6F 72 74 6C 61 6E 64             Portland
0050   20 20 20 20 20 20 20 20   20 20 20 20 4F 52 39 37                 OR97
0060   32 31 32 0D0A 53 75 6E   73 68 69 6E 65 20 20 20   212 . . Sun shine
0070   20 20 20 20 20 20 20 20   20 4D 61 72 79 20 20 20         Mary
0080   20 20 20 20 20 20 20 20   20 20 20 20 20 35 32 31           521
0090   20 4E 2E 20 43 65 64 61   72 20 4C 61 6E 65 20 20   N. Ceda r lane
00A0   20 20 20 20 20 20 20 20   20 20 20 20 20 53 68 61                 Sha
00B0   72 6F 6E 20 20 20 20 20   20 20 20 20 20 20 20 20   ron
00C0   20 50 41 31 36 31 34 36   0D0A 42 65 73 74 67 75    PA16146 . . Bestgu
00D0   65 73 73 20 20 20 20 20   20 20 20 20 20 20 4D 61   ess             Ma
00E0   72 6B 20 20 20 20 20 20   20 20 20 20 20 20 20 20   rk
00F0   20 20 31 32 20 4C 6F 6E   67 77 6F 6F 64 20 20 20    12 Lon gwood
0100   20 20 20 20 20 20 20 20   20 20 20 20 20 20 20 20
0110   20 20 50 69 74 74 73 62   75 72 67 68 20 20 20 20   Pittsb urgh
0120   20 20 20 20 20 20 50 41   31 35 32 32 32 0D0A 1A         PA 15222 . : .
```

Figura 13-2. Listagem de um arquivo externo criada pelo dBASE II.

Se o arquivo a ser convertido é um arquivo seqüencial, ele primeiro deverá ser convertido a acesso direto. A conversão pode ser feita de um outro arquivo seqüencial se todos os tamanhos de registros e campos são os mesmos, mas este exemplo usa um arquivo de acesso direto.

Execute o programa com seus dados atuais para criar, formatando o arquivo externo para o dBASE II. Use, então, o seu programa para listar arquivos e compare alguns registros da listagem anterior. Verifique os campos e o tamanho dos registros para estar certo de que estão corretos. Assegure-se de que qualquer campo numérico foi convertido corretamente e com o tamanho certo. Modifique o programa de conversão se necessário e reexecute-o até que as duas listagens sejam iguais.

Uma vez que você converteu o arquivo, elimine os registros vazios do dBASE II do arquivo e compacte-o. Agora leia este arquivo externo para o arquivo dBASE com o comando APPEND.

```
APPEND FROM temp.dat SDF
```

Use o comando EDIT para verificar alguns registros (como EDIT 3, EDIT 4 etc.) e liste o arquivo. Assegure-se de que campos e registros foram convertidos corretamente. Se a conversão não foi correta, reexamine as listagens anteriores para encontrar onde os campos não correspondem.

Uma vez que o seu novo arquivo dBASE II está correto, você pode usar os comandos do dBASE II para convertê-lo a qualquer formato que deseje. Campos poderão ser acrescentados, eliminados ou ter seus tamanhos alterados utilizando-se os métodos do Capítulo 12.

Se você está convertendo um sistema inteiro, a idéia seria escrever uma coleção de programas que convertam cada um de seus arquivos. Isto poderá ser montado e executado como um lote para o dBASE II. Utilizando este método você pode "capturar" cada um dos outros arquivos de dados para o dBASE II e então testar paralelamente seu novo sistema dBASE II contra o sistema atual em BASIC. Este método torna simples o teste paralelo do sistema durante o desenvolvimento com um impacto mínimo sobre a produção.

GRAVANDO ARQUIVOS EXTERNOS

O reverso do processamento pode ser usado para criar arquivos externos que possam ser lidos por programas fora do dBASE II. Usando este método, você pode criar um arquivo de dados que possam ser lidos por programas de processamento de textos ou de planilhas. Para criar um arquivo externo, use o comando COPY com a opção SDF:

```
COPY TO temp.dat SDF
```

Isto criará um arquivo normal de acesso direto com registros de tamanho fixo que podem ser lidos por qualquer programa fora do dBASE II. Cada registro terá campos com os tamanhos definidos na estrutura dos dados. Cada registro terminará com os caracteres de retorno de carro e carga de linha.

Após ser gravado, o arquivo externo poderá ainda necessitar ser processado com um programa de conversão em linguagem de alto nível que elimine os caracteres de retorno de carro e carga de linha ou que converte o conjunto de caracteres para campos numéricos. A seguir o exemplo de um programa BASIC que lê o arquivo externo criado pelo dBASE II:

```
10  I$="B:TEMP1.DAT"
20  OPEN "R", 1, I$, 101
30  FIELD 1, 20 AS A$, 20 AS B$, 32 AS C$, 20 AS D$, 2 AS E$,
        5 AS F$, 2 AS DUM$
40  FOR I=1 TO 3
50  GET 1,I
60  PRINT "**************************************************"
70  PRINT A$
80  PRINT B$
90  PRINT C$
100 PRINT D$
110 PRINT E$
120 PRINT F$
130 PRINT
140 PRINT "**************************************************"
150 NEXT I
```

USANDO A OPÇÃO DELIMITED

Em certos casos você pode querer ler ou gravar um arquivo externo no qual os itens de dados são separados por vírgulas e podem ou não estar delimitados por aspas. Os campos podem variar de tamanho (ao contrário dos registros fixos da última sessão) e os registros são lidos e gravados seqüencialmente. Este é um formato muito comum para programas escritos em BASIC.

Este tipo de arquivo pode ser lido pelo dBASE II com o comando APPEND ou gravados com o comando COPY. Em ambos os casos, contudo, você deve usar a opção DELIMITED. Um exemplo seria

```
COPY TO teste DELIMITED
```

ou

```
APPEND FROM teste DELIMITED
```

O arquivo externo conterá os itens de dados em ordem seqüencial com os itens entre apóstrofos e separados por vírgulas.

O comando APPEND aceitará arquivos de dados usando apóstrofos ou aspas. O comando COPY gravará apóstrofos. Você pode usar o termo WITH no comando COPY para forçá-lo a usar aspas:

```
COPY TO teste DELIMITED WITH"
```

Se você quer economizar espaço no arquivo, as aspas podem ser eliminadas se uma vírgula (,) é usada com WITH

```
COPY TO teste DELIMITED WITH,
```


Isto também eliminará os brancos finais no campo de dados, o que significa que você não pode usar vírgulas nos campos de dados, uma vez que ela será apenas um delimitador; ou seja, a vírgula delimitando ou separando campos de dados será o sinal do início de um novo campo de dados.

O formato completo do comando COPY é

```
COPY TO <arquivo>[<extensão>] [FIELD<campos>] [FOR<expressão>]
    SDF[WHILE<expressão>] [DELIMITED[WITH<delimitador>]]
```

USANDO O dBASE II COM PROCESSADORES DE TEXTO

O dBASE II pode ser usado para criar e manter arquivos de endereços. Estes arquivos podem ser utilizados com processadores de textos para imprimir cartas, envelopes ou etiquetas.

O WordStar com a opção Mailmerge, por exemplo, trabalhará eficientemente com o dBASE II. Nós podemos definir um arquivo de nomes e endereços. Campos adicionais podem ser definidos se desejarmos e movermos apenas campos selecionados para o WordStar. O comando COPY é então utilizado para mover os campos selecionados para o arquivo externo usando a cláusula DELIMITER.

Por exemplo:

```
COPY TO temp3.dat SDF DELIMITED
```

copiará os campos sem brancos nos finais e os conjuntos de caracteres não serão colocados entre apóstrofos. Este arquivo pode ser usado diretamente no MailMerge. Um exemplo de carta pré-

```
.op
.df b:temp3.dat
.rv nome, prenome,end,cid,est,CEP
                                    15 de fevereiro de 1983

&prenome& &nome&
&end&
&cid& &est& &CEP&

Caro Sr. &lastname&:

Grato por seu pedido. Ele está sendo despachado hoje.

                                    Atenciosamente,

                                    Joao Vendetudo
```

Figura 13-3. Entradas para o arquivo de dados na carta pré-formatada

15 de fevereiro de 1983

Marcos Compratudo
Rua Cruzeiro, 12
Cachaprego, BA 20202

Caro Sr. Compratudo:

Grato por seu pedido. Ele está sendo despachado hoje.

Atenciosamente,

Joao Vendetudo

Figura 13-4. A carta final.

formatada é visto na Figura 13-3. O comando .DF do WordStar referencia o arquivo que criamos com o comando COPY. O comando .RV do WordStar referencia o nome dos campos que serão referenciados na carta e a ordem dos campos.

Um exemplo da carta pronta é visto na Figura 13-4. Uma vez que a vírgula é usada como delimitador no arquivo externo, ela não pode fazer parte de nenhum campo. Qualquer rotina de ADD ou EDIT utilizada sob dBASE II para criar o arquivo deve verificar se o usuário entrou uma vírgula em um campo e rejeitar a entrada para eliminar futuros problemas com o MailMerge. Isto poderia ser feito da seguinte maneira:

```
IF ","$ENDERECO
        @ 23,1 SAY "ERRO – VIRGULA COMO ENTRADA      "
        STORE 1 TO xx
        DO WHILE xx<70
                STORE xx+1 TO xx
        ENDDO
ENDDO
```

Existe uma outra maneira de criar-se um arquivo de endereços para ser usado pelo WorStar. Seria pelo uso dos comandos "?" e "??" para criar o arquivo de endereços como um arquivo alternado. Um exemplo é apresentado abaixo:

```
SET TALK OFF
SET CONSOLE OFF
SET ALTERNATE TO templ.doc
SET ALTERNATE ON
? '.OP'
? '.DF TEMP2.DOC'
? '.REV  '+'  prenome,nome,end1,end2,cidade,est,cep'
```



```
SET ALTERNATE OFF
SET ALTERNATE TO temp2.doc
SET ALTERNATE ON
GOTO TOP
DO WHILE .NOT. EOF
        ?
        ?? CHR(34)+TRIM(prenome)+CHR(34)+','
        ?? CHR(34)+TRIM(nome)+CHR(34)+','
        ?? CHR(34)+TRIM(end1)+CHR(34)+','
        ?? CHR(34)+TRIM(end2)+CHR(34)+','
        ?? CHR(34)+TRIM(cidade)+CHR(34)+','
        ?? CHR(34)+TRIM(est)+CHR(34)+','
        ?? CHR(34)+TRIM(cep)+CHR(34)+','
        SKIP
ENDDO
SET CONSOLE On
SET ALTERNATE OFF
```

USANDO O dBASE II COM PROGRAMAS PROCESSADORES DE PLANILHAS

Se você usa um programa para planilhas, pode estar em condições de enviar dados do programa para planilhas para um arquivo dBASE II. Isto permite ao usuário efetuar simulações, extrair relatórios financeiros e carregá-los no dBASE II. Análise estatística e relatórios complexos podem ser feitos a partir do dBASE II.

Como um exemplo, o SuperCalc pode ser utilizado para criar uma planilha que será lida pelo dBASE II. Os dados do SuperCalc são normalmente salvos como um arquivo que contém fórmulas e são de difícil leitura pelo dBASE II. A opção Disk do SuperCalc pode ser utilizada para gravar um arquivo de dados padrão que pode ser lido pelo dBASE II. Use o comando OUTPUT com a opção "D" (para disco). Você terá um arquivo externo que contém registros completos com tamanho fixo e caracteres de retorno do carro e carga de linha.

Utilizando um utilitário você pode listar este arquivo, estudar a estrutura e criar um arquivo dBASE II com uma estrutura que é idêntica. Leia este arquivo, sob o dBASE II, com o comando APPEND e a opção SDF:

```
APPEND FROM temp.dat SDF
```

Após o arquivo ser carregado, verifique alguns registros com o comando EDIT, modificando a estrutura e repetindo o processo se necessário. Lembre-se que o comando APPEND não seleciona a nível de campo – tudo será movido para o arquivo sob o dBASE II. Você necessitará modificar algumas colunas e títulos, necessários ao SuperCalc, antes de gravar o arquivo e usar o dBASE II para ler os dados.

USANDO O dBASE II COM OUTROS UTILITÁRIOS

O dBASE II pode ser usado com um grande número de utilitários existentes. Alguns foram criados especialmente para o dBASE II, como classificadores, analisadores de estatísticas e utilitários para geração de gráficos. Outros que são vendidos permitem uma conexão mais efetiva com o dBASE II, (como o SuperCalc). Um terceiro grupo inclui programas existentes que, com um pouco de habilidade, trabalharão com o dBASE II.

Um exemplo deste terceiro tipo é o ACCESS/80, um tipo avançado de gerador de relatórios. Utilizando o ACCESS/80, o usuário pode criar tabelas complexas e outros relatórios que não podem ser feitos diretamente no dBASE II a menos de um grande trabalho de programação.

QUATORZE

Usando o dBASE II como Sistema Gerenciador de Databases do Tipo Rede

A maioria dos sistemas que são desenhados sob o dBASE II usará mais de um arquivo de dados. Cada arquivo armazena informações sobre uma coleção de dados. Isto poderia ser uma coleção de peças de um estoque, uma coleção de faturas ou uma coleção de clientes. Uma vez que o dBASE II só pode manter dois arquivos abertos ao mesmo tempo, a melhor estratégia é colocar o máximo possível de coisas em um arquivo. Em outras palavras, o melhor é usar muitos campos em poucos arquivos do que poucos campos em muitos arquivos. Os arquivos podem compartilhar informações tal como o que é um dado em um item sendo a chave em um outro arquivo. Um arquivo de faturas, por exemplo, pode ser indexado pelos números das faturas e conter o número do cliente como um campo de dados. Este mesmo número de cliente pode ser o valor-chave em um arquivo de clientes no qual os endereços e números telefônicos são armazenados. Isto significa que você pode esperar alguma multiplicidade de dados, mas, por meio de um desenho cuidadoso, você poderá minimizar esta duplicação de dados.

ARMAZENAMENTO HIERÁRQUICO DE DADOS

É necessário, algumas vezes, armazenar informações sobre o relacionamento entre partes nos arquivos de dados. Um exemplo seria uma indústria que fabrica peças para automóveis. O estoque contém montagens, submontagens e peças. O relacionamento entre estas três categorias é hierárquico. Cada submontagem é composta por peças. Peças "pertencem" (ou são usadas por) várias submontagens e montagens. Montagens, submontagens e peças são vendidas separadamente e a demanda de peças deve ser projetada contra a quantidade disponível, de forma que as decisões sobre compras e fabricações possam ser feitas (veja a Figura 14-1).

Este tipo de estoque é normalmente processado com um sistema de gerenciamento de databases do tipo rede. Todavia, ele pode ser feito com o dBASE II. Como exemplo, vamos criar um sistema de expedição que armazenará a demanda de peças no arquivo mestre do estoque para ser usada na criação de um relatório mostrando as saídas semanais em número de ordens (Figura 14-2).

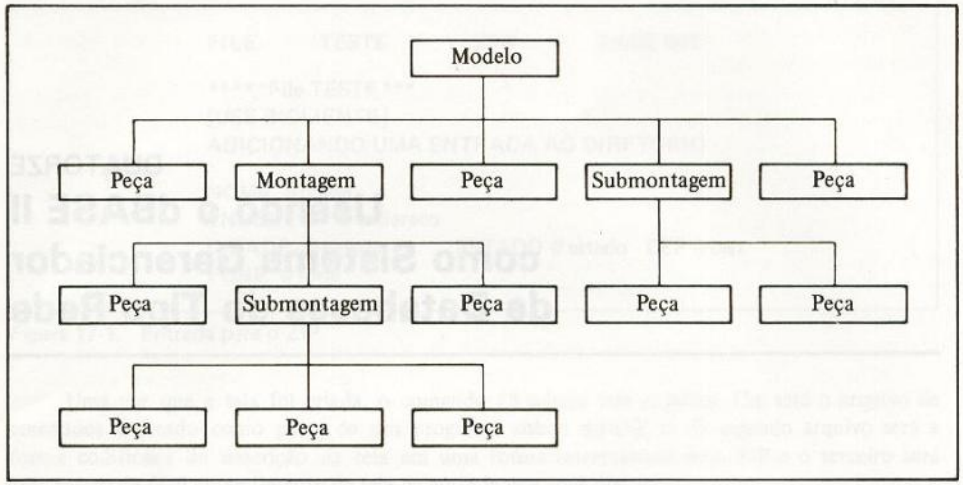


Figura 14-1. Informação organizada em um sistema do tipo rede

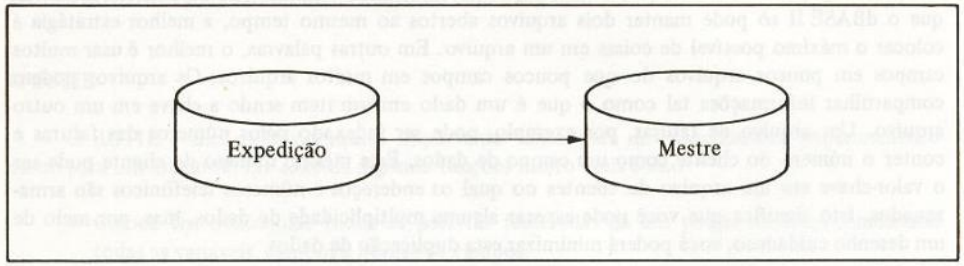


Figura 14-2. Armazenando a demanda no arquivo mestre

CRIANDO O ARQUIVO DEMANDA

Vamos assumir que um arquivo de expedição foi criado com o número de peça que contém um código indicando o tipo de peça (montagem, submontagem ou peça), a quantidade na fatura e a semana de expedição. Isto tudo é armazenado no arquivo DEMANDA. Este mostra a quantidade acumulada projetada de expedição para cada peça em cada semana, ou seja, se existir mais de uma ordem para um número de peça em uma semana, o arquivo DEMANDA mostrará apenas o acumulado da semana. Este arquivo é indexado com base na semana mais o número de peça. Uma listagem reduzida do programa de fase 1 é apresentada aqui.

```
* fase 1 da atualizacao do estoque
SELECT PRIMARY
USE exped
SELECT SECONDARY
USE demanda INDEX demanda
SELECT PRIMARY
DO WHILE .NOT. EOF
        STORE nopeca TO mnopeca
        STORE sem+nopeca TO x
          STORE sem TO msem
          STORE qf TO mqf
          SELECT SECONDARY
          FIND &x
          IF #=0
                APPEND BLANK
                REPLACE nopeca WITH mnopeca
                REPLACE qf WITH mqf
                REPLACE sem WITH msem
                REPLACE nivel WITH 0
        ELSE
                REPLACE qf WITH mqf+qf
        ENDIF
        SELECT PRIMARY
        SKIP
ENDDO
```


Tabela 14-1. Descrição dos arquivos

Arquivo	Campo	Tipo	Tamanho
EXPED			
	NOPECA	C	4
	SEM	C	2
	QF	N	4
DEMANDA			
	NOPECA	C	4
	SEM	C	2
	QF	N	4
	NIVEL	N	1
MESTRE1			
	NOPECA	C	4
	DISP	N	4
	DEMANDA	N	4
	SEM	C	2
LMAT			
	CODIGO	C	4
	NO:PEC	C	4
	UNID	N	2
	TIPO	C	1

A definição dos arquivos de ordem e DEMANDA é vista na Tabela 14-1. O campo nível no arquivo DEMANDA é sempre preenchido com zeros nesta fase (outras fases o preencherão com outros números). O diagrama do fluxo dos dados é mostrado na Figura 14-3.

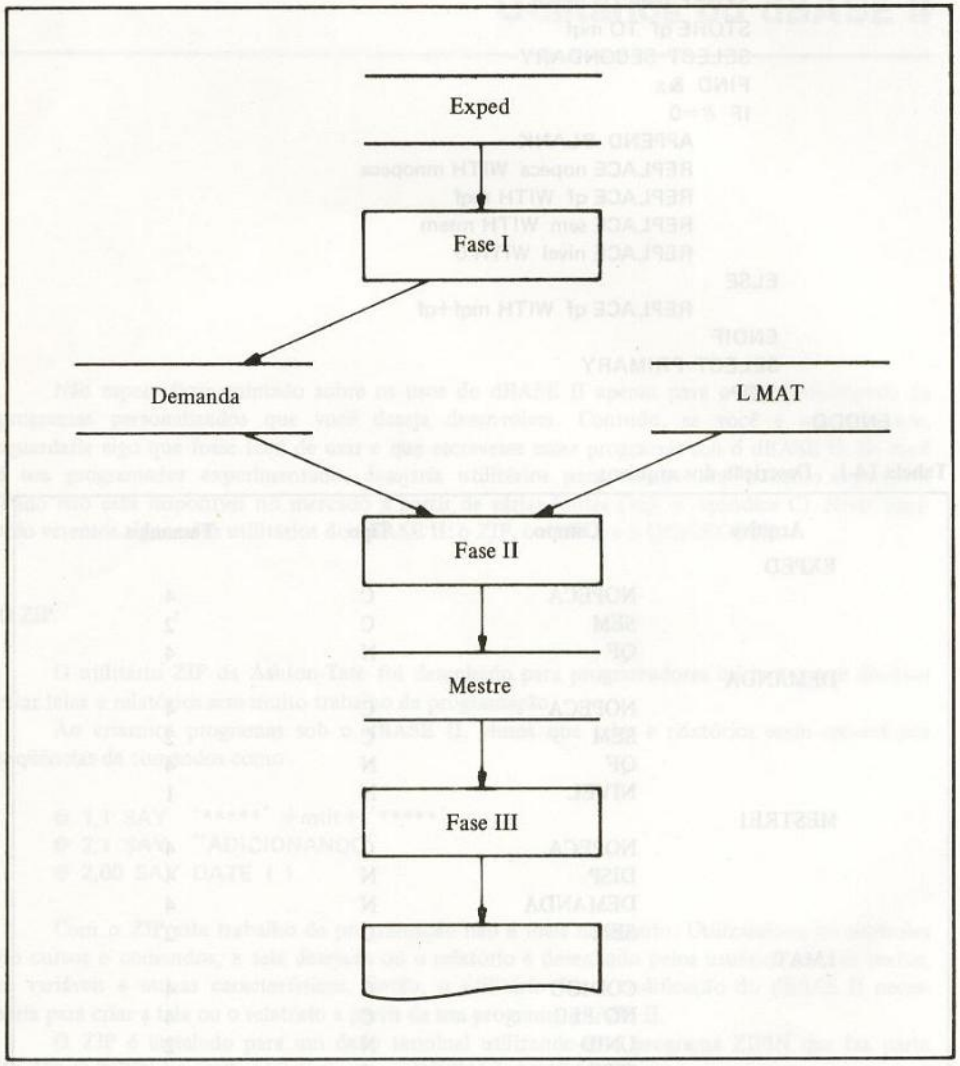


Figura 14-3. Diagrama do fluxo de dados do processamento do estoque

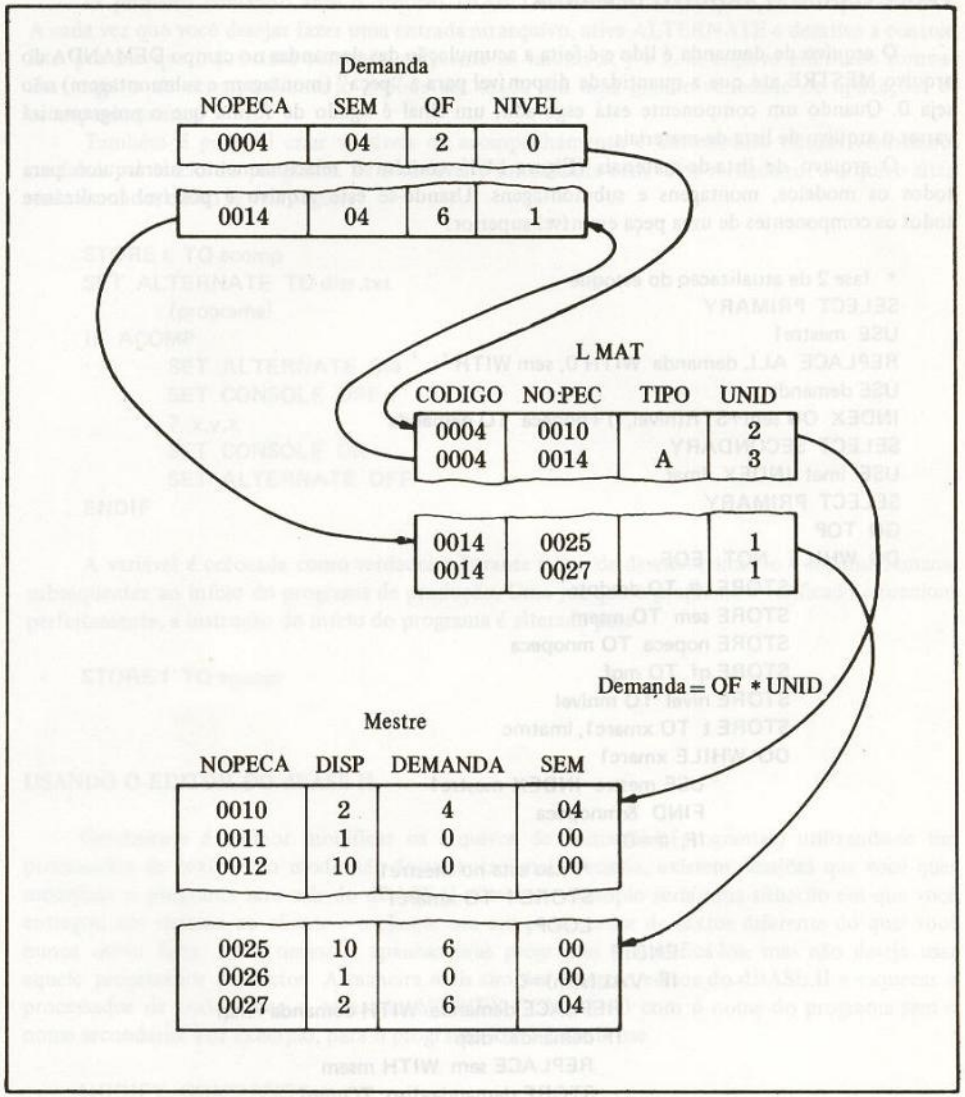


Figura 14-4. Usando o arquivo de lista-de-materiais (LMAT)

PROCESSANDO O ARQUIVO DEMANDA

O arquivo de demanda é lido e é feita a acumulação das demandas no campo DEMANDA do arquivo MESTRE até que a quantidade disponível para a "peça" (montagem e submontagem) não seja 0. Quando um componente está esgotado, um sinal é ligado de forma que o programa irá varrer o arquivo de lista-de-materiais.

O arquivo de lista-de-materiais (Figura 14-1) contém o relacionamento hierárquico para todos os modelos, montagens e submontagens. Usando-se este arquivo é possível localizar-se todos os componentes de uma peça em nível superior.

```
* fase 2 de atualizacao do estoque
SELECT PRIMARY
USE mestre1
REPLACE ALL demanda WITH 0, sem WITH '  '
USE demanda
INDEX ON sem+STR(nivel,1)+nopeca TO demanda
SELECT SECONDARY
USE lmat INDEX lmat
SELECT PRIMARY
GO TOP
DO WHILE .NOT. EOF
        STORE # TO dmdptr
        STORE sem TO msem
        STORE nopeca TO mnopeca
        STORE qf TO mqf
        STORE nivel TO mnivel
        STORE t TO xmarc1, lmatmc
        DO WHILE xmarc1
                USE mestre INDEX mestre1
                FIND &mnopeca
                IF #=0
                        * nao esta no mestre1
                        STORE f TO xmarc1
                        LOOP
                ENDIF
                IF VAL(sem)=0
                        REPLACE demanda WITH demanda+mqf
                        IF demanda>disp
                                REPLACE sem WITH msem
                                STORE demanda-disp TO mqf
                                STORE t TO lmatmc
                        ELSE
                                STORE f TO lmatmc
                        ENDIF
        ELSE
                STORE t TO lmatmc
```



```
        ENDIF
        STORE f TO xmarc1
ENDDO
IF lmatmc
        * comeca a busca no arquivo lmat
        SELECT PRIMARY
        USE mestre1 INDEX mestre1
        SELECT SECONDARY
        FIND &mnopeca
        STORE t TO xmarc3
        DO WHILE .NOT. EOF .AND. mnopeca=mcodigo .AND. xmarc3
                IF #=0
                        * nao esta no lmat
                        SELECT PRIMARY
                        USE demanda INDEX demanda
                        GOTO dmdptr
                        SKIP
                        STORE f TO xmarc3
                        LOOP
        ENDIF
        STORE unid TO munid
        STORE no:pec TO mno:pec
        IF tipo<>'A'
                        * temos a peca em lmat
                        STORE mqf TO xqf
                        SELECT PRIMARY
                        FIND &mno:pec
                        REPLACE demanda WITH munid*xqf+demanda
                        IF demanda>disp .AND. VAL(sem)=0
                                REPLACE sem WITH msem
                        ENDIF
        ELSE
                        * temos a montagem no lmat
                        STORE mqf TO xqf
                        SELECT PRIMARY
                        FIND &mno:pec
                        STORE f TO lmatmc
                        IF VAL(sem)=0
                                REPLACE demanda WITH munid*xqf+demanda
                        IF demanda>disp .AND. VAL(sem)=0
                          REPLACE sem WITH msem
                          STORE (demanda-disp)/munid TO xqf
                          STORE t TO lmatmc
                        ELSE
                          STORE t TO lmatmc
                        ENDIF
```

```
                    * cria novo registro em demanda
                    IF lmatmc
                        USE demanda INDEX demanda
                        APPEND BLANK
                        REPLACE qf WITH xqf*munid
                        REPLACE nivel WITH nivel+1
                        REPLACE nopeca WITH mno:pec
                        REPLACE sem WITH msem
                        USE mestre1 INDEX mestre1
                    ENDIF
                ENDIF
                SELECT SECONDARY
            ENDIF
            SKIP
        ENDDO
    ENDIF
    SELECT PRIMARY
    USE demanda INDEX demanda
    GOTO dmdptr
    SKIP
ENDDO
```

Se em LMATMC é estabelecido que deve ser feita uma busca pelo arquivo de lista-de-materiais, todas as peças componentes são localizadas em LMAT e suas demandas são verificadas no mestre. Se elas estão esgotadas, são adicionadas ao arquivo DEMANDA a um nível superior do atual. Uma vez que o arquivo DEMANDA é indexado por semana, número de peça e nível de ordem, estas peças serão "lidas" posteriormente por um verificador de demanda.

Uma vez que o campo NIVEL é definido como um campo numérico de apenas uma posição, o arquivo DEMANDA expandirá o processamento até dez níveis hierárquicos; certamente suficientes para o nosso sistema de estoque. O programa não vê uma relação entre montagens e submontagens – apenas níveis hierárquicos.

A listagem apresentada foi abreviada para ser mais simples de se entender. Uma versão de produção desta fase deve verificar todas as condições de "não encontrado". O processamento é vagaroso e, para grandes arquivos, deve ser feito durante a noite.

EXTRAINDO RELATÓRIOS A PARTIR DO ARQUIVO DEMANDA

Este processamento requer pouca programação. O relatório é feito copiando-se todas as peças com valores de semana diferentes de zero para um arquivo temporário. Este arquivo deverá ser indexado com base na combinação SEM+NOPECA e impresso.

PROBLEMAS COM A RAPIDEZ

O sistema de lista-de-materiais utiliza quatro arquivos, dos quais apenas dois podem estar abertos ao mesmo tempo. Normalmente, este processamento é feito semanalmente como parte

de outros processamentos de final de semana. Ele deverá, provavelmente, ser feito usando-se apenas discos rígidos, uma vez que os discos flexíveis são muito mais lentos.

Pode-se ganhar alguma rapidez armazenando-se o número do registro das peças no MESTRE1 no arquivo LMAT. O arquivo MESTRE1 poderá, então, ser utilizado sem o índice ganhando-se em velocidade.

QUINZE

Métodos de Pesquisa no dBASE II

São várias as técnicas disponíveis para o programador encontrar determinado registro em um arquivo. Todos eles caem em uma de duas classificações: acesso indexado e acesso por pesquisa.

ACESSO INDEXADO

O acesso indexado permite ao usuário encontrar qualquer registro desejado de um arquivo em menos de dois segundos. Qualquer número de índices pode ser definido para um dado database e até sete índices podem ser atualizados dinamicamente; ou seja, os registros são adicionados ou editados e até sete índices podem ser atualizados automaticamente com cada mudança.

Como exemplo, suponha que temos um arquivo com uma lista de endereços denominado MALA com índices criados sobre os campos NOME, CEP e CHAVE. Toda vez que o arquivo deva ser atualizado, abra-o com

```
USE mala INDEX nome, cep, chave
```

Agora abrimos o arquivo MALA com três índices: NOME, CEP e CHAVE. Qualquer FIND subseqüente apenas utilizará o índice NOME para localizar o registro. Qualquer comando APPEND, EDIT, REPLACE, READ ou BROWSE atualizará todos os três índices.

Para relatórios freqüentemente é desnecessária a utilização de um índice existente. Crie um índice "ad hoc" para cada campo, campo parcial ou combinação múltipla de campos, da seguinte maneira:

```
USE mestre
INDEX ON $(nopeca,2,2)+STR(disp,4) TO temp
```


Note que apenas conjuntos de caracteres são utilizados para construir o índice. Os números são convertidos a conjuntos de caracteres neste exemplo. Qualquer índice anterior do arquivo TEMP será destruído.

ACESSO POR PESQUISA

Em outras condições podemos desejar pesquisar um arquivo e localizar registros baseados em que ele atende ou não certas condições lógicas. Um exemplo seria um database de creches com a seguinte estrutura:

```
STRUCTURE FOR FILE: B:DIA .DBF
NUMBER OF RECORDS: 00034
DATE OF LAST UPDATE: 01/22/83
PRIMARY USE DATABASE
FLD     NAME       TYPE   WIDTH      DEC
001     NOME       C      025
002     END1       C      025
003     END2       C      025
004     CIDADE     C      020
005     EST        C      002
006     CEP        C      005
007     IDMIN      C      002
008     IDMAX      C      002
009     FONE       C      007
010     TIPO       C      007
011     ESPECIAL   C      007
012     ESCOLA     C      003
013     DISTRITO   C      003
**TOTAL**                00134
```

Neste database existem dois campos, TIPO e ESPECIAL que têm um tamanho de sete caracteres. Nós armazenaremos 1 ou 0 em cada caractere. Eles podem ser utilizados para indicar um dos sete tipos de chaves e qualquer das sete características especiais (meio-período, cuidados médicos e outros). O seguinte programa mostra uma rotina de pesquisa que pode ser usada sobre um database. Alguns parâmetros são entrados e utilizados para selecionar os registros que os satisfazem.

```
SET TALK OFF
DO WHILE t
        ERASE
        @ 4,1 SAY  'ENTRE OS PARAMETROS DE PESQUISA (EM BRANCO
                   PARA TERMINAR): '
        STORE '    ' TO minid
        STORE '    ' TO maxid
        STORE '    ' TO mescola
```

```
STORE 'N'   TO t1,t2,t3,t4,t5,t6,t7,s1,s2,s3,s4,s5,s6,s7
STORE '  '  to mdistrito
@ 6,1 SAY   'ENTRE A IDADE MINIMA DA CRIANCA ' get minid
@ 7,1 SAY   'ENTRE A IDADE MAXIMA DA CRIANCA ' get maxid
@ 8,1 SAY   'TIPO DE CUIDADO:'
@ 9,1 SAY   'TEMPO INTEGRAL ' GET t1
@ 9,25 SAY  'FINS DE SEMANA ' GET t5
@ 10,1 SAY  'MEIO-PERIODO ' GET t2
@ 10,25 SAY 'VISITAS ' GET t6
@ 11,1  SAY 'DIAS ' GET t3
@ 11,25 SAY 'ANTES E DEPOIS DA ESCOLA ' GET t7
@ 12,1  SAY 'TARDES ' GET t4
@ 14,1  SAY 'CUIDADOS ESPECIAIS:'
@ 15,1  SAY 'ESPANHOL ' GET s1
@ 15,25 SAY 'CUIDADOS MEDICOS ' GET s5
@ 16,1  SAY 'INDOCHINES ' GET s2
@ 16,25 SAY 'NAO FUMANTES ' GET s6
@ 17,1  SAY 'DESCANSO ' GET s3
@ 17,25 SAY 'PROGRAMA EDUCACIONAL ' GET s7
@ 18,1  SAY 'PROBLEMAS ' GET s4
@ 19,1  SAY 'DISTRITO ESCOLAR ' GET mdistrito
READ
IF minid='  '
        RETURN
ENDIF
USE dia
GO TOP
STORE 20 TO v
DO WHILE .NOT. EOF
        IF VAL(minid)<VAL(idmin) .OR. VAL(maxid)>VAL (idmax)
                SKIP
                LOOP
        ENDIF
        IF mdistrito<>'  '
                IF distrito <>mdistrito
                        SKIP
                        LOOP
                ENDIF
ENDIF
STORE tipo TO mtipo
STORE 1 TO ctr
STORE f TO xmarc
DO WHILE ctr<8
        STORE 't'+STR(ctr,1) TO var
        IF &var='S'
                IF $(tipo,ctr,1) <>'1'
```



```
                        STORE 9 TO ctr
                        STORE t TO xmarc
                        SKIP
                        LOOP
                ENDIF
        ENDIF
        STORE ctr+1 TO ctr
ENDIF
IF xmarc
        LOOP
STORE especial TO mespec
STORE 1 TO ctr
DO WHILE ctr<8
        STORE 'v'+STR(ctr,1) TO var
        IF &var='s'
                IF $(especial, ctr,1)<>'1'
                        STORE 9 TO ctr
                        STORE t TO xmarc
                        SKIP
                        LOOP
                ENDIF
        ENDIF
        STORE ctr+1 TO ctr
ENDDO
IF xmarc
        LOOP
        ENDIF
        @ v,1  SAY nome+'      '+end1
        STORE v+1 TO v
        SKIP
ENDDO
STORE 20 TO v
STORE '  ' TO continue
WAIT TO continue
ENDDO
```

Um tipo diferente de busca (neste caso para uma lista de endereços) permite ao usuário pesquisar por condições lógicas como

```
NOME="SILVA" .AND. CEP="97212"
```

O usuário deve conhecer os campos, os operadores lógicos (AND, OR, NOT) e operações relacionais (=,<,> etc). Desta forma o usuário tem um grande poder de controle sobre a pesquisa.

```
SET TALK OFF
SELECT PRIMARY
USE mala
SELECT SECONDARY
USE selecao
SELECT PRIMARY
STORE '                                    'TO var
ERASE
@ 5,1 SAY 'ENTRE A LOGICA PARA A PESQUISA ' GET var
READ
DO WHILE .NOT. EOF
STORE nome TO mnome
IF &var
        SELECT SECONDARY
        APPEND BLANK
        REPLACE nome WITH mnome
        SELECT PRIMARY
ENDIF
SKIP
```

Suponha, por exemplo, que uma ou mais palavras chaves são usadas no campo CHAVE para cada nome no arquivo. O usuário pode pesquisar, por palavras chaves, a que satisfaça um professor com

```
"PROFESSOR"$CHAVE
```

Se a palavra professor estiver em qualquer lugar no campo CHAVE do registro, o nome será carregado no arquivo SELEÇÃO. Nós podemos, também, combinar uma condição com outra como

```
"REMODELANDO"$CHAVE .OR. "AUTOMOVEL"$CHAVE
```

Neste caso, se "remodelando" ou "automóvel" aparecer no campo CHAVE, o nome será carregado no arquivo SELEÇÃO.

Um outro método diferente de se fazer uma pesquisa é usar o comando LOCATE:

```
LOCATE FOR disp=0
```

Isto posicionará o dBASE II na primeira ocorrência do arquivo em que DISP é igual a 0. Se não existir nenhum registro no arquivo que satisfaça esta condição, EOF terá um valor verdadeiro, logo o programa deverá verificar esta condição.

```
LOCATE FOR disp=0
IF EOF
        @ 23,1 SAY 'REGISTRO INEXISTENTE'
        STORE 1 TO xx
        DO WHILE xx<70
                STORE xx+1 TO xx
        ENDDO
ENDIF
```


COMPARANDO OS DOIS MÉTODOS DE PESQUISA

Qual método de pesquisa é melhor? O método indexado é o mais rápido. O problema é que normalmente apenas poucos campos são indexados. Suponha que o arquivo de endereços tenha índices por número, nome e CEP. Isto lhe dará três índices. A cada nome adicionado, todos os três índices devem ser atualizados. Isto será muito vagaroso para a maioria dos usuários. Se, por outro lado, você só atualizar o número ao registro adicionado, a velocidade de atualização será muito melhor. Se o índice por nome ou CEP for necessário, o índice será criado na hora. Você pode ir um passo além. Utilize uma variável para indicar se o índice CEP ou o índice por nome deve ser atualizado. Se um nome for adicionado este indicador será falso. Se você criar o índice para um relatório ou pesquisa, o indicador será colocado como verdadeiro até o arquivo ser atualizado novamente. O indicador poderá ser salvo em um arquivo de parâmetros ou em um arquivo de variáveis e ser utilizado por todos os programas.

Os demais métodos são úteis quando as condições de pesquisa são complexas. A rapidez irá variar quanto ao tamanho do arquivo que está sendo pesquisado, mas será sempre vagarosa. A chave primária para cada comparação deve ser salva em um arquivo separado. Este arquivo, que representa os registros que satisfazem a pesquisa, pode, então, ser utilizado em uma variedade de aplicações como impressão de etiquetas, cartas pré-formatadas ou um relatório.

DEZESSEIS

Corrigindo Programas com o dBASE II

Se você já escreveu um programa em BASIC, Pascal ou "assembly language", já sabe muito bem que ele nunca corre perfeitamente na primeira vez. Invariavelmente, você tem um ou mais erros. Quanto mais esperado o programa, maior é a possibilidade de erros e a expectativa de encontrá-los. Com um pouco de experiência, utilizando o dBASE II, você verá que pode criar programas que funcionarão logo na primeira vez sem erros. Todavia, você encontrará erros ocasionais e problemas ardilosos que o desafiam e o fazem arrastar-se sobre o seu programa. Quando isto acontece, o dBASE II tem uma variedade de ferramentas para auxiliá-lo.

DESENHANDO PARA OBTER-SE ZERO ERROS

A solução mais prática é desenhar o seu sistema sem erros. Com o dBASE II não é incomum gastar-se, no mínimo, metade do tempo de desenvolvimento desenhando-se o sistema. O diagrama do fluxo de dados e a disposição dos arquivos devem ser feitos e, em alguns poucos casos, apenas diagramas de fluxo devem ser desenhados. Desenhe os menus para mostrar uma divisão funcional dos programas e escreva os programas do menu. Se um computador já existir ou mesmo que um manual exista, teste o sistema paralelamente. Você pode achar que pode escrever um programa diretamente no processador de textos sem um rascunho se você utilizar as disposições dos arquivos e os diagramas de fluxo dos dados como um "esboço". Você pode manter um conjunto de programas comuns (menus, adições, edições) de forma a copiá-los, e adaptando variáveis e nomes de arquivos, formar um novo programa. Este procedimento também reduz erros de datilografia.

A CHAVE ECHO

Se um programa parece ter um problema desafiante, uma boa medida será usar a chave ECHO. Isto provocará o eco na console de cada comando que for executado. Este eco normalmente está desligado quando o dBASE II é inicializado. Se necessário, basta ativá-lo com o comando SET. Antes de iniciar o seu programa, ative o eco usando.

```
SET ECHO ON
```

Agora, inicie o seu programa normalmente

```
DO TESTE
```

A cada comando executado, este será repetido na tela. Se você tem telas de entrada no programa, a tela poderá parecer um tanto confusa, mas tudo (incluindo as entradas e o eco) funcionará. O comando poderá ser parte de um programa de forma que apenas uma parte do programa seja repetida.

```
SET ECHO ON
      (parte do programa)
SET ECHO OFF
```

Você também pode dirigir o eco para uma impressora ao invés da console com

```
SET DEBUG ON
SET ECHO ON
```

Após o término, assegure-se de desativar o eco.

A CHAVE TALK

Outra alternativa para retirar erros é o uso da chave TALK. Se ela está ativa, o resultado de cada comando é apresentado como

```
STORE 2+3 TO x
5
```

Uma vez que normalmente esta chave está ligada, você provavelmente vê o resultado dela estando ativa em alguns programas. A maioria dos programas sob dBASE II, como dissemos, deve iniciar desativando esta chave:

```
SET TALK OFF
```

Para trabalhos de depuração de programas você pode ativá-la:

```
SET TALK ON
```

Isto apresentará os resultados de cada comando executado e permite a você verificar os cálculos. Um exemplo poderia ser a verificação de um programa que arredonde um cálculo. Se

você está calculando e imprimindo uma porcentagem, verá que o dBASE II trunca o resultado. Uma simples rotina pode ajustar o cálculo de maneira que ocorra o arredondamento:

```
STORE (mtotal/xtotal)*100 TO q
STORE STR(q,5,2) TO x
STORE $(x, LEN(x),1) TO y
IF VAL(y)>4
        ?? STR (q+.1,5,1) + '%'
ELSE
        ?? STR (q,5,1) + '%'
ENDIF
```

Este cálculo pode ser verificado com a chave TALK ativa para se assegurar a sua precisão.

A CHAVE STEP

Para problemas realmente difíceis, use a chave STEP. Se você suspeita de uma determinada área do programa, coloque o comando SET STEP ON no início da área problema. Isto permitirá parar o programa para exame das variáveis ou para uma execução direta fora do programa. Um exemplo poderia ser a tentativa do uso de um valor chave para encontrar um registro em um arquivo e o registro não é encontrado. Utilizando-se na chave STEP, você pode parar e examinar o valor da chave e tentar de uma maneira direta:

```
FIND &x
IF #=0
        SET STEP ON
ENDIF
```

Isto irá parar o programa se um registro com um dado valor chave não for encontrado. Do mesmo modo, permitirá ao usuário sair do arquivo de comandos e entrar comandos pela console. Você poderá olhar o valor de x e imaginar o que deu errado.

USANDO ARQUIVOS ALTERNADOS

Uma outra maneira, particularmente eficiente para processamentos noturnos, é apanhar informações de uma variável em um arquivo para examinar os erros posteriormente. Um exemplo seria

```
SET ALTERNATE TO diar.txt
    (programa)
SET ALTERNATE ON
SET CONSOLE OFF
? x,y,z
SET CONSOLE ON
SET ALTERNATE OFF
    (programa)
```


O primeiro comando abre o arquivo DIAR.TXT e elimina qualquer informação anterior. A cada vez que você desejar fazer uma entrada no arquivo, ative ALTERNATE e desative a console. Isto previne que o próximo comando apresente os valores x, y e z. O arquivo alternado torna-se um arquivo de acompanhamento e pode ser usado em uma grande variedade de aplicações de acompanhamento.

Também é possível criar variáveis de acompanhamento e atribuir-lhes valores verdadeiros ou falsos no início do programa. Se a variável é inicializada com valor verdadeiro, o arquivo alternado é criado, como se vê a seguir:

```
STORE t TO acomp
SET ALTERNATE TO diar.txt
        (programa)
IF ACOMP
        SET ALTERNATE ON
        SET CONSOLE OFF
        ? x,y,z
        SET CONSOLE ON
        SET ALTERNATE OFF
ENDIF
```

A variável é colocada como verdadeira durante a fase de desenvolvimento e algumas semanas subseqüentes ao início do programa de produção. Uma vez que o programa é verificado e funciona perfeitamente, a instrução do início do programa é alterada para

```
STORE f TO acomp
```

USANDO O EDITOR DO dBASE II

Geralmente é melhor modificar os arquivos de comandos (programas) utilizando-se um processador de textos e o modo não-documentacional. Todavia, existem ocasiões que você quer modificar o programa sem sair do dBASE II. Um bom exemplo seria uma situação em que você entregou um sistema ao cliente e o cliente usa um processador de textos diferente do qual você nunca ouviu falar. Você necessita apanhar seus programas e modificá-los, mas não deseja usar aquele processador de textos. A maneira mais simples é usar o editor do dBASE II e esquecer o processador de textos. Use o comando MODIFY COMMAND com o nome do programa sem o nome secundário. Por exemplo, para o programa TESTE.CMD use

```
MODIFY COMMAND teste
```

Isto carregará o editor do dBASE II. A Tabela 16-1 mostra os comandos de edição disponíveis.

Existem algumas poucas restrições sobre o editor que devem ser observadas:

1. As linhas físicas podem, apenas, ser menores do que 78 caracteres de comprimento.
2. Caracteres de tabulação são convertidos em espaços.

Tabela 16-1. Comandos do editor

CONTROL-C	Avança
CONTROL-N	Insere uma nova linha
CONTROL-Q	Termina sem gravar alterações
CONTROL-R	Volta
CONTROL-T	Elimina a linha
CONTROL-V	Ativa/desativa modo de inserção
CONTROL-W	Grava as alterações

3. O cursor só pode voltar sobre 1000 caracteres.
4. Não existem formas de pesquisa ou movimentação por blocos.

Ao desenhar programas que possam posteriormente ser usados com o editor do dBASE II, mantenha linhas curtas (menos de 78 caracteres) para minimizar problemas posteriores de edição.

MENSAGENS DE CORREÇÃO

Ocasionalmente você poderá obter a seguinte mensagem durante a fase de desenvolvimento de seus programas sob o dBASE II:

```
*** SYNTAX ERROR ***
      ?
YYYY
CORRECT AND RETRY (Y/N)?
```

onde YYYY é uma parte de uma linha do seu programa. Isto significa que o dBASE II encontrou um erro em seu programa e está solicitando uma correção. Neste momento, você pode entrar com qualquer destas três alternativas:

Y	se você quer corrigir o erro imediatamente.
N	se você deseja saltar esta linha e prosseguir.
ESC	se você deseja sair do seu programa e parar.

Se você responder Y, o dBASE II solicitará "CHANGE FROM:". Entre com o texto errado da forma como ele está, com maiúsculas conforme for necessário, e tecle RETURN. Um novo texto será solicitado. Entre com o texto correto e tecle RETURN. O programa solicitará se outras correções adicionais são necessárias. Entre com N e o programa continuará com a linha corrigida. Isto será uma modificação temporária no programa. Após a execução você necessitará usar seu processador de textos para tornar a mudança efetiva.

ESTRATÉGIA GERAL

A estratégia geral é nunca ter de parar um programa em produção. Os erros devem ser apanhados, mas nunca devem ser usados para abortar o processamento de um programa. Por isso nunca use a chave STEP em um programa de produção. Caso o valor de uma chave não seja encontrado, detete a condição de erro e informe ao usuário ou, caso seja em um processamento noturno, salve a mensagem em um arquivo alternado.

Outras informações sobre a correção de programas podem ser encontradas no Capítulo 12.

DEZESSETE

Utilitários do dBASE II

Não espere ficar animado sobre os usos do dBASE II apenas para o desenvolvimento de programas personalizados que você deseja desenvolver. Contudo, se você é um iniciante, aguardaria algo que fosse fácil de usar e que escrevesse esses programas sob o dBASE II. Se você é um programador experimentado, desejaria utilitários para reduzir seu trabalho repetitivo. Tudo isto está disponível no mercado a partir de várias fontes (veja o Apêndice C). Neste capítulo veremos alguns dos utilitários do dBASE II: o ZIP, o dUTIL e o QUICKCODE.

O ZIP

O utilitário ZIP da Ashton-Tate foi desenhado para programadores iniciantes que desejam criar telas e relatórios sem muito trabalho de programação.

Ao criarmos programas sob o dBASE II, vimos que telas e relatórios eram criados por seqüências de comandos como

```
@ 1,1 SAY  '*****' +mtit+ '*****'
@ 2,1 SAY  "ADICIONANDO"
@ 2,60 SAY DATE ( )
```

Com o ZIP este trabalho de programação não é mais necessário. Utilizando-se os controles do cursor e comandos, a tela desejada ou o relatório é desenhado pelos usuários com os textos, as variáveis e outras características. Então, o utilitário faz a codificação do dBASE II necessária para criar a tela ou o relatório a partir de um programa dBASE II.

O ZIP é instalado para um dado terminal utilizando-se o programa ZIPIN que faz parte do ZIP. O ZIPIN é semelhante ao programa INSTALL do dBASE II.

O ZIP utiliza comandos especiais para controlar a geração da tela (como mostra a Tabela 17-1). Estes comandos podem ser modificados pelo usuário se necessário, mas alguns deles não estão disponíveis em alguns terminais.

Ao iniciar-se uma sessão do ZIP, os seus comandos serão apresentados. Após teclar qualquer tecla, você será solicitado a responder se será usado um arquivo antigo, um novo ou se deseja sair do ZIP. Se você desejar continuar, ser-lhe-á solicitado o nome do arquivo (poderá ser indicada a unidade de disco onde reside) e a tela será limpa exceto a última linha que o ZIP usa para "conversar".

Nesta linha de "conversa" é informada a verdadeira posição do cursor na tela que você está criando. As teclas de retrocesso, retorno e de controle do cursor estão todas ativas. O relatório ou a tela é agora "desenhado" usando-se estas teclas e os comandos disponíveis. Os comandos permitem que você movimente o cursor rapidamente para cima e para baixo da tela ou para o meio de uma linha. Você também pode desenhar ou apagar linhas verticais ou horizontais rapidamente por comandos ou apagar a tela inteira. Um exemplo de geração de uma tela de entrada é apresentado na Figura 17-1.

A tela ou o relatório podem conter no máximo 88 linhas. Comandos de dBASE II podem ser incluídos (como inicialização de variáveis) e as variáveis podem conter entradas ou saídas pelos comandos do ZIP.

Muitas variáveis padrão, como tamanho de página, tabulação e margens podem ser controladas dinamicamente durante o processo de criação.

Tabela 17-1. Comandos do ZIP

\	Prefixo dos comandos
\C	Centre o texto na linha
\T	Move o cursor para o alto da tela
\I	Ativa modo de inserção
\M	Move para o meio da linha ou da tela
\A	Adiciona uma linha ou coluna
\H	Desenha ou apaga uma linha horizontal
\N	Vá para a próxima tela
\F	Vá para a primeira tela
\E	Apaga a tela de trabalho
\Q	Volta ao sistema
\/	Vá para a tela de explicações
\<tab>	Vá para a margem
\B	Vá para o final da tela
\D	Elimina um caractere
\K	Elimina uma linha ou coluna
\V	Desenha ou apaga uma linha vertical
\P	Vá para a tela anterior
\L	Vá para a última tela
\S	Salva a tela

```
FILE:    TESTE        ZPR        PAGE 001

*****File TESTE***
[USE B:CLIENTE]
ADICIONANDO UMA ENTRADA AO DIRETORIO

NOME #nome
ENDERECO #endereco
CIDADE #cidade          ESTADO #estado  CEP #cep
FONE #fone
```

Figura 17-1. Entrada para o ZIP.

Uma vez que a tela foi criada, o comando /S salvará três arquivos. Um será o arquivo de comandos utilizado como parte de um programa sob o dBASE II. O segundo arquivo será a forma codificada de descrição da tela em uma forma interpretável pelo ZIP e o terceiro será uma versão de fácil entendimento da tela de entrada que você digitou.

Este utilitário é prático para programadores iniciantes, mas programadores mais experimentados o definirão como limitado e de grande consumo de tempo. Alguns produtos como o QUICKCODE integrarão as linhas de comando com a sua tela.

O dUTIL

O dUTIL é um utilitário muito importante tanto para um programador experimentado como para um iniciante. Ele executa algumas funções muito úteis como

- Coloca em maiúsculas todas as palavras reservadas de um programa e em minúsculas todas as variáveis, campos e nomes de campos.
- Verifica todos os IF e loops DO para assegurar-se que todos estão fechados. O dUTIL se encarregará de todas as finalizações (ENDDO e ENDIF) com expressões que correspondem ao comando DO ou IF inicial.
- Ordena seu programa em níveis igualmente espaçados. Isto é muito útil se você está movendo partes de um programa para um outro programa com uma ordenação diferente.
- Imprime mapas do relacionamento hierárquico entre programas.

O dUTIL é um produto da Fox e Geller. Ele deve ser instalado, como o dBASE II, para cada terminal. É um processo muito simples que utiliza o programa DINSTALL. Você poderá adequar o dUTIL para terminais especiais.

Após a instalação, o dUTIL é carregado e o menu principal é apresentado (veja a Figura 17-2). O usuário pode entrar com os nomes dos arquivos de entrada e de saída (opções 1 e 4) como também controlar outras opções.

O arquivo de saída normalmente tem o mesmo nome do arquivo de entrada com o tipo de arquivo igual a NEW. Você pode desejar alterar o valor máximo de DO para 9 e a tabulação


```
                    dUTIL MAIN MENU

INPUT FILES                          OUTPUT FILES
(1) SOURCE      – mainin. cmd        (4) SOURCE – pretty.new
(2) KEYWORDS – keywords.key          (5) TREE   – dtree.tre
(3) UNITLIST    – .unt               (6) SUBMIT – dtree.sub

(10) MAIN DRIVE          – A:        (14) SOURCE EXTENSION – cmd, new
(11) ALTERNATE DRIVE – B:            (15) LISTING DEVICE     – prn

(12) SOURCE TAB SIZE  – 4            (16) MAX DEPTH DO       – 5
(13) TREE TAB SIZE    – 6            (17) MAX DEPTH INCLUDE – 5

(20) skip      (21) include          (22) caps      (23) tree
(24) code      (25) display          (26) do        (27) process
(28) page      (29) sub              (30) nounit

(0) EXECUTE    (31) SAVE             (99) EXIT
        Please enter selection number —
```

Figura 17-2. Menu principal do dUTIL.

para 8. Entre com 0 para iniciar o processamento. A saída e as mensagens de erro serão apresentadas na tela, enquanto o arquivo de entrada é processado.

Algumas das opções incluem a capacidade de:

- Integrar chamadas DO a programas separados como um simples programa.
- Criar um arquivo que contenha um diagrama de sua árvore de arquivos de comando que mostra as chamadas a outros arquivos.
- Criar um arquivo que possa ser processado pelo programa CP/M SUBMIT para imprimir todas as listagens dos programas.
- Processar arquivos de entrada na modalidade de lotes.
- Usar um comentário INCLUDE para incluir partes de outros programas em tempo de processamento. Isto será útil se você tiver uma coleção de pequenas rotinas utilizadas em muitos programas.
- Verificar se cada DO tem um ENDDO correspondente, se cada IF tem um ENDIF correspondente e se cada CASE tem um ENDCASE correspondente.

O dUTIL é útil na depuração de programas e para fazer listagens de alta qualidade deles. Para um desenvolvimento sério de programas, o dUTIL é um utilitário muito importante.

O QUICKCODE

Todos nós sonhamos com o utilitário definitivo. Sentando-se no terminal do computador, você poderia entrar com "CONTABILIDADE" e o sistema escreveria um sistema contábil

completo. Nós ainda não atingimos este ponto, mas o QUICKCODE tem algumas coisas nesse sentido – ele escreverá programas para você.

O QUICKCODE, um produto da Fox e Geller, é quase o produto perfeito para o programador iniciante. O usuário cria uma tela de entrada. Então ela é utilizada para criar o database e uma coleção de doze programas que podem adicionar, editar ou relatar o database. O utilitário é razoavelmente fácil de ser usado, mas considerando suas capacidades tem-se um certo trabalho para conhecê-lo.

```
                QUICKCODE: HELP FOR THE WEARY USER

                      SCREEN EDITING COMMANDS

  CMD        ENTER      CMD        ENTER      CMD        ENTER

 RIGHT                  LINE                  GRID
 LEFT                   COLUMN
 UP                     DEL LIN
 DOWN                   DEL COL
 MIDDLE                 CENTER
 LMARGN                 LSHIFT
 RMARGN                 RSHIFT
 TAB                    *ERASE

                        OTHER COMMANDS

  CMD        ENTER      CMD        ENTER

 HELP                   FIELDS
 SAVE
 EXIT
 *QUIT

                   COMMANDS YOU CAN TYPE NOW

  CMD  WHAT IT DOES:          CMD  WHAT IT DOES:
    C  CONFIGURE SYSTEM         O  OLD SCREEN
    S  SCREEN CHARACT.          N  NEW SCREEN
    X  OUTPUT OPTIONS           T  LOAD TEXT FILE
    M  QUICKMENU                   GENERATE PGMS

    Q  **QUICKSCREEN MODE       E  ***EXIT***

                      CURRENT SCREEN IS
```

Figura 17-3. Menu principal do QUICKCODE

O QUICKCODE, como o dBASE II e o dUTIL, deve ser instalado antes de ser utilizado. Isto é um processo simples utilizando-se o programa QINSTALL. Você também pode adequar o QUICKCODE para terminais especiais.

Após a instalação, o QUICKCODE é carregado e o menu principal é apresentado (veja a Figura 17-3). Este menu apresenta o repertório dos comandos. A qualquer instante você poderá voltar a esta tela teclando "?".

A tela dá a você um mapa dos códigos de controle para o seu sistema. A maioria dos comandos da esquerda refere-se ao QUICKSCREEN que é integrado ao QUICKCODE (usando estes comandos, você pode movimentar o cursor por sobre a tela para criar a tela de entrada). Os valores padrão apresentados podem ser alterados com o comando "C". Você pode, se desejar, modificar os valores utilizados em qualquer dos comandos.

Para começar, você precisa selecionar o nome do arquivo (se você não o fizer, o sistema usará o nome padrão NONAME). Use a opção "N" se for um novo database e "O" se você está revisando um database velho. Você também pode entrar com o nome da unidade padrão.

Após o nome do arquivo ser selecionado, entre no modo QUICKSCREEN usando a opção "Q". O sistema apresentará uma tela quase limpa (ela conterá um nome de variável e pouca coisa a mais). O QUICKSCREEN como um utilitário interno de uma certa maneira similar ao ZIP.

Você poderá, então, desenhar sua tela de entrada. Nomes de variáveis e textos podem ser usados como

```
NOME      ;prim      ;ulti
```

Os ponto-e-vírgula são usados para definir variáveis que serão utilizadas como nome de campos no database. O texto sem ponto-e-vírgula será apresentado na tela de entrada na forma como você o colocou. Os comandos de controle para o menu principal estão disponíveis na criação de telas.

Uma vez que a tela foi criada, teclando-se CONTROL-B, iremos para a tela FIELDS. Ela é usada para alterar tamanho de campos, definir valores de chave que serão utilizados nos índices, definir valores padrão e definir os valores máximos e mínimos que serão utilizados para validação de cada variável. Você também pode selecionar os tipos dos campos. O QUICKCODE suporta sete tipos de campos: numéricos, inteiros, caracteres, datas, telefones, seguros sociais e lógicos.

Uma vez definidos os campos, você pode sair do QUICKSCREEN e teclar ESCAPE. O arquivo mestre, os arquivos de telas e os doze programas integrados serão criados com um menu mestre. Entre no dBASE II, chame o menu principal e você poderá começar a carregar os dados com os programas criados.

O QUICKCODE tem muitas outras características que podem ser úteis:

- Uma lista de valores pode ser armazenada para um campo e todas as novas entradas para o arquivo são verificadas contra esta lista durante a fase de entrada.
- Uma característica automática está incluída e pode criar programas independentes ou parte de outros programas.
- Os databases existentes podem ser utilizados.
- Você pode criar menus especiais.
- Uma grande variedade de opções padrão podem ser mudadas pelo usuário.

Os principais programas criados pelo QUICKCODE incluem:

CMD Um programa menu de uso geral criado para o usuário.

ADD Adiciona registros no database.

ED Edita registros do database.

GET Pesquisa o database utilizando os índices existentes ou mesmo sem eles.

RPT Extrai relatórios a partir do database.

LBL Imprime etiquetas de endereçamento a partir do database.

WS Cria um arquivo de endereços para o WordStar a partir do database para imprimir cartas pré-formatadas.

Além desses, os seguintes programas são criados e serão chamados pelos seis programas principais:

IO Apresente a tela de entrada e saída.

OUT Mostra campos de dados e títulos.

FAU Define valores padrão durante a entrada.

VAL Executa verificação e validação dos dados de entrada.

O usuário pode criar programas de menu para quase todas as aplicações e pode definir a tela de menu e as chamadas dos programas.

O QUICKCODE tem algumas limitações. Normalmente, ele é usado com apenas um database, como para a criação de um sistema de mala direta. Se você tem vários arquivos, informações comuns e chaves, o QUICKCODE torna-se difícil ou impossível de ser usado. Na maioria dos casos os programas de relatório, impressão de etiquetas e cartas pré-formatadas, que o QUICKCODE gera, não são tão sofisticados quanto necessário. O manual não tem índice e nem cartão de referência, mas podem ser modificados rapidamente. O QUICKCODE é rápido, escrevendo todos os programas necessários em apenas poucos minutos. Ele dará a você um bom conjunto de programas básicos a partir dos quais se pode construir um sistema. Os exemplos são bons para quem esteja começando com o dBASE II e deseja escrever seus próprios programas.

USANDO UM CONSULTOR DO dBASE II

Uma questão importante para grande parte de usuários do dBASE II é quando escrever seus próprios programas e quando chamar um consultor. O dBASE II é tão simples que permite que os usuários com pouca ou nenhuma experiência com computadores possa rapidamente começar a escrever seus próprios programas.

Se a aplicação é simples (como um programa de mala direta com um único arquivo), o usuário com um mínimo de experiência com computadores pode escrever o programa ou usar o QUICKCODE para escrever o programa e descobrir uma excelente experiência de aprendizado. Se você está criando um sistema com muitos programas e arquivos, geralmente é melhor chamar pelo menos um consultor para ajudá-lo com o desenho.

O consultor deverá conhecer desenho de sistemas e dBASE II, estará apto a ajudá-lo

a dividir o sistema funcionalmente e a criar os arquivos necessários. Análise de sistemas é uma ciência a parte.

Você pode querer auxílio para escrever programas, mas geralmente é melhor deixar o consultor escrever todos os esqueletos do sistema que fará o que você necessita. Informe ao consultor quais são seus objetivos a curto e longo prazo. Após os programas básicos estarem prontos, você poderá estudar o dBASE II e adicionar melhoramentos ou escrever novos programas.

Esteja certo de contratar um consultor que lhe forneça uma listagem da fonte dos programas. Você precisará dela se planeja modificar os programas.

DEZOITO

Do Desenvolvimento à Produção

Os programas são normalmente destinados para serem usados por uma dada aplicação. Uma vez que o programador completou seu trabalho de desenvolvimento, o programa é movido do estado de "desenvolvimento" para o estado de "produção". Isto deve ser feito de uma maneira organizada que minimize dificuldades por parte do usuário.

O DESENVOLVIMENTO

Desde o início da codificação de um programa, tudo deve ser feito tendo em mente o usuário. Nunca deixe o usuário adivinhar o que está acontecendo. As mensagens devem ser apresentadas na tela em um claro português. Use códigos de erro e documente-os, bem como o que o usuário deve fazer, se ocorrerem, no manual do usuário. Identifique todas as circunstâncias onde os registros poderão não ser encontrados e assegure-se de que o usuário seja avisado. Durante operações demoradas, mande continuamente mensagens sobre o que está acontecendo e o que deve ser feito se um erro ocorrer. Assegure-se de que o usuário não possa destruir o sistema ou os arquivos. Ao ser enviada uma tela de entrada de dados, envie também uma mensagem sobre como retornar ao menu caso não se queira entrar com dados. Feche os arquivos a partir do momento em que não mais serão utilizados. Tente apanhar todas as operações estranhas que você possa imaginar. Mantenha o nível das mensagens de forma inteligível ao usuário informando-o de como as coisas estão indo. Não use observações engraçadas em demasia, pois após um certo período de tempo essas observações cansarão o usuário.

ESCREVENDO A DOCUMENTAÇÃO

Uma boa documentação é composta por um manual de aprendizagem, um manual de referência e uma tabela para consultas rápidas. Todo manual deve ter um índice. Na maioria

das aplicações sob o dBASE II, você poderá estar apto a condensar isto em um manual técnico e em um manual de usuário. O manual do usuário destina-se ao usuário não-técnico. Ele não contém linguagem técnica e é escrito para ser lido e utilizado por alguém com uma experiência técnica mínima. O manual técnico, pelo contrário, é destinado aos programadores que farão a manutenção dos programas. Ele contém as listagens dos programas, descrição dos arquivos, definição dos índices e diagrama dos fluxos dos dados.

O manual do usuário deve conter toda informação para que uma pessoa não-técnica possa inicializar o sistema e usá-lo. Ele deve acompanhar a divisão funcional do menu com alguns capítulos sobre a inicialização do sistema e sobre os procedimentos para se efetuar cópias de segurança dos arquivos. Um apêndice deve mostrar todas as mensagens de erro e o que fazer em cada uma delas. Deverá existir, também, um glossário. Os detalhes das subdivisões devem seguir a subdivisão de seus programas, a menos dos capítulos de inicialização e de cópia. A Figura 18-1 é um esboço de um manual para usuários em uma companhia de manufaturas.

Um desenho de cada tela deve ser apresentado e referenciado no texto, se possível evite fotografias (obtenha uma cópia da tela).

A importância deste manual nunca deverá ser minimizada. Se você é um consultor e o usuário telefona-lhe com uma questão, oriente-o para a parte do manual onde a dúvida é resolvida. Faça com que o usuário se habitue a recorrer ao manual para resolver as questões. Isto lhe evitará muitos telefonemas e lhe sobrará tempo.

COLOCANDO UM PROGRAMA EM PRODUÇÃO

Quando um programa é escrito e completamente testado, é inevitável que ele ainda tenha alguns probleminhas. Ao contrário dos programas comerciais que são, geralmente, bem testados por um usuário base antes de ser vendido, o seu programa não é testado sobre nenhuma base. O usuário deve ter isto em mente. Se hoje o processo é manual, faça um processamento paralelo

AUTOMO MANUFATUREIRA
Manual do Usuário

I Introdução
II Inicialização
III Explicação do Sistema
IV O Sistema de Estoque
V O Sistema de Ordens de Compra
VI O Sistema de Expedição

Apêndice A: Glossário
Apêndice B: Mensagens de Erro
Apêndice C: Procedimentos Operacionais e de Cópias de Segurança

Figura 18-1. Um exemplo de esboço para um manual de usuário.

com o novo sistema. Se já existe um sistema computadorizado, processe os dois em paralelo. Compare os resultados e verifique a confiança do novo sistema neste período.

Os usuários podem ter idéias para mudanças. Ao contrário de outras linguagens de programação, você freqüentemente achará fácil incorporar as mudanças ao sistema. Por isso, escute as sugestões. Agrupe-as por programa e modifique cada um deles quando necessário, aplicando todas as mudanças que se aplicam ao programa de uma vez.

TREINANDO O USUÁRIO

O treinamento do usuário começa bem antes do programa ser colocado no status de produção. Os usuários devem ser envolvidos desde o início do desenho do sistema. Como um programador, você deve acompanhar o usuário pela entrada de dados e pelos seus passos antes de iniciar o desenho do sistema. Onde existir um processo manual ou computadorizado sendo utilizado, você deverá estar ciente do fluxo atual das informações. Ao ser desenvolvido o sistema, você deve esperar uma confiança crescente por parte do usuário como um resultado de suas perguntas e visão de construção do sistema. Não é incomum em sistemas sob dBASE II gastar-se um terço ou um quarto do tempo de desenvolvimento em diálogos com o usuário no planejamento do sistema.

Quando o programa for passado para produção, o usuário já saberá o que esperar e o trauma da mudança será minimizado. Assegure ao usuário que o sistema não pode ser danificado. Encoraje-o, dê-lhe suporte e seja paciente.

O manual do usuário deverá estar pronto quando o usuário começar a usar o programa. Mostre ao usuário como utilizar o manual e como a documentação corresponde a organização do sistema. Lembre-se que você, como desenhista, conhece muito sobre o seu sistema, mas o usuário pode conhecer o sistema de produção. Você deverá mudar o seu sistema quando necessário para atender as necessidades do usuário.

APÊNDICE A
Glossário

Arquivo. Uma coleção de registros relacionados tratados com um único item.

Arquivo de acesso direto. Arquivo no qual o tamanho dos registros é fixo e o tempo de chegar em qualquer registro do arquivo é o mesmo (veja também arquivos seqüenciais).

Arquivo de transações. Um arquivo usado para armazenar dados temporariamente onde ele possa ser listado, editado e acompanhado até ser usado na atualização de um outro arquivo.

Arquivo externo. Um arquivo externo ao sistema de databases que pode ser gravado ou lido. Os arquivos externos são utilizados na transmissão de dados entre sistemas de gerenciamento de databases e programas de aplicação externos ou outros sistemas de gerenciamento de databases.

Arquivos seqüenciais. Arquivos nos quais o tamanho dos registros é variável e o tempo para chegar a um determinado registro é uma função da sua distância da atual posição no arquivo.

Byte. Um caractere de memória no computador. Oito bits formam um byte.

Campo. A menor unidade de dados que pode ser usada para descrever um item.

Chave. Um item de dado (ou campo) utilizado para localizar um registro.

Database. Uma coleção de informações interligadas ao usuário por uma coleção de programas.

Delimitador. Qualquer caractere ou símbolo utilizado para separar informações em um registro.

Diagrama de fluxo dos dados. Um desenho em um sistema de informações que mostra blocos funcionais, arquivos e o fluxo da informação.

Gerenciador de multiarquivos. Uma coleção de programas que providencia ferramentas para adicionar, editar e extrair relatórios de arquivos de dados.

Índice. Uma tabela usada para definir a localização de um registro baseada em campos deste registro.

Lançamento. O processo de utilizar-se um arquivo de transações para atualizar o arquivo mestre.

Linguagem estruturada. Uma linguagem em que os programas podem ser desenvolvidos em módulos bem definidos e executados hierarquicamente ou de cima para baixo.

Loop. Seqüência de instruções que, no seu final, existe uma condição que faz com que seja executada novamente a mesma seqüência.

Registro. Uma coleção de itens de dados relacionados.

Registro físico. Um registro de um database da maneira como é visto pelo gerenciador de databases. Ele inclui o registro lógico e qualquer indicação ou encadeamento de informações necessárias criadas pelo database (veja também registro lógico).

Registro lógico. Um registro de um database como visto pelo usuário e por um programa externo ao database (veja também registro físico).

Sistema de gerenciamento de databases. Um grupo ou coleção de programas que conecta o usuário a uma coleção de informações.

Sistema em rede. Um sistema de database no qual um registro subordinado pode ser subordinado a mais de um registro. Registros "filhos" podem ter mais de um registro "pai".

Sistemas hierárquicos. Um sistema de database no qual alguns registros estão subordinados a outros em uma estrutura semelhante a uma árvore.

Sistema operacional. Um grupo de programas que organiza um grupo unidades físicas em um sistema de trabalho integrado permitindo as pessoas que as usem.

Sistema relacional. Um sistema de database constituído de qualquer número de arquivos bidimensionais ou relações vistas como uma simples unidade e capaz de combinar elementos de dados baseado em qualquer número de critérios relacionais.

Variável. Uma localização na memória do computador que pode ser usada para armazenar dados.

APÊNDICE B

Bibliografia

Brin, Stan. "The Evolution of dBASE II." *Popular Computing*, Fev. 1983, p. 74-80 e 154.
Uma interessante narrativa da história do dBASE II e como ele evoluiu.

Byers, Robert A. *Everyman's Database Primer.* Culver City, Califórnia: Ashton-Tate, 1982.
Uma boa introdução ao conceito de databases destinado para alguém que esteja iniciando-se no dBASE II.

Edelson, George. "dBASE II: A Relational DBMS for CP/M." *Interface*, Age, v. 6, nº 8 (Ago. 1981), p. 60-63.
Uma explanação sobre o dBASE II.

Green, Adam B. *dBASE II User's Guide.* Duxbury, Massachusetts: Software Banc, 1982.
Um manual de instrução do dBASE II utilizado em vários seminários nos Estados Unidos.

Kruglinski, David. *Database Management Systems.* Berkeley, California: Osborne/McGraw-Hill 1983.
Uma excelente explanação sobre os maiores sistemas de gerenciamento de databases disponíveis para computadores pessoais.

Townsend, Carl. *CP/M Database Management.* Portland, Oregon: Dilithium Press, 1983.
Contém ensinamentos sobre gerenciamento de databases e uma explanação sobre os maiores sistemas de gerenciamento de databases com exemplos, especificações etc. Este livro é útil se você está planejando comprar um sistema e deseja comparar o dBASE II com outras alternativas.

APÊNDICE C

Lista de Produtos

A lista a seguir contém informações sobre produtos que trabalham com o dBASE II e seus respectivos fabricantes. Uma rápida descrição do objetivo do produto é dada entre parênteses.

ABSTAT (Análise estatística para o dBASE II)	Anderson-Bell 5336 S. Crocker St. Littleton, CO 80120 303-794-7509
ACCESS/80 (Gerador de relatórios)	Friends Software Tioga Building, Suite 440 P.O. Box 527 Berkeley, CA 94701 415-540-7282
Autocode 1 (Geração de programas para o dBASE II)	Stemmos Ltd. 666 Howard St. San Francisco, CA 94105 415-777-3800
DBPlus (Classificação, compressão e transformação de arquivos)	Human Soft 661 Massachusetts Ave. Arlington, MA 02174 617-641-1880
dGRAPH (Apresentação de gráficos com o dBASE II)	Fox and Geller P.O. Box 1053 Teaneck, NJ 07666 201-794-8883
DUTIL (Utilitários para o dBASE II)	Fox and Geller
QUICKCODE (Codificação sob o dBASE II)	Fox and Geller

APÊNDICE D

Procedimentos para Desenvolvimento de Programas

Os seguintes passos são recomendados para assegurar que seu programa seja facilmente interpretado tanto por você como pelos usuários que irão trabalhar com ele. Você deverá incorporar todos estes procedimentos em cada programa que você escrever.

1. Todos os programas devem ter comentários no seu início que contenham as seguintes informações:

   Nome do programa e sua descrição funcional
   Copyright se necessário
   Autor
   Nome do Arquivo
   Arquivos de entrada e índices utilizados
   Arquivos adicionais
   Versão, data e iniciais do programador

   Por exemplo:

   ```
   * estoque minimo – processa estoque minimo
   * por Carl Townsend
   * arquivos de entrada: exped, mestre, lmat
   * arquivos de saida: mestre
   * arquivos adicionais:nenhum
   * versao 1.0 13/02/83 CT
   ```

2. A estrutura da programação deve ser conforme a saída do programa do dUTIL discutida no Capítulo 17. A maneira mais simples de fazer isto é processando todos os programas com o dUTIL. A estrutura da programação é:

   a) Todas as palavras reservadas do dBASE II devem estar em maiúsculas.
   b) Todas as palavras não reservadas devem estar em minúsculas.
   c) Loops DO e IF devem ser comentados.
   d) Linhas de comando não devem ter mais de 100 caracteres.
   e) Ninhos de comandos DO devem ser limitados a nove níveis.
   f) Comandos IF e DO devem ser ordenados consistentemente.

3. Comentários deverão ser usados liberalmente por programa para explicar o que está ocorrendo. Isto não afetará em muito a velocidade de programação, mas para manter os programas tão rápidos quanto possível mantenha os comentários fora do loop a que se referem. Por exemplo:

```
* pesquisa Imat por modelo
DO WHILE .NOT. EOF
```

4. Chamadas DO para programas adicionais farão cair a velocidade do programa principal, uma vez que o arquivo do programa adicional necessite ser aberto. Como regra geral, o arquivo é aberto apenas a primeira vez que o programa é chamado. Se múltiplas chamadas DO para outros programas estão sendo usadas, será mais rápido mantê-las junto ao programa principal.

5. Programas que se utilizam de vídeos devem mostrar na tela um cabeçalho mostrando o nome da empresa, função e data. O cabeçalho deverá sempre ser mantido na tela até que seja substituído por outro cabeçalho. Por exemplo:

```
ERASE
@ 1,1 SAY '***** SILVA MANUFATUREIRA *****
@ 2,1 SAY 'ADICAO AO ESTOQUE'
@ 2,60 SAY DATE ( )
```

6. Todos os programas e nomes de arquivos devem estar de acordo com a convenção de nome de arquivos que está na documentação de sua aplicação.

7. Não use como nome de variáveis as palavras reservadas para o dBASE II.

8. Se um programa gastar mais de 5 segundos imprimindo, indexando ou classificando, você deve dizer ao usuário o que está acontecendo e, se possível, mostrar-lhe uma mensagem que fique atualizada constantemente. Por exemplo, em um programa atualizando um cadastro de endereços, a mensagem apresentaria o número do registro que está sendo processado no momento.

9. Use tantas mensagens quantas possíveis para orientar o usuário. Nunca suponha que o usuário tenha qualquer manual ou folha de referência. Por exemplo, em um programa de emissão de relatórios deve pedir ao usuário que ponha papel na impressora.

10. Se o programa encontrar um erro em um arquivo, ele deve apresentar ao usuário, em um português bem claro, a causa do erro e demais informações que ajudem o usuário a descobrir a causa do erro. Se possível, mantenha o processamento ativo.

11. Se múltiplos caminhos são necessários em um programa (como em um programa de menu), utilize o comando DO CASE em vez do comando IF. O comando DO CASE é muito mais rápido.

12. Utilize menus para o acesso e a operação de programas. Divida os programas em grupos funcionais e utilize os submenus necessários para cada grupo. Não espere que o usuário lembre os nomes dos programas. Os menus devem dar descrições funcionais.

13. Esteja certo de que o sistema inclui programas para recuperação de arquivos de dados e sua restauração, e um programa de reindexação de todos os arquivos.

14. Ao iniciar um programa que não possa ser interrompido (como um programa de lançamentos), assegure-se de que ESCAPE OFF foi estabelecido e, ao término do programa, ESCAPE ON seja restabelecido.

15. Reserve a vigésima-terceira linha da tela para mensagens.

16. Estabeleça uma rotina de retardamento e acione o alarme do terminal ao apresentar uma mensagem, de maneira que chame a atenção do usuário e dê tempo para que ele leia a mensagem.

17. Sempre feche os arquivos que não estejam sendo utilizados. Uma maneira simples de se fazer isso é usando o comando CLEAR em cada loop de submenu. Ao ser terminado um programa, o controle é passado para o submenu que executará o comando CLEAR.

18. Uma vez que apenas dois arquivos podem estar abertos ao mesmo tempo, carregue cada arquivo com tantos campos quanto possíveis de forma a reduzir o número de arquivos utilizados.

19. Libere as variáveis com a maior freqüência possível. Novamente o comando CLEAR em um submenu será uma boa técnica. Se as variáveis vão além dos limites deste submenu (como o nome da empresa) armazene-as em um arquivo MEM.

20. Antes de escrever um programa, faça um desenho da análise, os diagramas dos fluxos dos dados e o desenho dos arquivos. Isto evitará erros e correções nos programas.

21. As entradas efetuadas pelo usuário devem ser interpretadas em maiúsculas de maneira que o usuário faça a seleção tanto em maiúsculas como em minúsculas. Utilize a função !:

```
CASE ! (opcao) = 'A'
     DO estadi
CASE ! (opcao) = 'E'
     DO estedi
```

APÊNDICE E

Estrutura dos Arquivos de Dados e Índices

DATA FILES

A Tabela E-1 mostra a estrutura dos arquivos de dados.
O material deste apêndice é reproduzido com a permissão da Ashton-Tate.

Tabela E-1. Estrutura dos arquivos de dados

BYTE	CONTEÚDO	SIGNIFICADO
0	02H	Identificação de arquivo do dBASE II
1-2	número de 16 bits	Número de registros neste database
3-5	3 bytes	Data da última atualização (MM/DD/AA) (binário)
6-7	número de 16 bits	Tamanho dos registros (binário)
8-520	conjunto de 16 bytes	Descrição dos campos (32 no total)
	byte 0-9	Nome do campo em ASCII (preenchido com zeros)
	byte 10	Zero (binário)
	byte 11	Tipo do campo: "C", "N", ou "L" (ASCII)
	byte 12	Tamanho do campo (binário)
	byte 13-14	Reservados
	byte 15	Número de casas decimais (binário)

Notas:

1. Os registros de dados são precedidos por um byte que é um espaço (hexadecimal 20) se o registro está ativo ou por um asterisco (hexadecimal 2C) se estiver eliminado.
2. Os campos de dados são colocados no registro sem separações ou marcação de término.
3. O último campo de descrição é seguido por um caractere de retorno de carro (hexadecimal OD).

ARQUIVOS DE ÍNDICE

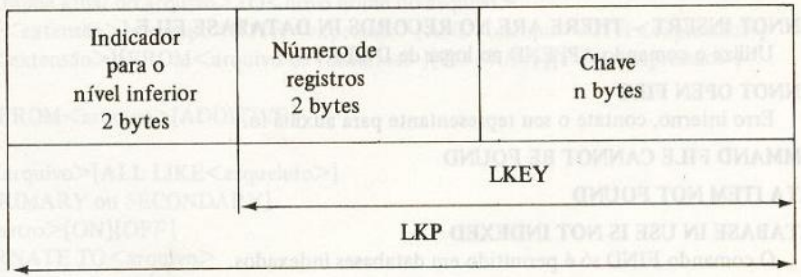
A Tabela E-2 mostra a estrutura de um arquivo de chaves.

Tabela E-2. Estrutura de um arquivo de chaves

BYTE	CONTEÚDO	SIGNIFICADO
0-1	2 bytes	Reservados
2-3	Número de 16 bits	Número do núcleo de raízes
4-5	Número de 16 bits	Número do núcleo do próximo núcleo disponível
6	Número de 8 bits	Tamanho da chave (LKEY)
7	Número de 8 bits	Tamanho da chave + indicador (chave + 2) (LKP)
8	Número de 8 bits	Número máximo de chaves por núcleo
9	Marca	0 se a chave é de caracteres, diferente de 0 se numérica
10-109	Conjunto de bytes	Expressão da chave em ASCII finalizada por OOH
110-511	Bytes	Não utilizados

NÚCLEOS:

BYTE	CONTEÚDO	SIGNIFICADO
0	Número de 8 bits	Número de chaves neste núcleo
1-511	Conjuntos	Chaves + indicadores



Notas:

1. Em um núcleo que tem n núcleos, existem n + 1 indicadores. O último indicador aponta para um núcleo de nível inferior que é MAIOR em valor do que a chave (n).
2. No último nível, os indicadores têm um valor de zero.
3. Todas as chaves são representadas ao último nível. Quando um núcleo se divide, a chave que vai para um nível superior também é mantida no último nível.

APÊNDICE F

Mensagens de Erro

O material deste apêndice é reproduzido com a permissão da Ashton-Tate.

BAD DECIMAL WIDTH FIELD
O comprimento decimal é muito longo.

BAD FILE NAME
Erro de sintaxe no nome do arquivo.

BAD NAME FIELD
O nome do campo é inválido.

BAD TYPE FIELD
O tipo do campo deve ser C, N ou L.

BAD WIDTH FIELD
Redefina o tamanho do campo de dados.

CANNOT INSERT – THERE ARE NO RECORDS IN DATABASE FILE
Utilize o comando APPEND no lugar de INSERT.

CANNOT OPEN FILE
Erro interno, contate o seu representante para auxiliá-lo.

COMMAND FILE CANNOT BE FOUND

DATA ITEM NOT FOUND

DATABASE IN USE IS NOT INDEXED
O comando FIND só é permitido em databases indexados.

DIRECTORY IS FULL
O diretório do disco não pode ter mais nenhum arquivo.

DISK IS FULL
Não há espaço vazio no disco.

END OF FILE FOUND UNEXPECTEDLY
O database que está em USE não tem um formato correto. Se todos os registros estão presentes e corretos, faça um PACK e re-indexe o database.

"FIELD" PHRASE NOT FOUND

FILE ALREADY EXISTS

FILE DOES NOT EXIST

FILE IS CURRENTLY OPEN
Emita um comando USE ou CLEAR para fechar o arquivo.

FORMAT FILE CANNOT BE OPENED

FORMAT FILE HAS NOT BEEN SET
Utilize o comando SET para abrir o arquivo de formatos correto.

ILLEGAL DATA TYPE

ILLEGAL GOTO VALUE
Valores devem estar entre 1 e 65534.

ILLEGAL VARIABLE NAME
Apenas alfanuméricos e dois pontos são permitidos como nome de campos e de variáveis.

INDEX DOES NOT MATCH DATABASE
O dBASE II verificou que a chave não corresponde ao database.

INDEX FILE CANNOT BE OPENED

JOIN ATTEMPTED TO GENERATE MORE THAN

65,534 RECORDS
A cláusula FOR permite até 65,534 registros de saída.

KEYS ARE NOT THE SAME LENGTH

MACRO IS NOT A CHARACTER STRING
Macros & macros devem ser formados por conjuntos de caracteres.

MORE THAN 5 FIELDS TO SUM

NESTING LIMIT VIOLATION EXCEEDED

NO EXPRESSION TO SUM

NO "FOR" PHRASE
Reescreva o comando com a sintaxe correta.

NO "FROM" PHRASE
Reescreva o comando com a sintaxe correta.

NO FIND
Mensagem de diagnóstico.

NON-NUMERIC EXPRESSION

NONEXISTENT FILE

"ON ' PHRASE NOT FOUND
Reescreva o comando com a sintaxe correta.

OUT OF MEMORY VARIABLES
Reduza o número ou o tamanho das variáveis em uso.

RECORD LENGTH EXCEEDS MAXIMUM SIZE (OF 1000)

RECORD NOT IN INDEX
O arquivo de índice não foi atualizado após a adição do registro.

RECORD OUT OF RANGE
O número do registro fornecido é maior do que o número de registros no database.

SORTER INTERNAL ERROR, NOTIFY SCDP
Erro interno, contate o seu representante para auxiliá-lo.

SOURCE AND DESTINATION DATA TYPES ARE DIFFERENT
Verifique se o tipo dos dados são numéricos ou em caracteres.

*****SYNTAX ERROR*****

SYNTAX ERROR IN FORMAT SPECIFICATION

SYNTAX ERROR, RE-ENTER

"TO" PHRASE NOT FOUND

TOO MANY CHARACTERS
Reduza a linha de comando.

TOO MANY FILES ARE OPEN
Um máximo de 16 arquivos são permitidos de serem abertos por vez.

TOO MANY MEMORY VARIABLES
Há um máximo de 64 variáveis.

TOO MANY RETURNS ENCOUNTERED
Provavelmente um erro na estrutura do arquivo de comandos.

"WITH" PHRASE NOT FOUND

UNASSIGNED FILE NUMBER
Erro interno, contate seu representante para auxiliá-lo.

*****UNKNOWN COMMAND**

VARIABLE CANNOT BE FOUND
É necessário criar a variável.

APÊNDICE G

Limitações e Restrições

As limitações do dBASE II são apresentadas na Tabela G-1.
O material deste apêndice é reproduzido com a permissão da Ashton-Tate.

Tabela G-1. Limitações e restrições

Número de campos por registro	32
Número de caracteres por registro	1000
Número de registros por database	65535
Número de caracteres por conjunto de caracteres	254
Precisão para campos numéricos	10 dígitos
Maior número permitido	$1{,}8 \times 10^{63}$
Menor número permitido	$1{,}0 \times 10^{-63}$
Número de variáveis na memória	64
Número de caracteres por linha de comando	254
Número de expressões no comando SUM	5
Número de caracteres no cabeçalho do REPORT	254
Número de caracteres em chaves de índices	99
Número de GETS pendentes	64
Número de arquivos abertos ao mesmo tempo	16

APÊNDICE H

Comandos e Funções

O material deste apêndice é reproduzido com a permissão da Ashon-Tate.

COMANDOS

?<expressão>[,<expressão>]
@<coordenadas>[SAY<expressão>]USING'<figura>']
[GET<variável>[PICTURE'<figura>']]
ACCEPT ["<caracteres>"] TO<variável>
APPEND [FROM<arquivo> [SDF][DELIMETED][FOR<expressão>]] ou [BLANK]
BROWSE [FIELDS<lista de campos>]
CANCEL
CHANGE FIELD<lista>[<extensão>][FOR<expressão>]
CLEAR [GETS]
CONTINUE
COPY TO<arquivo>[<extensão>][FIELD<lista>][FOR<expressão>]
[SDF][DELIMITED[WITH<delimitador>]] ou [STRUCTURE]
COUNT [<extensão>][FOR<expressão>][TO<variável>]
CREATE [<nome do arquivo>]
DELETE [<extensão>][FOR<expressão>]
DELETE FILE <arquivo>
DISPLAY [<extensão>][FOR<expressão>] [<lista de expressões>][OFF]
[FIELDS<lista de campos>]
DISPLAY STRUCTURE
DISPLAY MEMORY
DISPLAY FILES [ON<unidade de disco>][LIKE<esqueleto>]
DO<arquivo>
DO WHILE<expressão>
EDIT
EJECT

ELSE
ENDDO
ENDIF
ENDTEXT
ERASE
FIND<chave>
GO ou GOTO [RECORD], ou [TOP], ou [BOTTOM],<n>
HELP [comando]
IF<expressão>
INDEX
INDEX ON<expressão de caracteres>TO<nome do arquivo índice>
INPUT ["<caracteres>"]TO<variáveis>
INSERT [BEFORE][BLANK]
JOIN TO<arquivo>FOR<expressão>[FIELDS<lista de campos>]
LIST
LOCATE [<extensão>][FOR<expressão>]
LOOP
MODIFY STRUCTURE
MODIFY COMMAND<arquivo de comandos>
NOTE ou *
PACK
QUIT [TO<lista de comandos a nível CP/M ou arquivo .COM>]
READ [NO UPDATE]
RECALL [<extensão>][FOR<expressão>]
REINDEX
RELEASE [<lista de variáveis>[ALL[LIKE<esqueleto>]]
REMARK
RENAME<nome atual do arquivo>TO<novo nome do arquivo>
REPLACE [<extensão>]<campo>WITH<expressão>[AND<campo>WITH<expressão>]
REPORT [<extensão>][FROM<arquivo de formatos>][TO PRINT][FOR<expressão>]
RESET
RESTORE FROM<arquivo>[ADDITIVE]
RETURN
SAVE TO<arquivo>[ALL LIKE<esqueleto>]
SELECT [PRIMARY ou SECONDARY]
SET<parâmetro>[ON][OFF]
SET ALTERNATE TO<arquivo>
SET DEFAULT TO<unidade>
SET DATE TO<conjunto>
SET FORMAT TO<nome do arquivo de formato>
SET HEADING TO<conjunto>
SET INDEX TO<lista de arquivos índice>
SET MARGIN TO<n>
SKIP<+/->[<n>]
SORT ON<campo>TO<arquivo>[ASCENDING], ou [DESCENDING]
STORE<expressão>TO<variável>

SUM<campo>[<extensão>][TO<lista de variáveis>][FOR<expressão>]
TEXT
TOTAL TO<arquivo>ON<chave variável>[FIELDS<lista de campos>]
UPDATE FROM<arquivo>ON<chave variável>[ADD<lista de campos>][REPLACE
<lista de campos>]
USE<arquivo>[INDEX<nome do arquivo índice>]
WAIT [TO<variável>]

FUNÇÕES

@ ("<conjunto>", < "conjunto 2" >)
*
#
!("<conjunto de caracteres>")
$("<conjunto de caracteres>",<início>,<tamanho>)
CHR (<expressão numérica>)
DATE ( )
EOF
FILE ("<arquivo>")
INT (<expressão numérica>)
LEN ("< conjunto de caracteres >")
RANK ("<conjunto>")
STR (<expressão numérica>,<comprimento>[,<decimais>])
VAL ("<conjunto de caracteres>")
TRIM ("<conjunto de caracteres>")
TYPE (<expressão>)

Índice Analítico

Impresso na Grol
03043 Rua Martim Burchard, 246
Brás - São Paulo - SP
Fone: (011) 270-4388 (PABX)
com filmes fornecidos pelo Editor.

OUTROS LIVROS NA ÁREA:

Ciarcia — Construa o seu Próprio Microcomputador Z80
Distefano — Sistemas de Retroação e Controle
Fox/Fox — Iniciação ao Basic
Gifford — Diskguide-Guia de Referência — APPLE II
Gottfried — Programação com Basic
Hogan — CP/M — Guia do Usuário
Hurley — Programação TK-82/TK-83/TK-85/CP-200
Ingraham — Diskguide-Guia de Referência — CP/M
Osborne — A Nova Revolução Industrial — Na Era dos Computadores
Osborne — Introdução aos Microcomputadores
Osborne — Introdução aos Microprocessadores
Poole — APPLE II — Guia do Usuário
Poole — Programas Práticos em Basic
Poole — Programas Usuais em Basic
Poole — Programas Usuais em Basic — APPLE II
Poole — Programas Usuais em Basic — TRS-80
Peckham — Manual de Basic para o APPLE II
Scheid — Computadores e Programação
Scheid — Introdução à Ciência dos Computadores
Taub — Circuitos Digitais e Microprocessadores
Tremblay — Ciência dos Computadores
Verzello — Processamento de Dados, Vols. I e II
Wilson — Diskguide-Guia de Referência — Visicalc

0-07-450441-X