BUSCA EM ARQUIVOS POR MULTI-CHAVES PARA MICROPROCESSADOR

EDUARDO CARNEIRO CAMPÊLO JUNIOR

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS (M. Sc.)

Aprovada por:

Guilherme Chagas Rodrigues
(Presidente)

Nelson Maculan Filho

Ysmar Viana e Silva Filho

Luiz Antonio Carneiro da Cunha Couceiro

RIO DE JANEIRO, RJ - BRASIL
SETEMBRO DE 1980

CAMPELO JUNIOR, EDUARDO CARNEIRO

Busca em arquivos por Multi-chaves para Microprocessador. Rio de Janeiro, 1980.

viii, 225p. 29,7 cm (COPPE-UFRJ, M. Sc. , Engenharia de Sistemas e Computação, 1980).

Tese - Univ. Fed. do Rio de Janeiro - Mestrado em Engenharia de Sistemas e Computação.

1. Recuperação de Informação I. COPPE/UFRJ II. Título

À minha esposa, Ceres, pelo apoio e incentivo, e aos nossos filhos

Fabrício, Gabriela e Bruna

AGRADECIMENTOS

À Superintendência do Desenvolvimento do Nordeste - SUDENE, que propiciou meu engajamento no Programa de Engenharia de Sistemas da COPPE e ao Centro de Prestação de Serviços Técnicos de Pernambuco - CETEPE que tornou possível a realização deste trabalho.

Ao Professor Guilherme Chagas Rodrigues, pela eficaz orientação no desenvolvimento desta tese, e pelo seu empenho em possibilitar sua realização, bem como aos demais funcionários do NCE pelo apoio e compreensão, em especial a Paulo Cesar Melo e José Antônio dos Santos Borges pela colaboração no uso da linguagem PLTi e do microcomputador POTi, além de suas valiosas discussões sobre esse trabalho.

Ao colega Jackson Raimundo Mendonça pela revisão do anexo I.

E finalmente, a Marília de Dirceu Pinto de Souza, Dayse Avany Cavalcanti de Moraes, Ocelme Maria Vasconcelos de Alencar, Telma Vital Lordão e Márcia Maria Alencar de Freitas, pelos excelentes trabalhos de normatização, datilografia e desenho.

RESUMO

Esta tese apresenta um conjunto de rotinas desenvolvido para auxiliar ao projetista a implementar sistemas de recuperação de informação através de múltiplas chaves.

É feito um estudo de alternativas, considerando a uso de microprocessadores, optando-se pelo esquema de randomização para o Diretório e lista invertida para o particionamento do arquivo de dados. Foi desenvolvida uma linguagem para formulação de Pedido utilizando os operadores booleanos AND, NOT e OR para relacionar em uma sentença os diversos subconjuntos de entidades definidas pelo par chave-secundária/valor da chave-secundária.

É também apresentada a maneira de utilizar essa ferramenta, juntamente com detalhes para sua implementação.

ABSTRACT

This thesis presents a set of routines developed to help designers to implement retrieval information systems by the use of multi-keys.

An analysis of alternatives is made, considering the use of micro-processors, and concluding for a choice of the randomization scheeme for the directory and the inverted list for the data files partitioning. A language was developed for Query formulation, using the boolean operators AND, NOT and OR in order to relate in a statament the diferent entity subsets defined by the secondary-key/secondary-key value pair.

It is also shown the way to use this tool including implementation details.

SUMÁRIO

1. INTRODUÇÃO

1. INTRODUÇÃO

Este documento diz respeito a especificação, projetamento e implementação de um Sistema de Recuperação de Informações através da formulação de um Pedido (QUERY), o qual relaciona conjuntos constituídos por entidades que possuam os mesmos atributos. Considerando o estágio da indústria brasileira de computadores, protegida pela política do Governo Federal em proibir a importação de equipamentos que tenha similar nacional, bem como desestimular a importação daqueles de porte além dos nacionais, julgou-se oportuno avaliar a possibilidade de um software dessa natureza voltado para microprocessadores.

Nesta introdução define-se mais adiante o objetivo e restrições ao desenvolvimento desse trabalho.

No Capítulo (II) procurou-se definir as bases das discussões posteriores, bem como fazer uma revisão do assunto.

No Capítulo (III) propõe-se um modelo para o processo conversacional com recuperação por multi-chaves, o qual procurou-se detalhar nos Capítulos IV e V.

No Capítulo VI comenta-se detalhes da implementação desse projeto no equipamento POTI, desenvolvido pelo Laboratório Digital do NCE, e no último Capítulo se avalia os resultados alcançados em função daquilo que se perseguia.

I.1 OBJETIVO

Este trabalho tem como objetivo principal desenvolver um "sistema de armazenamento e recuperação de informação" para equipamento eletrônico de processamento de dados de pequeno porte, que seja capaz de possibilitar o seu uso como instrumento auxiliar à tomada de decisão. Ou seja, através de um processo interativo com seu usuário final, recuperar entre as informações armazenadas relativas a um universo de entidades*, aquelas desejadas pertinentes ao subconjunto do mesmo universo, que satisfazem às condições expressas por uma sentença (ou expressão) lógica. Tal expressão lógica representa um pedido (query), devendo ser formulado segundo as regras de sintax da linguagem definida em (II.4). Para melhor clareza, pode-se tomar como exemplo de um pedido a tradução da seguinte questão: quais os produtos farmacêuticos disponíveis em um hospital que contenham a substância "A" ou a substância "B" além da substância "C"?. Em resposta a tal pedido, deveriam ser relacionados todos aqueles produtos em cuja composição encontram-se as substâncias (A e C) ou as substâncias (B e C), constituindo um subconjunto do universo dos produtos sobre os quais se mantém informações armazenadas em suporte magnético. Vale observar que tal subconjunto pode ser vazio ou se constituir de um, ou mesmo vários elementos.

É desejável, por um lado, que o processo interativo com o usuário final se dê de modo claro, acarretando aprendizagem rápida, como por outro lado, que possibilite uma redução significativa do esforço de análise e programação nas fases de implantação da aplicação e de sua inevitável manutenção, reflexo das próprias mutações de seu ambiente. Sem dúvida tais

---

* - Dúvidas quanto ao significado de termos aqui utilizados podem ser dirimidas no Capítulo II.

fatores, aliados aos custos relativamente baixos dos equipamentos aqui em pauta, são fundamentais à viabilização de seu emprego em maior escala, visto tornar-se acessível a um maior número de indivíduos e organizações, além de atender a mais diversificadas necessidades.

1.2 AMBIENTE

O "sistema de armazenamento e recuperação de informação" aqui proposto, foi desenvolvido e implementado utilizando-se o Terminal Inteligente (Ti) construído pelo Núcleo de Computação Eletrônica - NCE da Universidade Federal do Rio de Janeiro. O Ti, que recebeu a denominação de "POTi", é um microcomputador, que apesar de seu caracter interativo permite executar tarefas em "Batch", e encontra-se com as seguintes características:

- CPU - microprocessador INTEL-8080 de 8 (oito) bits, com 2,5 segundos de ciclo;

- Memória - 4 (quatro) Kbytes em ROM reservados para algumas funções do sistema operacional (SOCO) tais como partida do sistema, rotinas comuns a várias rotinas de entrada/saída, etc., bem como reservados ainda para o interpretador da PLTi, sua linguagem de alto nível; os programas de aplicação rodam em memória RAM atualmente limitada a 32 (trinta e dois) Kbytes, limite este que em breve será estendido a 64 (sessenta e quatro) Kbytes;

- Periféricos - O POTi pode controlar uma linha de comunicação (atualmente em teste como terminal RJE do Burroughs - 6700 do NCE), e mais leitora de cartão, impressora, vídeo/teclado, fita cassete, disco, disquete ou qualquer outro periférico compatível com a linha PDP (CURA[14]). Sendo de especial interesse para implementações na forma a que se propõe este projeto, convém citar que pode suportar até 8 (oito) unidades de disco para uma unidade de contro

le, e que cada disco (disk-pack) contêm 200 (duzentos) cilindros de 2 (duas) trilhas cada, 24 (vinte e quatro) setores/cilindro e 512 bytes/setor. A organização do disco é a seguinte: cilindro zero - programa de partida fria, tabelas do disco; cilindro-1 - diretório; cilindro 199 - cópia do cilindro zero; e do cilindro 2 a 198 área disponível.

- Sistema Operacional (SOCO) - Vale mencionar pela relevância em relação à implementação deste trabalho, que o SOCO é um sistema operacional residente em disco, limitado à mono-programação, interativo, permitindo a substituição de periférico em tempo de execução, bem como a concatenação de arquivos na mesma ou em outra unidade de disco, ou mesmo atribuir a um arquivo o conceito de arquivo vazio ou periférico nulo (Dummy), para o qual quando o programa tenta executar um comando de entrada/saída lhe é informado fim de arquivo. A configuração mínima exigida é de 16 (dezesseis) Kbytes de memória. Quando se tratando de arquivo em disco magnético, segundo a declaração do mesmo arquivo, pode ser designada rotina de entrada/saída para utilizar buffer individual ou compartilhado com outros arquivos.

  Essa característica se torna importante já que as rotinas de entrada/saída em disco se encarregam de realizar a blocagem e deblocagem automática dos registros lógicos em um único bloco por setor, ou seja, as leituras e gravações são executadas para um setor inteiro, sobre o buffer monitorado pelo SOCO. Assim, a cada comando de leitura ou gravação um só registro é transferido entre esse buffer e o buffer do usuário. Consequentemente, livra o programador de tal preocupação e possibilita reduzir o número de acessos físicos a disco, quando processando-se registros consecutivos (serialmente). Caso o buffer monitorado esteja sendo compartilhado o mesmo raciocínio é válido no período em que não haja interferência de operações de entrada/saída relativas a outros arquivos. É permitido o processamento aleatório através do endereçamento relativo ao

i-ésimo registro de um arquivo em disco, pela execução de um comando de "SEEK" a tal número de ordem dentro do arquivo, considerando que o endereço do primeiro registro é zero, para o mesmo arquivo.

Seu editor de referências externas (REFEX) tem a finalidade de reunir em um só módulo um programa chamado principal e todas as rotinas chamadas direta ou indiretamente por esse programa. O REFEX é dotado da capacidade de criar módulos executáveis em "overlay", bastando para isso o usuário especificar a estrutura desejada através de comandos apropriados.

Informações complementares podem ser encontradas nas obras de SILVA[13], FAISOL[11] e em especial nas de RODRIGUES[15] e CURA[14].

PLTi - É uma linguagem de alto nível especialmente desenvolvida para microprocessador, que foi baseada na linguagem PL/M descrita em INTEL[17]. É de fácil aprendizagem e suas características principais são:

. dois tipos de variáveis:
  BYTES - com intervalo de definição de $0 \leq$ byte ≤ 255, e em hexadecimal de #0 a #FF; e

  ADDRESS - com intervalo de definição de $-32.768 \leq$ address ≤ 32.767, e em hexadecimal de #0 a #FFFF;

. os caracteres são representados em ASCII;

. aceita variáveis subscritas de uma dimensão de zero a ≤ 32.767, podendo o subscritor ser expressão;

. permite estruturas formadas por identificador grupo, composto de variáveis simples ou grupo de nível inferior (a exemplo do COBOL, PL/1, etc.);

. variáveis "based" para endereçamento indireto;

. definição hierarquizada (em blocos) de variáveis e rótulos, definição de variáveis globais;

. comandos;

controle de malhas: DO, DO TO, e DO WHILE;
DO CASE;
IF THEN ELSE;
PROCEDURE, CALL e RETURN;

. operador aritmético de "Shift";

. facilidades de depuração (TRACE);

. permite ligação com rotinas em ASSEMBLER;

. definição de arquivos;

. permite compilação de subrotinas em separado;

. comandos de E/S ao nível de READ, WRITE, REWRITE, SEEK e etc.

A PLTi gera códigos interpretáveis fazendo com que os programas executáveis sejam extremamente compactos. Ressalte-se que o próprio SOCO foi desenvolvido em PLTi. (instruções sobre PLTi encontram-se em RODRIGUES[15] e MELLO[16]).

- Biblioteca - a biblioteca reside em disco e contém programas e arquivos.

- Programas de Uso Geral

. Editor de Textos - cria ou altera programas fonte tornando o sistema independente de leitora de cartões. Entre outras funções ele permite criar arquivos novos, corrigir, inserir ou apagar uma ou várias linhas, etc.

TICOP - programa que pode copiar arquivos de diversos periféricos para o mesmo, em caso de disco, ou para outros periféricos com opção de alterar o tamanho do registro, e mais diversas funções.

1.3 RESTRIÇÕES

Como restrições fortes ao desenvolvimento e implementação do "sistema de armazenamento e recuperação de informações" aqui em estudo, destacam-se o pouco espaço de memória principal disponível, a capacidade de armazenamento em suporte de acesso direto (disco magnético), tamanho máximo de registro igual ao setor (512 byte), número de registros por arquivo limitado a 32.768, visto que uma variável "address" pode assumir valores positivos de zero a 32.767. A versão do POTi utilizada para o desenvolvimento, possuía no momento 16 (dezesseis)Kbytes de memória principal (RAM), 2 (duas) unidades de disco magnético, 1 (uma) leitora de cartão, uma unidade teclado/vídeo, e uma impressora de 132 (cento e trinta e duas) barras de impressão. Quanto ao tempo de resposta a um "Pedido" não vê-se porque estabelecer um limite padrão como tolerável, apesar de sempre se desejar que seja o menor possível. Deve ser observado que tal tempo de resposta para uma mesma aplicação, pode variar bastante em função da formulação do "Pedido", envolvendo maior ou menor número de operações. O limite considerado tolerável para o tempo de resposta a um pedido, deve ser objeto de avaliação e definição na fase de análise de cada aplicação (problema infológico abordado por SUNDGREN[3], DUEIRE[18] e LANGEFORS[19]).

DUEIRE[18] ressalta que "o valor da informação esvazia-se no tempo, existindo patamares em que o valor continua constante, tendo, logo após, uma perda acentuada". Numa análise de custo/benefício da informação, é necessário considerar que o seu valor também é função de sua qualidade, entendendo-se por qualidade da informação, sua adequalidade e capacidade de diminuir ou eliminar o estado de incerteza na tomada de decisão

(de tornar o sistema mais estável), associando-se a ela um conceito de valor.

Por outro lado, enquanto aumentos iguais e progressivos na qualidade da informação acarretam valores incrementais decrescentes no valor, esses mesmos aumentos de qualidade trazem incrementos crescentes no seu custo de obtenção. Um mesmo benefício de uma informação se pode obter com diversos custos, variando-se a tecnologia usada no sistema de informação. Uma tecnologia pode ser desvantajosa em relação a outra em um determinado nível de qualidade da informação, mas ser vantajosa noutro nível, ora acarretando custos mais elevados ora mais baixos que aquela. Assim, ao fixar a qualidade da informação necessária à tomada de decisão, seja no nível estratégico, tático ou operacional da organização, tem-se que definir o tempo e custo de obtenção da mesma, o que pode ou não viabilizar o emprego de determinada tecnologia.

II - DEFINIÇÕES E REVISÃO BIBLIOGRÁFICA

II. DEFINIÇÕES E REVISÃO BIBLIOGRÁFICA

Objetivando dotar a explanação aqui desenvolvida de maior clareza e precisão, julgou-se necessário definir a conotação dada a alguns dos termos empregados, sem se deixar cair nas minúncias, por considerar que os demais são de uso geral entre aqueles que lidam com o assunto em pauta. Por outro lado, ao se discorrer sobre os conhecimentos fundamentais à este trabalho, subsidiado por extensa pesquisa bibliográfica, não houve qualquer preocupação em exaurir o assunto, mas explorá-lo de modo a atender satisfatoriamente os objetivos preconizados.

II.1 DEFINIÇÃO DE TERMOS BÁSICOS

- Entidades e Atributos

Quando tratando com informações é necessário se ter em mente a distinção entre os três "domínios" ou nível de abstração em que pode estar se dando o enfoque:

a) fenômenos do mundo real;

b) informações acerca de fenômenos do mundo real; e

c) representação das informações acerca de fenômenos do mundo real.

Apesar do escopo deste trabalho estar centrado no domínio (c), faz-se necessária rápida incursão aos anteriores, com vistas a uma melhor precisão de conceituação.

Denomina-se "sistema objeto", a parcela do mundo real a respeito do qual deseja-se armazenar informações em uma base-de-dados.

Ao especificar um modelo de um "sistema objeto", chama-se intuitivamente de ENTIDADE (ou objeto) a tudo aquilo, pertencente a parcela (ou fatia) do mundo real em estudo, que se julga interessante e do qual se deseja adquirir informação a respeito.

Entidade pode ou não ser um ente físico (ou um objeto tangível). Empresa, departamento, profissão e acidente de trânsito são tão bons candidatos a entidade como pessoa, casa e veículo a motor.

Sempre em processamento de dados se está relacionado com coleções de similares entidades, tais como empregados, e deseja-se registrar informações acerca das mesmas relevantes propriedades de cada uma dessas entidades. A tais coleções de similares entidades se faz referência como um "conjunto de entidades". Pode ser tão relevante querer obter informações sobre propriedades de uma entidade quanto sobre suas relações para com outras entidades.

As propriedades e características de entidades que se deseja registrar denominam-se "ATRIBUTO".

Como atributos da entidade "empregado", por exemplo, pode-se ter nome, função, número-de-matrícula enquanto que JOSÉ, VENDEDOR, 1410 são denominados de "VALOR DE ATRIBUTO", respectivamente dos atributos acima mencionados, para um determinado empregado.

Atributo Identificador, ou apenas identificador, denomina-se a todo atributo ou conjunto de atributos que, dentro de seu domínio de valores, tem a propriedade de assumir um valor único para cada entidade do universo de entidades. Exemplos:

a) o número-matrícula de um empregado em uma empresa, ou seu número do CIC no Cadastro de Pessoa Física (CPF) da Receita Federal - MF; e

b) já a reserva para um passageiro em uma linha aérea exige conjunção da data com número do vôo, pois em diversos dias o número do vôo é o mesmo, e as companhias comumente têm mais de um vôo no mesmo dia para o mesmo par origem-destino.

Quanto a representação dos dados (ou valores assumidos pelos atributos) acerca de cada entidade do universo em estudo, torna-se necessário para o âmbito deste trabalho definir:

. Item-de-dados - ou "dado elementar", é o menor e indivisível dado ao qual é possível se fazer uma referência simbólica, através de um nome a ele atribuído, estando o mesmo associado a um relevante atributo e seu domínio, que qualifica uma relevante entidade para o sistema objeto. A representação um item-de-dados sobre um suporte físico é contida em um espaço denominado de campo, segundo forma de codificação predeterminada (alfanumérico, sucessão de bit "bit strings", etc.).

. Registro - ou "tupla", é um agregado lógico de campos, sendo a maneira mais comum de associar atributos de uma mesma entidade armazenando-os em locais contíguos e em uma sequência fixada. Representam a unidade básica no que tange ao manuseio de arquivos.

. Arquivo - consiste de um conjunto de registros, que possuem as mesmas características. A localização desses registros dentro do espaço reservado ao arquivo, seguem algumas regras ditadas pelos procedimentos de manutenção do mesmo, bem como o de recuperação de informação.

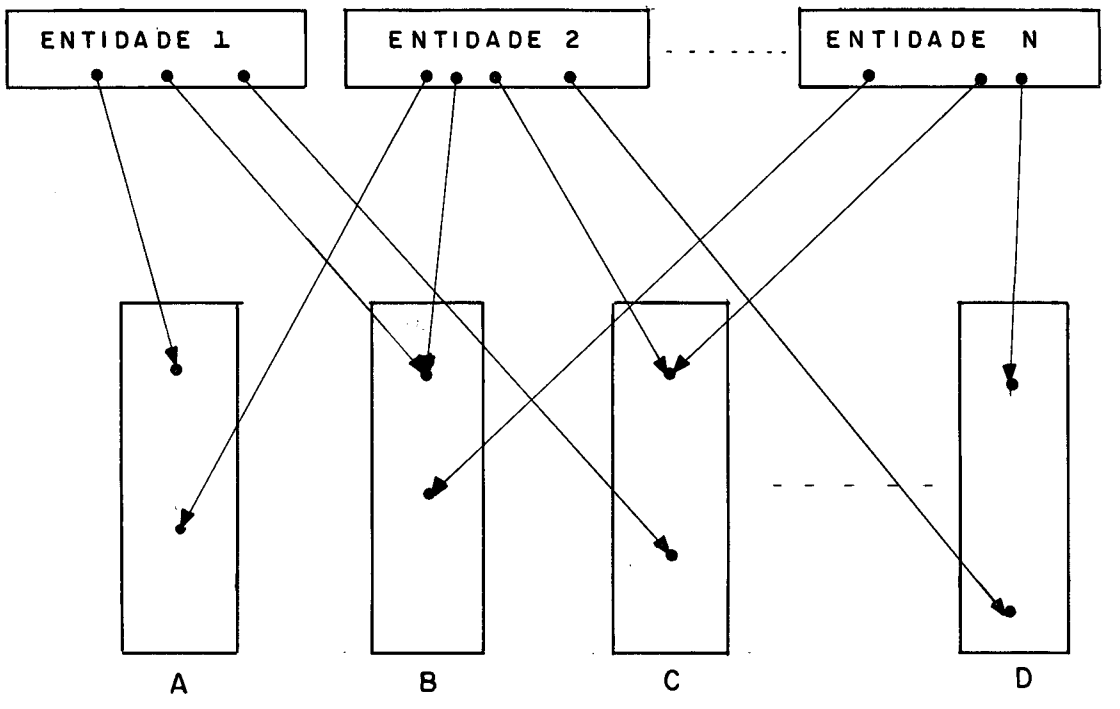
. Chave - é todo campo utilizado por procedimento de seleção e recuperação de registro, cujo valor de seu conteúdo satisfaz a determinada condição (chave = K, onde K é o argumento de pesquisa). A quantidade de registros recuperados depende da condição e da classe de atributos que tal campo representa.

. Chave Primária - é o campo ou conjunto de campos destinados a conter os valores dentro do domínio de um "atributo identificador", e que por possuir a propriedade de identificar unicamente um registro em um arquivo, é utilizado pela "estratégia de acesso" a registros individuais desse mesmo arquivo. Assim, é de grande importância por ser usado pelo computador na localização de registro ou tupla, por meio de um índice ou algoritmo de endereçamento.

. Chave Secundária - são aquelas chaves não primárias. Em geral não identificam unicamente um registro ou tupla, mas identificam todos aqueles relativos a entidades que possuem uma certa propriedade. Por exemplo, o campo correspondente ao atributo "COR", pode ser usado como chave secundária caso seja desejado identificar todas as entidades de cor "verde", ou aqueles registros que contém o valor "verde" no domínio "COR".

Quanto a relação entre entidades e atributos podemos ter:

VÁRIOS ATRIBUTOS - PARA - VÁRIAS ENTIDADES



Esta é a ocorrência mais frequente, visto que o interesse em um sistema de informação sobre as entidades envolvidas, em geral se manifesta através de um conjunto de atributos relevantes aos objetivos deste sistema, a respeito dos quais se deseja obter, armazenar e recuperar os seus valores em determinados instantes ou períodos de tempo.

II.2 CLASSIFICAÇÃO DE "INQUIRIÇÕES OU PEDIDOS" (QUERIES) DO USUÁRIO

Passamos a tecer comentários sobre formas e propriedades de "inquirições ou pedidos" formulados por um usuário que interage com um sistema de informação automatizado, visando obter respostas que possam subsidiá-lo em um processo de tomada de decisão. Esses pedidos encerram as "condições" que devem ser satisfeitas ao se recuperar um subconjunto de registros, dentre aqueles que constituem o universo armazenado. É muito frequente entre as aplicações ditas comerciais se encontrar arquivos com uma única chave por registro. Entretanto, não raros são aqueles que requerem registros com mais de uma chave, o que torna a organização do arquivo mais complexa. Arquivos com múltiplas chaves são comumente encontrados em sistemas de informações nos quais uma variedade de pedidos pode ser feita, acerca de diferentes aspectos da informação.

CARDENAS[22] classifica os "pedidos" (queries) de acordo com a sua complexidade, sendo do interesse deste trabalho as seguintes considerações:

1 - Condição Atômica (A) - é uma condição que tem a forma

$$\text{ATRIBUTO} \left\{ \begin{bmatrix} < \\ = \\ \neq \\ > \end{bmatrix} \right. \text{VALOR}$$

2 - Condição de Grupo (I) - é uma disjunção de condições atômicas, A1 OR A_2 OR ... OR A_L tal que cada A_i refere-se a um mesmo atributo. O número de "Condições Atômicas" por "Condição de Grupo" (I) é representado por (CAI). Exemplo:

IDADE = 20 OR IDADE = 21, onde (CAI = 2)

3 - Condição de Registro (R) - é uma conjunção de "Condições de Grupo" em que se utiliza os operadores AND e NOT I_1 AND I_2 AND ... NOT I_m tal que I_j reflete um distinto atributo.

O número de "Condições-de-Grupo" por "Condição-de-Registro" (R) é representado por (CGR). Exemplo:

(IDADE > 20) AND (SEXO = FEMININO), onde CGR = 2

4 - Condição de Inquirição (Q) - é uma disjunção de "condições de Registro", R_1 OR R_2 OR ... OR R_n. O número de "Condições de Registro" por "Condição de Inquirição" (Q) é representado por (CRQ). Exemplo:

(IDADE > 18) AND (SEXO = FEMININO) OR

(IDADE > 20) AND (SEXO = MASCULINO) AND

(NÍVEL = SENIOR OR NÍVEL = GRADUADO), onde (RCQ = 2).

Essas definições permitem que uma "Condição-Atômica" seja também uma "Condição-de-Grupo", uma "Condição-de-Registro" e uma "Condição-de-Inquirição".

Elas estão definidas de modo a permitir que o usuário do sistema especifique qual o registro (ou entidade) ele está interessado. Assim, praticamente todos os tipos de "inquirições" podem ser formuladas através dos formatos acima. A complexidade do requerimento de recuperação de informação afetam o tempo médio de acesso para recuperar os registros que satisfazem as condições especificadas pela "inquirição", e isto, por

seu turno, afeta a escolha da organização de arquivos, assunto tratado na seção II.3, próxima.

De acordo com o grau de complexidade, chamaremos a inquirição (A) de "forma simples" e as demais de "forma complexa".

Para o caso particular em que o atributo se tratar de um "identificador de entidade", e a condição for IDENTIFICADOR = VALOR, tal condição representa a formulação da questão: "Qual os valores dos atributos, ou parte dos atributos, da entidade cujo identificador tem determinado valor, único dentre aqueles pertencentes a um mesmo universo de entidades?".

Exemplo - do empregado de número-matrícula = 1070 deseja-se saber, nome, função, salário, data admissão e número de dependentes.

Esta é a forma mais comum de inquirição.

Em resposta as demais "formas simples", ou qualquer das "formas complexas", obtém-se uma lista do subconjunto de entidades pertencente a um mesmo universo, as quais satisfazem à inquirição.

Observe-se que o comprimento da mencionada lista diminui rapidamente, a medida que o grau de complexidade da inquirição cresce.

II.3 ORGANIZAÇÃO DE ARQUIVO PARA RECUPERAÇÃO POR MÚLTIPLAS - CHAVES

Está implícito que o sistema requer resposta em tempo-real pois ele opera em um modo interativo (on-line). Com relação ao hardware, isto implica na utilização de dispositivos de armazenamento secundário que permitam rápido acesso aleatório, enquanto requer ao software arquivo de dados particionado de modo a tornar possível que subconjuntos relativamente pequenos desse arquivo sejam de imediato acesso, pela aplicação de adicionais procedimentos habilitados a esse fim. O determinante quantitativo de resposta em tempo-real é o tamanho da partição do arquivo e a estratégia para recuperação de combinações especificadas dessas partições. Por outro lado, a flexibilidade do sistema é largamente determinada pelo número de partições. A organização de arquivo que é usada para implementar tais partições é denominada de "estruturação em lista". Várias técnicas de estruturações em listas são descritas e avaliadas em LEFKOVITZ[4 e 5], KNUTH[1], MARTIN[6] e CÁRDENAS[21 e 22]. Basicamente, a resolução de um pedido de modo interativo pode ser vista como um processo em duas etapas:

- Etapa 1 - envolve a codificação ou tradução do "pedido" para endereços de listas* de endereço (ou chaves primárias). Tal pedido constitui uma "expressão lógica", sendo formulado em uma linguagem, que pode ser próxima de uma linguagem natural, ou por atributos codificados relacionados por operadores lógicos, sendo os mais frequentes os Booleanos "AND", "OR" e "NOT"; e

- Etapa 2 - consiste na execução da busca por acesso randômico ao "arquivo de dados" baseada nas listas de endereços (ou chaves primárias) definidas pela etapa anterior.

---

* - No sistema implementado utilizou-se lista de chave primária.

De acordo com a estratégia de particionamento do arquivo de dados, pode ocorrer na etapa 1 ou 2 o reconhecimento do subconjunto de endereços (ou chaves primárias) referentes aos registros do "arquivo de dados" que representam as entidades cujos atributos satisfazem as condições expressas pelo pedido.

Em LEFKOVITZ[5] encontra-se uma classificação hierárquica das técnicas de organização de arquivo que são usadas para desenvolver essas duas funções, cuja finalidade é auxiliar ao projetista a realizar uma seleção baseada nas propriedades requeridas pelo decodificador, claramente diferenciadas. Na primeira etapa os endereços das listas são obtidos consultando-se para cada atributo o "Diretório de Atributos" (ou índice secundário), o qual se constitui em um arquivo segmentado por atributo, cujas entradas contém endereços de início de "lista de endereços" daqueles registros que possuem o mesmo valor para um mesmo atributo (campo).

- Técnicas de Organização de Diretório

As técnicas que dizem respeito ao "Diretório de Atributos" estão divididas em duas classes gerais, uma chamada de "decodificação por randomização (hash coding)" e a outra de "decodificação por estrutura em árvore". A técnica de randomização aqui tratada, consiste em transformar o valor de uma chave, representado por uma sucessão (String) de caracteres de comprimento fixo ou variável, em um inteiro pertencente a um domínio especificado. Usualmente essa transformação é realizada por meio de uma "função de mapeamento de bit". Desde que a função de mapeamento pode gerar colisão entre chaves (ambiguidade - $f(x) = f(y)$), deve-se procurar utilizar artifícios que a tornem a mais uniforme possível. A randomização apesar de permitir diversas variações, funcionalmente é classificada em um único grupo.

A decodificação por árvore possui várias ramificações funcionais, sendo o esquema mais adotado aquele no qual é fixado o comprimento das chaves, pois acarreta programas de decodifica

ção algo mais simples para escrever e mais rápido em execução. Existem dois métodos gerais para fixar comprimento de chaves que possuem dimensões quaisquer: um é por simples truncamento da chave para um determinado número de caracteres; o outro é pela aplicação da técnica de randomização da chave a fim de transformar em valor pertencente a um domínio de inteiros, que é representado por um número fixo de bits.

Em geral, a decodificação por randomização requer menos espaço de armazenamento em dispositivo de acesso direto que a por árvore, embora em algumas aplicações a diferença seja pouco representativa.

Por outro lado, a árvore tem a propriedade de permitir a recuperação para intervalo de valores no domínio de um atributo, a qual é necessária para a implementação dos operadores relacionais "maior do que", "menor do que", etc., enquanto a randomização não tem essa propriedade, a não ser em casos particulares em que se possa introduzir algum artifício, como deslocamento para um valor base ou fixar faixas dentro do domínio.

As velocidades de decodificação dos dois métodos são equivalentes, sob diversas circunstâncias. Tanto a técnica por randomização como por árvore de chave de comprimento fixado são passíveis a produzir ambíguas decodificações, exigindo sua resolução por um subsequente exame da chave inteira, em outro subdiretório (onerando o uso de memória) ou no arquivo de dados em tempo de busca (diminuindo a sua eficiência). Entretanto, são considerados atrativos pela simplicidade e porque o custo em memória adicional e perda de eficiência nem sempre são significantes. Com a adoção de árvore de comprimento de chave fixado, fica prejudicada a propriedade que permite a recuperação por intervalo de valores, a não ser em casos especiais de truncamento.

O "Diretório de Atributos" pode ser mantido na memória principal ou em dispositivo de armazenamento secundário de acesso

direto, dependendo do seu tamanho, da disponibilidade de memória principal e requerimentos de velocidade de decodificação.

A seleção do esquema para um "diretório decodificador" recai no balanceamento (tradeoff) entre os fatores quantitativos mais relevantes, que em geral são velocidade de decodificação, requerimento de memória principal e secundária em dispositivo de acesso direto, e complexidade de programação, enquanto que a capacidade de realizar, ou não, busca para intervalos de valores pode ser considerada como o fator qualitativo de forte influência, o qual depende unicamente de requerimentos da aplicação. Como atenuante à adoção de um esquema que não satisfaça às mais rígidas exigências desse fator qualitativo, é frequentemente utilizada a estratégia de se realizar uma seleção após a recuperação dos registros de dados, sobre os quais são aplicados os operadores relacionais como requeridos ("maior do que", "menor do que", etc.).

- Técnicas de Particionamento do Arquivo de Dados

A recuperação "em linha" (on-line), em resposta a "pedidos" formulados segundo "expressões lógicas", como é aqui abordado, tem como requerimento básico para estruturação de "arquivo de dados" o seu particionamento lógico. Isto é, dado que cada registro do arquivo deve ser armazenado em um único local na memória secundária de acesso direto, e que é desejável ter a capacidade de acessar esses registros com base em qualquer de suas chaves secundárias juntamente com outros registros, todos os quais contendo o mesmo valor para a mesma chave, a organização lógica deve isso impor:

1 - sobre o programa que armazena, recupera e atualiza esses registros;

2 - com relação ao local do registro na memória secundária; e

3 - sobre estrutura de dados nos registros.

Basicamente, o particionamento de arquivo com vistas a esse específico problema, encontra solução nas estruturas em lista segundo os dois esquemas denominados de Miltilista e Lista Invertida. Para cada chave secundária*, as estruturas de lista se constituem do encadeamento dos registros que possuem mesmo valor do atributo cujo campo tenha sido escolhido como chave secundária. O endereço do início de cada uma dessas listas é mantido em uma entrada do "Diretório de Atributos" ou "Índice Secundário".

No esquema de Multilista as listas são construídas sobre os próprios "registros de dados". Associado a cada chave secundária é acrescentado um campo de endereçamento (link address) onde são mantidos os endereços relativos ou simbólicos dos registros sucessores (algumas vezes também dos registros antecessores), explicitando-se as relações entre aqueles cujo campo referente à particular chave secundária assume mesmo valor. A figura (II.1) exemplifica um esquema em multilista, tendo-se um arquivo de dados com 10 (dez) registros, particionado por duas chaves secundárias, cujos índices secundários constituem o "Diretório de Atributos". Cada entrada no índice de uma particular chave secundária corresponde a um valor assumido pelo atributo respectivo, dentro de seu domínio.

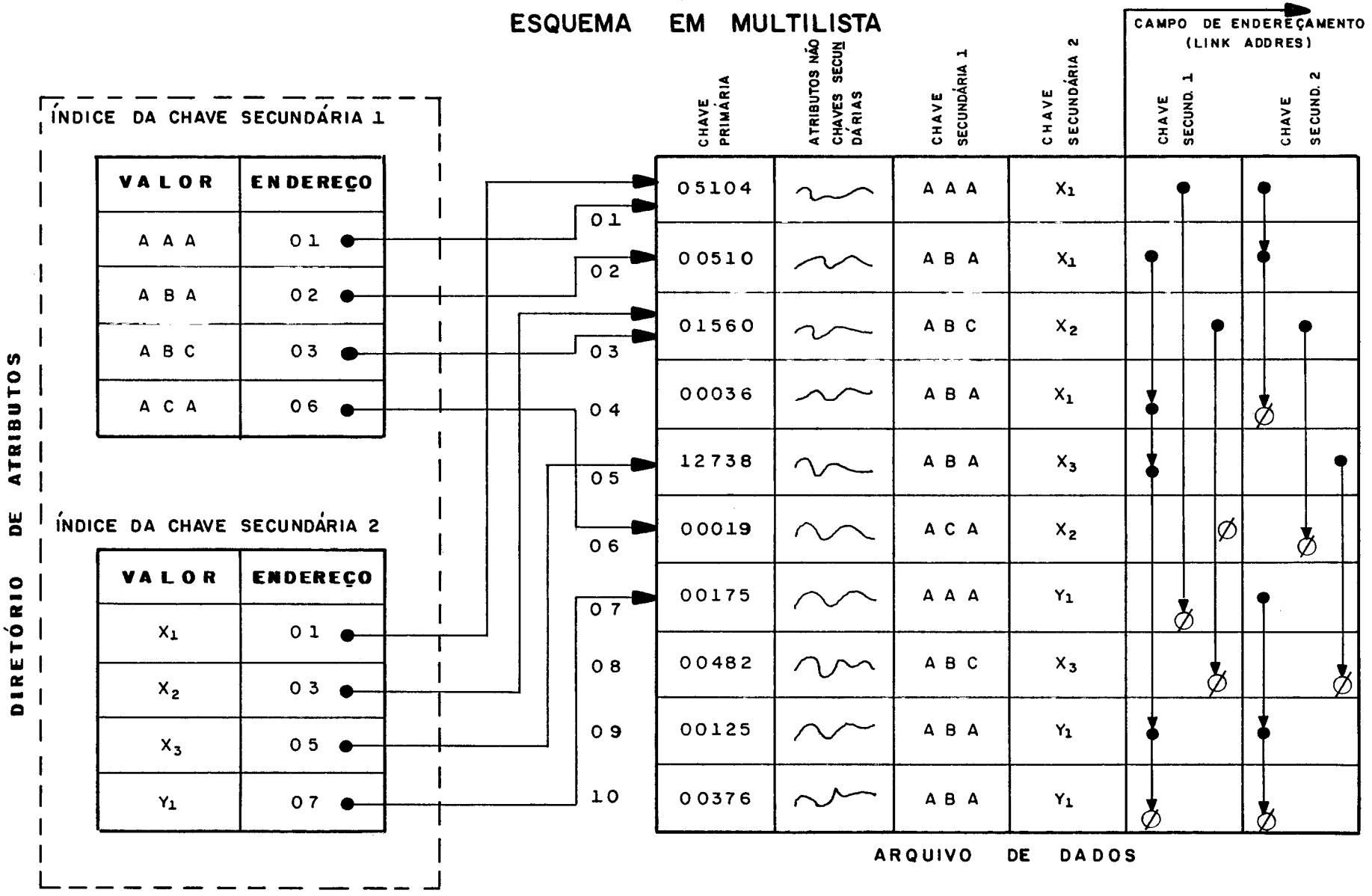
Caso fosse desejado conhecer quais entidades que possuem as propriedades encerradas na seguinte expressão lógica (chave secundária 1 = ABA) AND (chave secundária 2 = X1), obter-se-ia como resultado da primeira etapa do processo os endereços iniciais das listas: 02 para a primeira "condição atômica" e 01 para a segunda "condição atômica". Para se concluir quais registros estão na interseção das duas listas, seria necessário recuperar pelo menos todos de uma delas (a menor naturalmente) e na memória verificar quais deles pertencem a outra lista, ou seja, quais satisfazem a "condição atômica" desta.

---

* - Chaves secundárias, chama-se aqui àqueles campos que possuem índices secundário próprio.

FIG. II - 1

ESQUEMA EM MULTILISTA



- Ø SIGNIFICA FIM DE LISTA

No exemplo, apenas as duas entidades de chave primária 00510 e 00036 satisfazem as condições do pedido. Há situações na prática que podem acarretar grandes listas, especialmente se tratando de união (operador OR) de várias delas, o que pode exigir um grande número de acessos à memória secundária.

No esquema de Lista Invertida não são acrescidos campos de endereçamento nos registros do arquivo de dados, mas o particionamento lógico é estabelecido pela geração de informações redundantes mantidas em um outro arquivo, chamado "arquivo invertido", cuja finalidade é acelerar o processo de recuperação por chaves secundárias.

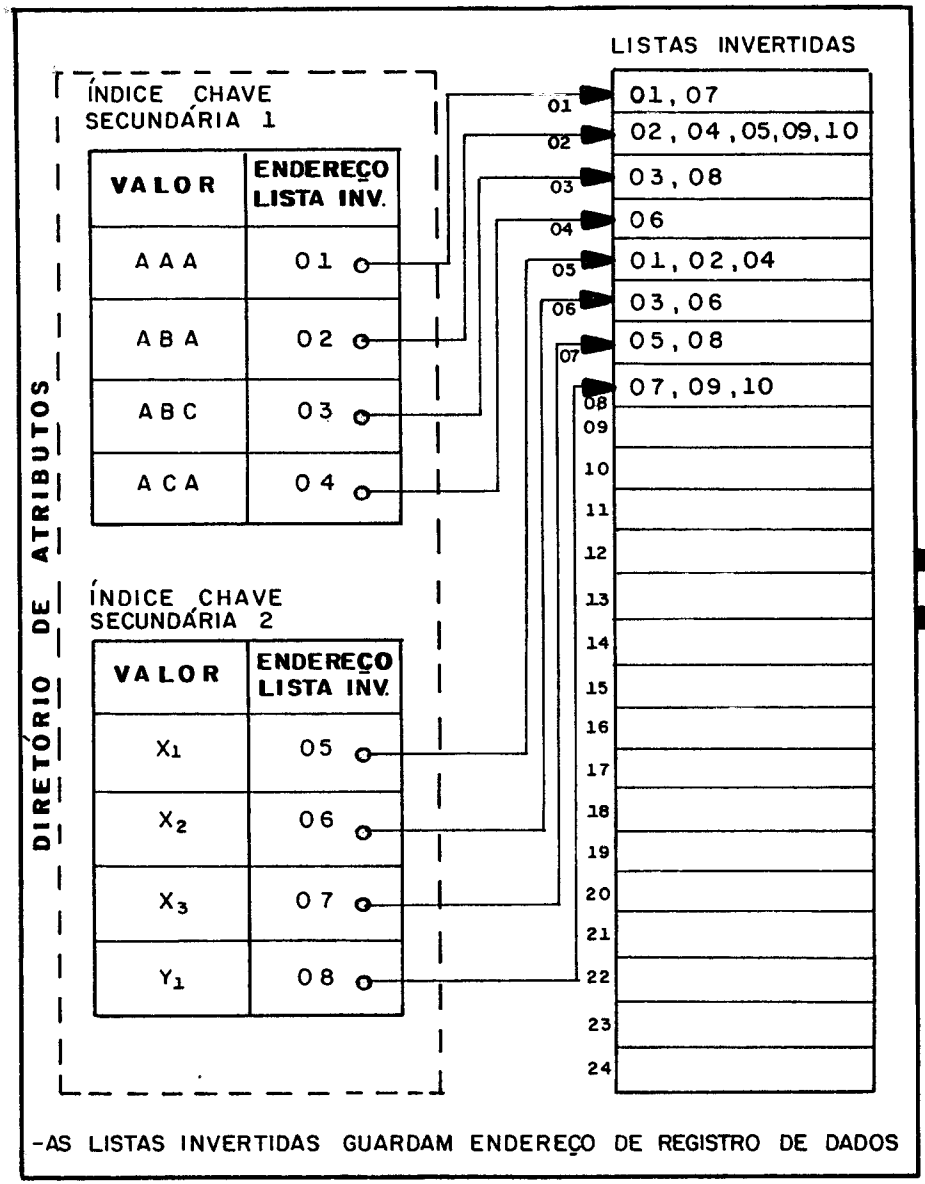
Frequentemente em arquivos não invertidos, as relações entre atributos de uma mesma entidade são representadas pela localização de seus campos em um mesmo registro, para o qual o acesso se processa utilizando um único campo chamado chave primária. Já os arquivos invertidos são formados pela aglomeração de listas que identificam aqueles registros de dados, os quais, para uma particular chave secundária, possuem o mesmo valor. Portanto, para cada registro de dados é mantida uma referência em alguma lista de cada chave secundária. Consequentemente, o número de referências para cada "registro de dados" é o mesmo de suas chaves secundárias. Cada referência é mantida em uma entrada (campo de endereçamento) da lista pertinente, contendo o endereço do registro de dados, o qual pode ser endereço físico, relativo ao arquivo ou simbólico* (chave primária). Esta lista se compõe dos endereços de todos os registros que possuem mesmo valor para o campo relativo a mesma chave secundária. Por sua vez, o "Diretório de Atributos" mantém no índice de chave secundária, uma entrada apontando ao início de cada uma de suas respectivas listas.

---

* - O endereçamento simbólico pode onerar o uso de memória secundária e o tempo de processamento por exigir uma função para transformar chave em endereço, porém pode salvar o custo de manutenção das listas quando de uma reorganização do arquivo de dados.

FIG. II - 2

ESQUEMA EM LISTA INVERTIDA



	CHAVE PRIMÁRIA	ATRIBUTOS NÃO CHAVES SECUNDÁRIAS	CHAVE SECUNDÁRIA 1	CHAVE SECUNDÁRIA 2
01	05104		AAA	X_1
02	00510		ABA	X_1
03	01560		ABC	X_2
04	00036		ABA	X_1
05	12738		ABA	X_3
06	00019		ACA	X_2
07	00175		AAA	Y_1
08	00482		ABC	X_3
09	00125		ABA	Y_1
10	00376		ABA	Y_1

ARQUIVO DE DADOS

Dentro do processo de recuperação, o "arquivo invertido" passa a ser um intermediário entre o "Diretório de Atributos" e o "Arquivo de Dados".

A figura (II.2) representa um modelo de organização de um sistema, cujo particionamento lógico do arquivo de dados se encontra estruturado por listas invertidas. Para facilitar a comparação entre os dois esquemas, tomou-se por exemplo o mesmo arquivo de dados da figura (II.1). Ainda neste esquema, cada entrada do índice de uma particular chave secundária corresponde a um valor assumido pelo respectivo atributo, dentro de seu domínio. Quando o "Diretório de Atributos" é decodificado por randomização do valor da chave secundária, em virtude de possíveis ambiguidades, torna-se necessário no processo de tradução para endereço de lista comparar a chave completa. Para isso uma cópia do referido valor deve ser mantida no diretório, o que pode tornar-se inconveniente ocasionando mesmo relevante disperdício de espaço, considerando-se as características das funções de randomização e que os campos de chaves secundárias podem apresentar grande variação de comprimento. Como alternativa, pode-se manter no diretório uma sucessão (String) de bits extraída como subproduto da função de randomização, uma palavra do computador por exemplo, obtendo-se assim uma alta probabilidade de resolução das colisões no próprio diretório.

As possíveis colisões restantes podem ser solucionadas no arquivo invertido, mantendo-se para isso cópia da chave completa nas respectivas listas.

Para efeito de ilustração, passa-se a analisar a mesma expressão lógica utilizada como exemplo na explanação sobre o esquema em Multilista (Chave secundária 1 = ABA) AND (Chave secundária 2 = X_1), a qual seria inicialmente traduzida para os endereços das listas invertidas: 02 para a primeira "condição atômica" e 05 para a segunda "condição atômica". As listas poderiam ser tratadas na memória principal e daí se concluiria que os registros de dados que estariam na interseção seriam 2

(dois): aqueles cujos endereços na representação do arquivo de dados são 02 e 04, correspondendo respectivamente as chaves primárias "00510" e "00036". Só então se processaria a recuperação dos 2 (dois) registros apenas, no arquivo de dados. É frequente na prática a ocorrência de longas listas, enquanto que a interseção delas (operador AND) pode resultar em um pequeno número de entidades, e consequentemente reduzindo em muito o número de acesso ao arquivo de dados.

Os dois esquemas representam diferentes estruturas de arquivo e requerem diferentes tipos de programação, mas eles podem representar a mesma estrutura de informação. Em alguns aspectos eles são idênticos, pois usam o mesmo meio para definir logicamente as partições, que é o encadeamento por endereço, e ambos requerem o mesmo número de campos de endereçamento (link adresses) para particionar um dado arquivo por um determinado número de chaves secundárias. A diferença básica é a localização desses campos de endereçamento, o que, entretanto, dota essas duas estruturas com diferenças de propriedades significantes em termos de tempo de resposta de recuperação, tempo de atualização, facilidade de programação, estatística prévia sobre a recuperação, manutenção de espaço e do próprio processador de pedido. Com o objetivo de balancear (trade-off) as vantagens de cada uma dessas estruturas, existem sistemas que combinam esses dois esquemas, produzindo um sistema Multilista parcialmente invertido.

A principal vantagem da organização em Lista Invertida sobre a Multilista é que o processo de intercalação ou interseção de listas para um pedido é consideravelmente mais eficiente, já que é desenvolvido sobre compactas listas em lugar de requerer um acesso randômico para cada registro pertencente às listas. Assim, o número exato de endereços que responde ao pedido formulado pela expressão lógica (booleana), é transmitido do Diretório do Arquivo, embora possa ser necessário que cada registro seja recuperado, para então ser examinado se satisfaz a condições relativas a campos não chaves secundárias (qualificadores). Pode-se, então, obter uma mais precisa es

tatística de recuperação, que fornecida ao terminal remoto antes da busca de fato ao arquivo de dados, possibilita ao usuário fazer uma modificação no pedido, baseando-se no tamanho do conjunto resultante.

A Lista Invertida apresenta maior flexibilidade para ajustes que a Multilista. É possível em tempo de geração de arquivo, variar o seu grau de inversão como desejado. Assim, o gerente do sistema pode periodicamente alterar os seus parâmetros de acordo com as mais recentes estatísticas sobre o arquivo e seus pedidos, de modo a manter ou aperfeiçoar dinamicamente a capacidade de resposta do sistema.

A desvantagem principal da Lista Invertida é o requerimento de áreas de trabalho para desenvolver os processos de interseção, intercalação e eliminação impostos pelas relações lógicas que correspondem aos operadores booleanos "AND", "OR" e "NOT" respectivamente.

II.4 Seleção de Índice de Chaves Secundárias

Índices Secundários são frequentemente usados em sistemas, que requerem recuperação por chaves secundárias, como um recurso capaz de reduzir significativamente o tempo de recuperação de informação, especialmente quando entidades são qualificadas por diversas chaves secundárias. Entretanto, desde que índices são representações redundantes de dados, eles acarretam incrementos nos custos do sistema por necessitarem de adicional memória secundária para seu armazenamento e manutenção, bem como de procedimentos particulares. Para aplicações que tenham muita atividade de manutenção é importante selecionar cuidadosamente as chaves de indexação. Considerando que "n" é o número de atributos os quais são candidatos a chaves secundárias indexadas, existem 2^n alternativas de escolha para a composição do índice, ou se for fixado que o índice deve ter exatamente K chaves indexadas, existem $\binom{n}{K}$ alternativas de escolha.

Conforme afirma COMER[23], um algoritmo para fazer a seleção do índice ótimo pode requerer 2^k passos ou mais, e adverte que embora o tempo de computação para encontrá-lo possa ser tolerável para alguns, ele pode ser duplicado caso seja acrescido apenas um atributo ao índice de chaves secundárias. Portanto, deve ser imprudência incorporar tais algoritmos em sistemas objetivando recalcular o índice ótimo após várias atualizações. Não obstante o fato que um ótimo conjunto de indexadores é difícil de ser encontrado, pode ser fácil aproximar-se de tal solução rapidamente, produzindo uma boa seleção do índice, o que tem motivado estudos e análises de eficientes algoritmos de aproximação.

Contribuições nesse sentido foram dadas por ANDERSON[20], CARDENAS[21], COMER[23], LIU[24], CASEY[25] e LUM[26], sendo este último um dos mais citados entre os autores consultados. ANDERSON[20] desenvolveu uma função de custo que incorpora recuperação por multiatributos, manutenção de índice e armazenamento de índice. Para isso estabeleceu restrições a formulação de "Pedidos",

visando reduzir a complexidade do problema. Ele obteve como resultado do trabalho uma "função objetivo" a qual pode ser usada para encontrar a alternativa de escolha de índices de menor custo.

Em resumo, diz ANDERSON[20], "O custo de manutenção foi encontrado ser pequeno comparado as variações em custo de recuperação, mas não despresível". No caso, por exemplo, em que a atividade de manutenção é moderada (ou baixa) e distribuição uniforme de recuperação sobre todas as chaves candidatas, sugere indexações total como a melhor alternativa.

Se a atividade de manutenção torna-se grande e a recuperação irregular, poucas chaves secundárias podem ser recomendadas como índice.

Em geral, é inviável manter listas de indexações para todos os atributos representados no arquivo de dados, pois isso pode demandar espaço para armazenamento secundário de grande monta, chegando até mesmo a ser superior àquele necessário ao próprio arquivo de dados. Além disso, pode acarretar um adicional custo de manutenção que não compense a parcela do custo de recuperação salvo. Segundo esta ótica, é necessário definir as prioridades dos atributos em relação ao interesse e conveniência de os tornar chaves secundárias com listas de indexação próprias. É de acordo com o nível de prioridade que se pode reconhecer um atributo como candidato a pertencer ao índice de chaves secundárias. O interesse pode ser caracterizado pela frequência estimada com que um atributo estaria envolvido entre todos os processos de recuperação de informação previstos para um determinado período de tempo.

O conjunto de chaves secundárias que compõe o "Diretório" deve ser frequentemente ajustado ao longo do tempo de uso do sistema, tendo-se para isso que avaliar as tendências, com base em observações sobre períodos passados, ou mesmo por eventual conhecimento de mudança no comportamento futuro do ambiente por motivos de natureza peculiar à aplicação.

Por outro lado, quando o "Diretório" tem sua estrutura segundo o esquema randônico, impõe restrições no uso de operadores aritméticos do tipo "maior do que" ou para valores entre faixas, etc., como já foi dito, o que pode tornar inconveniente manter como chave secundária qualquer atributo que assuma grande variedade de valores dentro de seu domínio, como por exemplo aqueles que representam quantidades (ex.: em unidades métricas, em cruzeiros, em número de entidades, etc.). Nesse caso, raramente cada valor ocorre mais que uma vez entre as entidades. Esse fato gera um grande número de listas, podendo em consequência desperdiçar muito espaço em memória secundária.

O caso oposto também tem suas inconceniências, quando um atributo pode assumir uma pequena variedade de valores dentro do seu domínio, como por exemplo o atributo sexo: apenas duas grandes listas existem diminuindo consideravelmente a velocidade do processo de resolução de qualquer "pedido" que as envolva. Caso esse atributo apareça frequentemente em conjunção com um outro, pode se tornar adequado tratá-los como uma única chave secundária, ao que se chama chave "secundária composta". Como vantagem tem-se lista de comprimento reduzido e uma pré-resolução da conjunção, diminuindo o custo da resolução dos pedidos que os envolve na forma composta.

Em princípio, aparenta ser atrativo candidato à chave secundária todo campo que representa atributos codificados, visto distinguir claramente classes dos mesmos, agregando várias entidades. Observa-se na prática ser frequente o interesse dos usuários por atributos com tais propriedades como por exemplo: quais produtos compõem um determinado "Pedido de fornecimento?; quais são os pedidos pendentes de um cliente? quais são os economistas do departamento A? e etc. Entretanto, as observações anteriores devem ser preservadas.

III. MODELO DE UM PROCESSO CONVERSACIONAL PARA RECUPERAÇÃO DE INFORMAÇÃO POR MULTI-CHAVES

III. MODELO DE UM PROCESSO CONVERSACIONAL PARA RECUPERAÇÃO DE INFORMAÇÃO POR MULTI-CHAVES

Conforme as considerações das seções (II.3) e (II.4) e levando-se em conta as restrições do ambiente mencionadas em (I.2), parece ser mais razoável que o "Modelo" orientado para equipamentos de pequeno porte fosse delineado sob um esquema voltado para reduzir o uso de memória principal, sem no entanto comprometer o tempo de recuperação ou o processo de manutenção dos arquivos de dados e índices secundários.

Diante de tais premissas concluiu-se por adotar para a estrutura do "Diretório de Atributos" o esquema "randômico", tendo como consequência a impossibilidade de resolver as funções dos operadores relacionais (maior do que, entre faixas, etc.) na fase anterior a recuperação dos registros de dados. Isso pode não ser uma restrição muito forte já que é recomendável levar a cabo o processo de seleção em duas etapas, tornando-o mais flexível. Assim, na segunda etapa, pode-se aplicar sobre os registros recuperados em consequência do resultado da primeira, um pedido complementar, refinando o resultado do processo com maior liberdade de escolha quanto aos operandos e operadores. Estando o registro de dados na memória, é possível aplicar sobre o mesmo qualquer "condição atômica" (conforme sua definição no capítulo anterior), para quaisquer dos campos de atributo. Na realidade, isso nem sempre vem reduzir a eficiência do processo, mas dota o sistema de maior flexibilidade, devendo ser lembrado que o resultado da primeira etapa em geral se constitui de um número reduzido de registros. Vale ressaltar que essas restrições se impõem não só aos passos do processo de interação usuário/computador, mas à própria linguagem de formulação dos pedidos como será visto mais adiante.

O esquema de particionamento escolhido para o Arquivo de Dados (Arquivo Mestre) foi o de "listas-invertidas", por garantir para as formas mais frequentes de pedidos (aplicação do

operador AND) um número reduzido de acesso à memória secundária em relação ao esquema em "multi-lista". Ressalte-se que o esquema em "listas-invertidas" oferece como vantagens adicionais:

a) uma estatística prévia à recuperação no arquivo de dados, que pode servir de subsídio ao se tomar a decisão de refinar ou ampliar o pedido anterior;

b) a liberação de espaço do Arquivo de Dados, pois não implica em se manter campos de endereçamento nos registros de dados; e

c) uma maior flexibilidade para alterar o conjunto de chaves-secundárias mantidas no "Diretório de Atributos", pois não implica em atualização do Arquivo de Dados (Arquivo Mestre).

III.1 Algoritmo do Processo Conversacional

O processo conversacional para recuperação de informações por multi-chaves, dotado da flexibilidade citada, pode ter os seguintes passos*

0. (u) - Solicita inicialização do programa;

1. (c) - Carrega programa e executa as rotinas de inicialização (fornece cópia da Tabela de Definição do Registro Mestre-ou-Dicionário de Dados);

2. (c) - Mensagem 'Tecle seu Pedido ou Descontinui (D)';

3. (u) - Usuário tecla seu pedido segundo linguagem própria (definida em (3.3), ou a letra "D" se deseja encerra as consultas;

4. (c) - Se 'D' descontinua programa;

5. (c) - Chama Analisador de Pedido;

   5.1 - Se erro, dá mensagem e retorna a 2;

   5.2 - Executa Processo de Pré-Seleção obedecendo a ordem de prioridade.

---

* - Os passos assinalados com (u) significam ações tomadas pelo usuário e os com (c) significam ações executadas pelo computador. As mensagens são fornecidas pelo computador através de Vídeo e usuário responde por Teclado.

6. (c) - Fornece quantidade de elementos do subconjunto reconhecido e pergunta se deseja refinar: mensagem- "XXX Entidades Resultantes do seu Pedido: se deseja refinar tecle (s)";

7. (u) - Usuário pode responder "S" se deseja refinar ou qualquer outro carater em caso contrário;

8. (c) - Lê, se for "S" retorna a 2;

9. (c) - Mensagem - "Deseja selecionar após Recuperação (S)"?

10. (u) - Usuário tecla "S" se sim, ou qualquer outro caracter se não;

11. (c) - Lê, se "S" dá mensagem "Forneça Pedido";

11.1 (u) - Usuário fornece expressão lógica;

11.2 (c) - Analisador de Pedido verifica se há erro; em caso positivo, dá mensagem de erro e retorna para 9;

12. (c) - Executa rotina de recuperação de registro do arquivo de dados para cada chave-primária do subconjunto selecionado em (5.2);

12.1 - Aplica expressão lógica de (11.1) se houver, eliminando aqueles registros para condição falsa;

12.2 - Coloca cada registro selecionado no arquivo de saída (vídeo e impressora no modelo implementado);

13. (c) - Retorna para 2.

Como mostrado no algoritmo anterior, o usuário interfere no processo apenas na formulação de Pedido, e naturalmente na sua inicialização ou finalização. Entretanto, pode alguma aplicação deste sistema requerer alternativas para as saídas do resultado do Pedido que impliquem em alterações no passo (12). Assim por exemplo:

a) poderia o usuário desejar saída em impressora ou vidro, de apenas alguns campos dos registros resultantes, o que seria necessário informar qual unidade de saída e segundo qual formato; e

b) poderia o usuário querer uma cópia do conjunto de registros selecionados, em disco por exemplo, para tratamento posterior.

No modelo implementado não foram considerados os passos (9), (10) e (11), o que em nada prejudica o caráter experimental deste trabalho. Caso venha ser requerido, o acréscimo de tais rotinas não interferem naquelas implementadas, mas ao contrário, a interface entre elas encontra-se claramente definida (área definida para registro mestre e a tabela do Dicionário de Dados).

Para a formulação dos Pedidos o usuário necessita conhecer as regras de sintaxe de linguagem, que quando do "passo 3" deve ser a definida em (3.3), bem como deve restringir as "condições atômicas" aos atributos designados como "Chave-secundária" os quais são destacados na "Tabela de Definição de Registro Mestre" com códigos especiais* , observando o seu comprimento (número de caracteres do campo).

---

* - No modelo implementado, as chaves-secundárias são unicamente aqueles atributos assinados com o Tipo "1", no Dicionário de Dados.

Para melhor clareza, imagine-se que se tem um arquivo de artigos técnicos de uma biblioteca, e que alguém deseja saber quais publicações da 'COPPE' tratam de 'RECUPERAÇÃO DE INFORMAÇÃO', mas que tenham sido editadas após determinada data.

Inicialmente o usuário no passo (3) do algoritmo anterior, formula seu Pedido de Publicações do Editor = COPPE e ASSUNTO = RECUPERAÇÃO DE INFORMAÇÃO.

Então é analisado o Pedido e realizada a pré-seleção dos artigos que satisfazem tais condições identificados apenas pelo endereço (simólico ou físico) dos registros pertencentes a tais artigos. A quantidade deles é informada (passo 6), mas suponha-se que o usuário não deseja fazer neste instante qualquer refinamento quanto ao pedido anterior. Entretanto, sendo que a condição "data posterior a" implica em um operador relacional, só no passo (11) o usuário poderia complementar seu pedido "DATA > = JAN/74". No passo (12) cada um dos registros selecionados no passo (5), seria recuperado e na memória seria verificado se satisfaz a condição relativa a data da publicação. Apenas aqueles para os quais essa condição fosse verdadeira seriam expostos ao usuário.

III.2 Modelo Simplificado do Processamento de Pedidos

A figura (III.1) é uma representação gráfica simplificada dos módulos funcionais e suas ligações* que compõem o sistema de recuperação de informações por "multi-chaves", objeto deste trabalho, capaz de executar o algoritmo do processo conversacional exposto em (III.1). Seu objetivo é dar uma idéia das partes componentes e do mecanismo global, facilitando a compreensão das funções de cada módulo, os quais são separadamente descritos, com mais detalhes nos próximos capítulos.

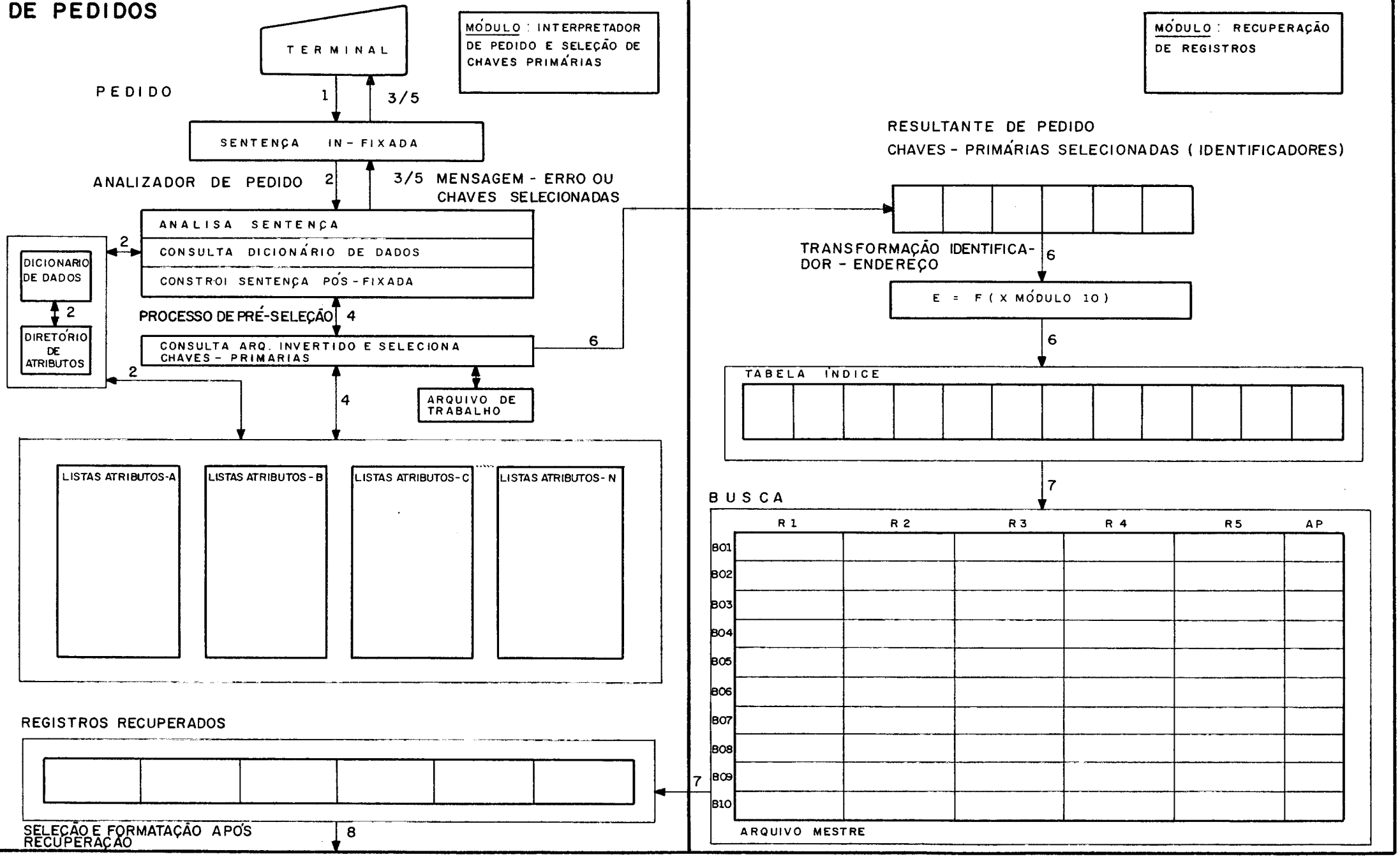
Segue uma sumária descrição do "Processamento de Pedido" - figura (III.1), o qual é iniciado a partir do fornecimento de um "Pedido" formulado pelo usuário.

---

* - A sequência de execução das funções encontra-se representada na figura (III.1) pela numeração das setas indicadoras do fluxo de informação e do controle do processo.

FIG. III - 1

PROCESSAMENTO DE PEDIDOS



PROCESSAMENTO DE PEDIDOS - FIGURA (III.1)

MÓDULO INTERPRETADOR DE PEDIDOS E SELEÇÃO DE CHAVES-PRIMÁRIAS

1. Formulação de Pedido

Usuário fornece seu "Pedido" expresso por uma sentença lógica, na forma in-fixada, conforme a linguagem definida em (III.3);

2. Analisador de Pedidos

2.1 - Analisa a sentença lógica quanto a sintaxe da linguagem de inquirição (III.3).

2.2 - Consulta "Dicionário-de-Dados", apanhando localização e comprimento de cada atributo, caso tenham sido reconhecidos como aqueles que possuem listas-invertidas (chaves-secundárias) e apanha os endereços de início de cada lista.

2.3 - Transforma sentença lógica da forma in-fixada para pós-fixada, substituindo as "condições atômicas" pelos endereços das listas relativas aos valores de cada atributo, através de consultas sucessivas ao Dicionário de Dados*, ao Diretório de Atributos e ao primeiro registro da lista** caso ela não seja vazia.

---

* - Em cada entrada do Dicionário de Dados encontra-se o endereço do registro do Diretório de Atributos que corresponde a localiza do índice das listas invertidas, correspondentes aos valores assumidos por uma mesma chave-secundária.

** - A consulta a lista invertida, nesta fase, é necessária para solucionar possíveis ocorrências de colisão.

3. Mensagem de Erro

Em caso de erro reconhecido pelo "Analisador de Pedido" fornece mensagem, despresando o Pedido.

4. Processo de Pré-Seleção

Com base na sentença pós-fixada, executa entre os operandos representados pelos endereços de listas, aquelas funções lógicas* compreendidas, obedecendo a precedência estabelecida. Para isso, percorre essas listas invertidas, guardando os resultados intermediários no Arquivo de Trabalho, e o resultado final na área denominada "Resultante de Pedido", constituindo o subconjunto de chaves-primárias dos registros (entidades) que satisfazem as condições expressas pelo Pedido.

5. Mensagem sobre Quantidade Resultante

Concluído o bloco anterior (4) o usuário é informado sobre a quantidade de registros que satisfazem ao seu "Pedido", cujos endereços sejam guardados na "Resultante do Pedido", e lhe é perguntado se deseja alterar o "Pedido" anterior. Se se tiver um novo "Pedido" o processo é reiniciado. Se não, o controle é passado para o "Módulo de Recuperação de Registros".

---

* - As funções lógicas aqui referidas são as Sooleanas "AND", "OR" e "NOT".

. MÓDULO DE RECUPERAÇÃO DE REGISTROS

A cada chave-primária selecionada e armazenada na área "Resultante de Pedido", a qual é uma inter-face entre os módulos, são executados os seguintes passos:

6. Transformação Identificador-a-Endereço

Aplica a função "hashing" a qual transforma valor da chave-primária para endereço de entrada na "Tabela Índice do Arquivo Mestre", onde encontra-se o endereço do registro do Arquivo Mestre a partir do qual deve ser iniciada a busca para essa chave.

7. Busca

A partir do endereço obtido na "Tabela Índice do Arquivo Mestre" é realizada uma busca sequencial* para a chave-primária que se estar tratando naquele instante. Quando encontrado, o registro é transferido para um arquivo de Registros Recuperados.

8. Seleção e Formatação Após Recuperação

Caso tenha sido requerido, nessa etapa pode ser realizado os seguintes procedimentos:

8.1 - Verificar, para cada registro recuperado, se satisfaz ao Pedido Complementar (passo II. de III.1); são despresados aqueles registros para os quais essa condição é falsa.

8.2 - Os registros selecionados são transferidos para o arquivo informado pelo usuário, segundo o formato especificado pelo mesmo.

- O Arquivo Mestre é organizado de tal forma que chaves-primárias sinônimas (que colidem em um mesmo endereço) são dispostas em registros em posições adjacentes.

O bloco (8) corresponde ao passo (12 de III.1), e por conseqüência deve prever a execução daqueles requisitos quando necessários, relembrando que o mesmo não foi contemplado.

III.3 Linguagem para Formulação de Pedido

Na construção dessa linguagem considerou-se os seguintes fatores:

- Facilidade de compreensão e consequente utilização;

- Possibilidade de interação, compreendendo a flexibilidade de se poder usar o pedido anterior na formulação de um novo pedido; e

- Necessidade de se tornar simples e rápido o processo de análise de uma expressão lógica, quanto a sintaxe da linguagem, o que levou a se codificar as identificações dos atributos com três caracteres, sendo os dois últimos o número de ordem de sua entrada na Tabela de Definição do Registro Mestre.

LINGUAGEM CONVERSACIONAL - BNF

(Forma Normal de Backus)

```
< pedido > :: = < expressão >?|
                < pedido anterior > < operador > <expressão >?

< expressão > :: = < qualificador >
                   < qualificador > <operador> <expressão>

< qualificador >:: = <atributo   =  valor
                     (<subexpressão >)

< subexpressão > :: = <expressão>

< atributos>     :: = <alfa> <numérico> <numérico>

< valor >        :: = <alfanumérico>|
                      <alfanumérico> <valor>

< alfanumérico > :: = <alfa> | <numérico>

< alfa >         :: =  A|B|C|D| ...  X|Y|Z

< numérico >     :: =  1|2|3| ...  8|9|0

< operador >     :: = '|'|'&|'N'

< pedido anterior> :: =  PA
```

Notas Explicativas

1. Os símbolos operadores |, &, N, correspondem aos operadores booleanos OR, AND, NOT respectivamente.

2. "PA" significa que se deseja aplicar a expressão lógica do Pedido sobre o resultado do "Pedido Anterior".

3. Significado dos operadores:

   a) seja o Pedido: (qualificador 1/qualificador 2); serão recuperados todos os registros de entidades que tenham o qualificador 1 ou o qualificador 2 ou ambos;

   b) seja o Pedido: (qualificador 1 & qualificador 2); serão recuperados todos os registros de entidades que satisfaçam ambos os qualificadores; e

   c) seja o Pedido: (qualificador 1 N qualificador 2); serão recuperados todos os registros de entidades que possuam o qualificador 1 mas que não possuam o qualificador 2.

Delimitadores da linguagem :: = carater branco, "=", "("e")"

Símbolo terminal - ?

Palavras reservadas - PA
- identificadores de atributos:

C01 - Chave de identificação
A01 - ... A99 - atributos qualificadores

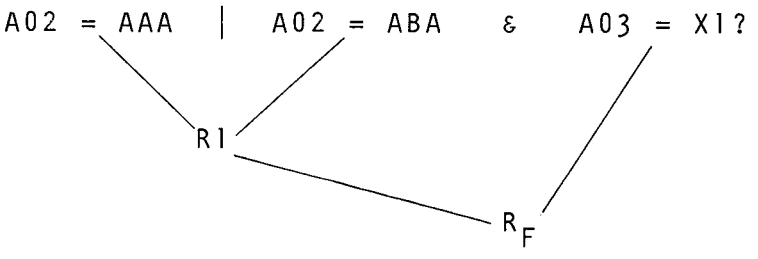
4. A identificação do qualificador (nome do campo) e o valor a ele atribuído na expressão tem que estar de acordo com a Tabela de Definição do Registro Mestre (Dicionário de Dados).

Prioridade das Operações

Os três operadores possuem o mesmo nível de prioridade, sendo que as expressões lógicas são resolvidas da esquerda para a direita.

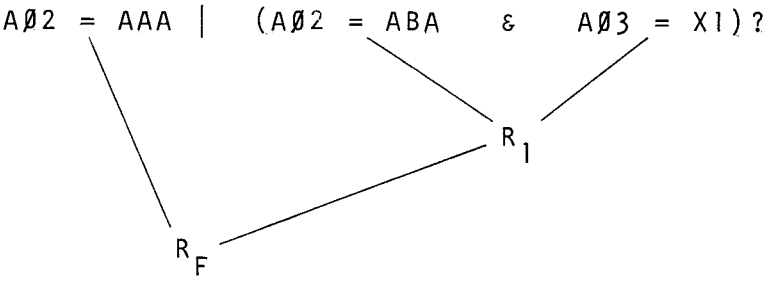
Para efeito de ilustração, veja-se os exemplos a seguir considerando que AØ2, AØ3, AØ4 e AØ5 tenham sido definidos como chaves-secundárias; R_1, R_2, R_3 ..., representam a sequência de Resultantes das Resoluções intermediárias e R_F e a Resultante Final.

Exemplo:



Pode-se utilizar parênteses para quebrar a sequência de Resoções.

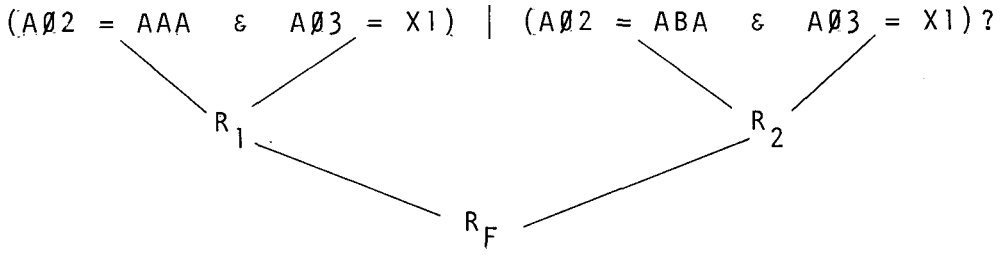
Exemplo:



Apesar de neste exemplo estarem representadas as mesmas condições atômicas e operadores, na mesma sequência que no exemplo anterior, a colocação dos parênteses do sequendo exemplo altera a Resultante Final.

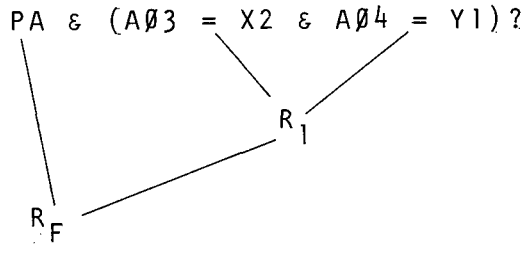
A expressão abaixo ilustra uma forma alternativa para se obter o mesmo resultado da primeira expressão, porém envolvendo um maior número de Resoluções e consequentemente, mais tempo de processamento.

Exemplo:



Finalmente, se houver especificação de Pedido Anterior (PA) à esquerda da expressão, não é alterada a sequência, mantendo-se válidas as regras anteriores.

Exemplo:



cujo Resultado Final é um refinamento do Pedido Anterior.

IV. MÓDULO INTERPRETADOR DE PEDIDOS E SELEÇÃO DE CHAVES PRIMÁRIAS (MODELO IMPLEMENTADO)

IV. MÓDULO INTERPRETADOR DE PEDIDOS E SELEÇÃO DE CHAVES PRIMÁRIAS (MODELO IMPLEMENTADO)

Como já mencionado no Capítulo (II.3), a recuperação de informação por múltiplas-chaves, relacionadas por operadores booleanos (AND, OR, NOT), tem como requisito desejável que o arquivo de dados (Arquivo Mestre) seja particionado, de modo a proporcionar ao processo um grau de eficiência satisfatório, o que se torna um imperativo quando se tratando de sistema interativo.

Com o objetivo de possibilitar ao leitor melhor nível de compreensão dos mecanismos que compõem o modelo implementado que resultou deste trabalho, julgou-se necessário detalhar inicialmente a organização do conjunto de arquivos orientados para o acesso secundário. Tal conjunto é composto dos arquivos seguintes:

- Dicionário de Dados ou
  Tabela de Definição do Registro Mestre;

- Diretório de Atributos; e

- Arquivo Invertido.

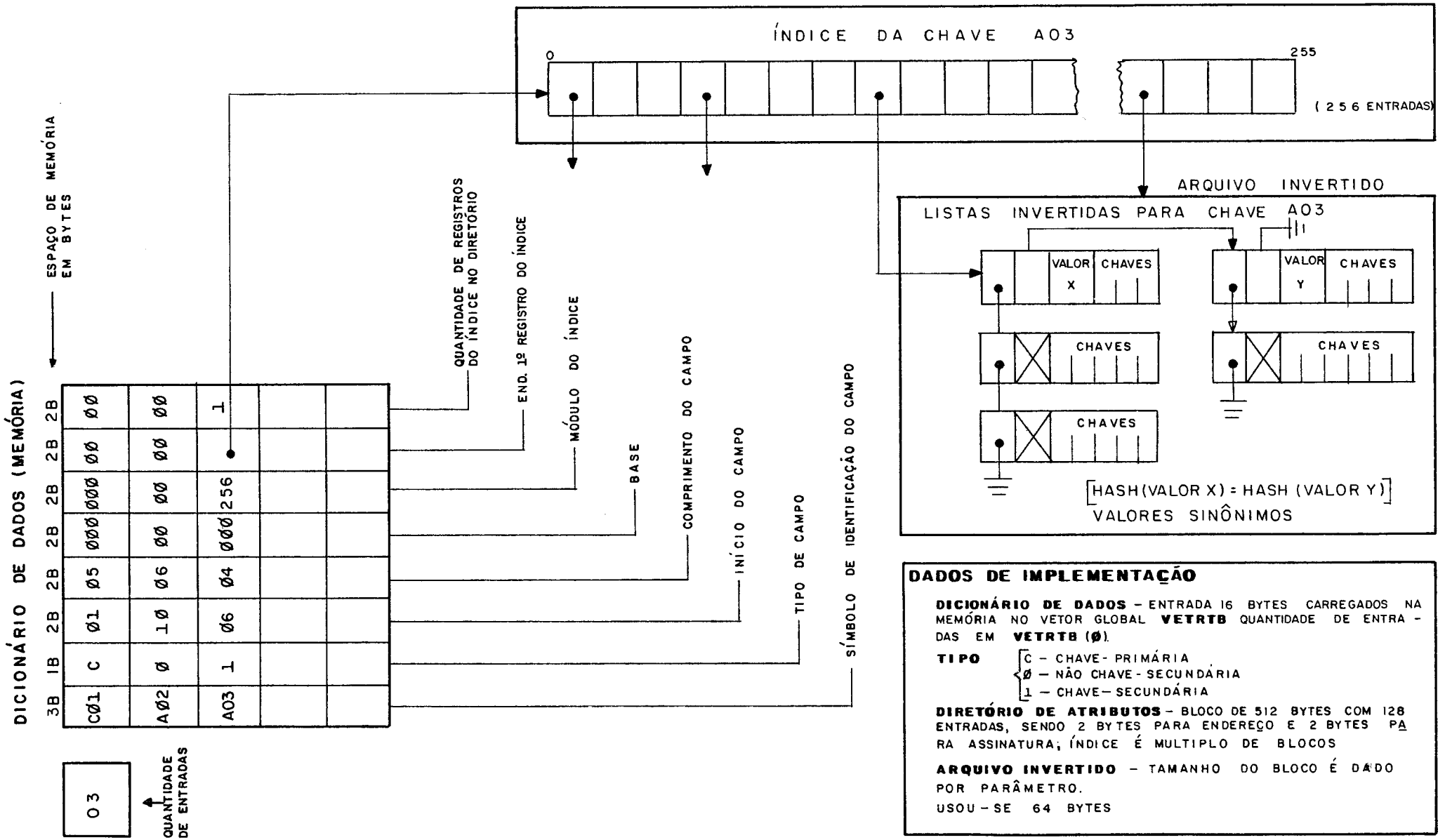
Lembra-se que no esquema de particionamento por "lista invertida", tais listas são construídas de forma a identificar todos aqueles registros que, para um determinado atributo, tenham assumido o mesmo valor. Para tanto, quando da manutenção ou consulta às listas, o caminhamento é iniciado pela visita ao "Dicionário de Dados". Cada entrada deste corresponde a um campo (atributo de entidade) do Registro Mestre, contendo informações tais como a identificação do atributo, seu comprimento e localização no registro mestre, e caso se trate de uma chave-secundária*, indica quais registros do arquivo "Di

---

* - Só existe lista para chave-secundária.

FIGURA IV.1

DIRETÓRIO DE ATRIBUTOS



DADOS DE IMPLEMENTAÇÃO

DICIONÁRIO DE DADOS – ENTRADA 16 BYTES CARREGADOS NA MEMÓRIA NO VETOR GLOBAL **VETRTB** QUANTIDADE DE ENTRADAS EM **VETRTB (Ø)**.

TIPO
- C – CHAVE-PRIMÁRIA
- Ø – NÃO CHAVE-SECUNDÁRIA
- 1 – CHAVE–SECUNDÁRIA

DIRETÓRIO DE ATRIBUTOS – BLOCO DE 512 BYTES COM 128 ENTRADAS, SENDO 2 BYTES PARA ENDEREÇO E 2 BYTES PARA ASSINATURA; ÍNDICE É MULTIPLO DE BLOCOS

ARQUIVO INVERTIDO – TAMANHO DO BLOCO É DADO POR PARÂMETRO.
USOU-SE 64 BYTES

retório de Atributos" foram alocados para constituírem o Índice correspondente a mesma chave-secundária, que suporta seu módulo.

Aplicando uma função "hashing" sobre o valor da chave-secundária obtém-se o provável endereço da entrada do diretório onde se encontra o endereço da lista (pode ocorrer colisões). Obtido o endereço da lista, a mesma é visitada, seja para atualizá-la ou apenas consultá-la.

A figura IV.1 é uma representação esquemática da relação entre os três arquivos orientados para o acesso secundário. Nela aparece destacada uma única chave-secundária (A03), seu Índice, e exemplificando o caso de colisão de valores de mesma chave, e o artifício que possibilita a formação de listas independentes para esses valores. No exemplo subentende-se que os valores X e Y do atributo A03 são sinônimos e aparecem apenas no primeiro registro de suas respectivas listas, com a finalidade de solucionar as colisões.

IV.1 Tabela de Definição do Registro Mestre (Dicionário de Dados)

Tem como finalidade descrever as características dos campos que compõem o registro do "Arquivo Mestre", tais como código de identificação, localização, comprimento e número de ocorrências, bem como informações necessárias ao tratamento daqueles escolhidos como chave-secundária.

Tal tabela é mantida em arquivo magnético (DECJTB-ver lay-out e processo de manutenção em anexo), sendo carregadas na memória aquelas suas informações relevantes ao tratamento de dados dos registros mestres, seja para manter o "Arquivo Mestre", seja para manter ou possibilitar consulta às listas invertidas através de chaves-secundárias.

A carga se procede na fase de inicialização do programa (rotina ECTBMT) sobre o vetor (VETRTB), definido como global, sendo guardado no seu primeiro byte o número de entradas da tabela, e nos bytes seguintes cada uma das entrada constituída de 16 bytes, distribuídas sobre o vetor de acordo com seu número de identificação. Assim possibilita o acesso imediato pelo cálculo de seu deslocamento como múltiplo de 16 (dezesseis) bytes.

Essas informações relevantes e seus usos, descreve-se abaixo:

- Símbolo de identificação do campo - (3 bytes) utilizado nas condições atômicas na formulação de um pedido, como referência ao campo desejado.

- Tipo - (1 byte) caracteriza o campo quanto a possibilidade de ser utilizado como chave-secundária. Pode assumir os valores seguintes:

  . C - Chave-primária (sempre a primeira entrada da tabela) ou campo de controle;

. Ø - campo não utilizado como chave-secundária;

. 1 - campo escolhido como Chave-secundária, mas que só é aceito em condição atômica para a qual o operador relacional é "=" (igual);

. 2 - (não implantado) campo para o qual seu domínio é dividido em número de intervalos igual ao módulo do Índice da Chave (número de entradas), utilizado quando se tratando de quantidades ou valores e se desejar aplicar operadores relacionais do tipo (maior do que, menor do que, etc.). Nesse caso, o módulo da função hashing por divisão* utilizada para endereçamento ao Índice da Chave é determinado pelo quociente de Base = ⌈ (Valor Máximo ÷ Número de Entradas do Índice) ⌉ arredondado para o inteiro imediatamente superior. Desse artifício resulta que a ordem ascendente das faixas é mantida pelos endereços de suas correspondentes entradas no Índice da Chave, garantindo-se a independência das listas. A implementação desse tipo de campo, implica em adaptação nas rotinas do Analisador de Pedido e do Processo de Pré-seleção.

- Início do Campo - é o número de ordem da posição do Registro Mestre em que inicia o campo.

- Comprimento do Campo - número de posições (em bytes) do Registro Mestre ocupadas pelo campo.

- Base - valor base para introdução de artifícios que permitam em casos especiais, o uso de operadores relacionais do tipo (maior do que, menor do que, etc.) como ocorre para campo do tipo (2) descrito anteriormente.

---

* - Ver KNUTH[1], SOUSA[10], SEVERANCE[28], LUM[31] ou CRUZ[9].

- Módulo do Índice - número de entradas do Índice da Chave-secundária pertencente ao "Diretório de Atributos", o qual deve ser dimensionado em função da quantidade de ocorrências prováveis dos valores assumidos pelo campo que representa o atributo.

- Endereço do Primeiro Registro do Índice - é o endereço do primeiro registro do arquivo Diretório de Atributos (DECJ ii) alocado como Índice da Chave Secundária.

- Quantidade de Registros do Índice - é a quantidade de registros do arquivo Diretório de Atributos alocados como Índice da Chave-secundária. Essa quantidade é determinada pela relação entre o Módulo do Índice e o número de entradas em um registro (bloco) do arquivo Diretório de Atributos, que no modelo implantado possui 128 entradas; portanto Quant.-reg. = $\lceil$ Módulo ÷ 128 $\rceil$ arredondado para o inteiro imediatamente superior.

  A quantidade de entradas é utilizada no processo de carga da tabela, bem como para verificar se um endereçamento excede o limite da mesma, erroneamente. Pode um atributo ter mais de uma ocorrência para uma entidade, o que reflete em campos múltiplos em entradas adjacentes no Dicionário de Dados, mas compartilhando Índice de Chave.

. Acesso ao Dicionário de Dados

A consulta ao "Dicionário de Dados" carregado sobre o vetor (VETRTB), é realizada pela soma do deslocamento da entrada deseja em relação a origem da Tabela com o deslocamento do campo em relação ao início dessa entrada. Exemplo - informação de início de campo cujo símbolo de identificação é "AØ3", que corresponde à terceira entrada do "Dicionário de Dados":

deslocamento = (3-1) X 16 + 4

IV.2 Diretório de Atributos

Tem como finalidade endereçar as listas pertencentes a cada valor assumido por uma mesma Chave-secundária. Seu arquivo (DECJII) é constituído de registros de 512 "bytes" de 128 entradas com 2 (dois) "bytes" para endereço de lista e 2 (dois) para guardar a respectiva assinatura*. Os campos de endereçamento ocupam a primeira metade do bloco e os campos de assinatura a segunda do mesmo bloco. Mantém-se um deslocamento de 256 bytes como relação de correspondência do campo de endereçamento de ordem "r" para o campo de "assinatura" de mesma ordem. Por outro lado, um Índice de Chave é formado por tantos registros consecutivos quantos forem necessários a conter o número de entradas (Módulo) definido no Dicionário de Dados para sua respectiva Chave-secundária. A localização de cada Índice de Chave dentro do diretório é explicitada no Dicionário de Dados como já mencionado, ocorrendo essa alocação no instante de inicialização do conjunto de arquivos pertencentes ao "Módulo Interpretador de Pedidos e Seleção de Chaves Primárias". Tal inicialização está descrita mais adiante.

Endereçamento a Índice de Chave

Tendo-se no Dicionário de Dados quais registros do Diretório de Atributos estão alocados ao Índice de uma determinada chave-secundária, torna-se, agora, necessário conhecer o processo para determinação da entrada correspondente a um valor assumido por essa chave-secundária.

---

* - Denomina-se "assinatura", SOUSA[10], à sucessão de bits resultante da primeira fase da função de endereçamento, obtida pela transformação do valor da chave-secundária ao domínio de valores que uma variável "address"(2 bytes ou uma palavra do POTI) pode assumir.

A transformação de um conteúdo qualquer de uma chave-secundária para endereço da sua correspondente entrada pertencente ao Índice dessa Chave, é executada em três etapas:

a) transforma o valor da chave-secundária, qualquer que seja sem comprimento, para um valor dentro do domínio de valores positivos que uma variável "address" (2 bytes ou uma palavra do POTI) pode assumir (0 a 32767), aplicando-se sucessivamente os comandos "SHIFT" (deslocamento de 1 bit a esquerda) e "XOR" (ou exclusivo) entre a variável "address" e cada caracter da Chave-secundária * . Esse método, conhecido como "dobramento" (folding), é bastante útil quando se tratando com chaves de comprimento maior que uma palavra do computador, CRUZ[9], SOUSA[10], LUM[31].
Como resultado dessa etapa, obtém-se na variável "address" o que chamou-se anteriormente de assinatura";

b) mapea o valor positivo da assinatura sobre o Índice da chave-secundária utilizando-se seu módulo (n) definido no Dicionário de Dados. Para isso adotou-se o método "divisão", KNUTH[1], LUM[31], que é simplesmente E (Hi) ← ASSINATURA mod n. Segundo LUM[31], é suficiente que n não tenha divisores menores de 20 para que se obtenha bons resultados do mapeamento. Aqui, o resultado E (Ki) indica qual o número de ordem da entrada do Índice da Chave, deveria pertencer ao valor em tratamento; e

c) determina em qual registros do Diretório de Atributos encontra-se a entrada E (Ki), conhecido aquele em que inicia o Índice da Chave e sabendo-se que existem 128 entra

---

* - O comprimento da Chave-secundária é apanhado no Dicionário de Dados. (Essa "PROCEDURE" foi denominada de ECHASH e encontra-se embutida na sub-rotina ECRATB).

das por registro. O método utilizado é também da "divisão", onde o quociente Q(Ki), somado ao número de ordem (m) do primeiro registro do Índice da Chave, indica aquele (I), em que se está interessado, cuja entrada de ordem R (Ki) dada pelo resto da divisão, é a procurada. Assim:

Q (Ki) ← E (Ki) / 128;

R (Ki) ← E (Ki) mod 128; e

I ← Q (Ki) + m.

Naturalmente que em (a) e (b) podem ser geradas colisões*, para as quais foram previstos dois níveis de solução:

1) No próprio Índice - por endereçamento aberto, KNUTH[1], SEVERANCE[28], e outros, quando a colisão é gerada apenas em (b). Na estratégia adotada, ao se consultar a "entrada" endereçada, verifica-se a igualdade das assinaturas (a gerada e a da entrada). Caso sejam diferentes, consulta-se as subsequentes entradas, tomando o registro do "Índice" como uma lista circular. A ocorrência de igualdade de "assinaturas", ou uma "entrada" vazia, interrompe o processo.

2) na lista invertida - para o caso em que a colisão ocorre na etapa (a), ou seja, quando valores diferentes de uma mesma "chave-secundária" ocasionam assinaturas iguais. Essa ocorrência obrigou a que fosse mantido na respectiva lista invertida uma cópia do próprio valor da Chave-secundária, a fim de garantir listas distintas para valores distintos. A colisão de "assinaturas" gera uma es

---

* - "Colisão" chama-se ao fenômeno em que h (Ki) = h (Kj) para $K_i \neq K_j$, e diz-se que K_i e K_j são sinônimos.

trutura em lista de listas(Plexo), como representado no "arquivo invertido" da figura (IV.1).

A manutenção da assinatura no próprio Índice justifica-se pela necessidade de reduzir o comprimento das listas de listas no "arquivo invertido", reduzindo consequentemente o custo do processo de reconhecimento da lista referente a um determinado valor. Tem-se como principal vantagem solucionar a maioria das colisões com operações sobre o adequado registro do Índice de Chave, mantido na memória principal, visto que existem 2^{15} possíveis assinaturas diferentes. Por outro lado, seria incoveniente armazenar no Índice de Chave o próprio valor da Chave-secundária, pois em geral são de comprimento superior a 2 "bytes" (1 palavra do POTI) além de bastante diversificado, tornando a comparação mais lenta. Utilizando-se a assinatura em lugar do valor da própria Chave-secundária podemos garantir um maior número de entradas por registro, mantendo esse número fixo, além de não incorrer em qualquer Ônus adicional, visto que a assinatura é gerada, em primeiro lugar, para atender a outra necessidade.

IV.3 Arquivo Invertido

Tem como finalidade prover espaço para criação e manutenção das listas invertidas "LI's" relativas a cada valor assumido por cada uma das chaves-secundárias. (ver figura IV-2). Vale lembrar que o esquema em lista invertida adotado particiona o Arquivo Mestre, mantendo explicitamento as relações entre todos os registros de um mesmo subconjunto, definido pelo par "Chave-secundária e seu valor". Para isso, tais listas endereçam simbolicamente cada registro, armazenando as respectivas Chaves-primárias.

Os registros do Arquivo Invertido* são de três formatos, de comprimento** fixo, porém o sistema permite ao analista escolher o mais adequado a sua aplicação, sendo necessário apenas que seja informado através de parâmetro, em tempo de inicialização dos arquivos desse módulo. Os dados que integram os referidos registros, são os seguintes:

Registro "Header"

Esse é sempre o primeiro registro do arquivo (registro zero). Os campos que o compõem estão relacionados abaixo:

- comprimento do Registro Li - (Address) comprimento dos registros do Arquivo de Listas Invertidas (DECJLi), representado em hexadecimal. Seu valor é fornecido pelo usuário juntamente com os dados de geração do Dicionário de Dados, e está limitado a 512 bytes ou um setor

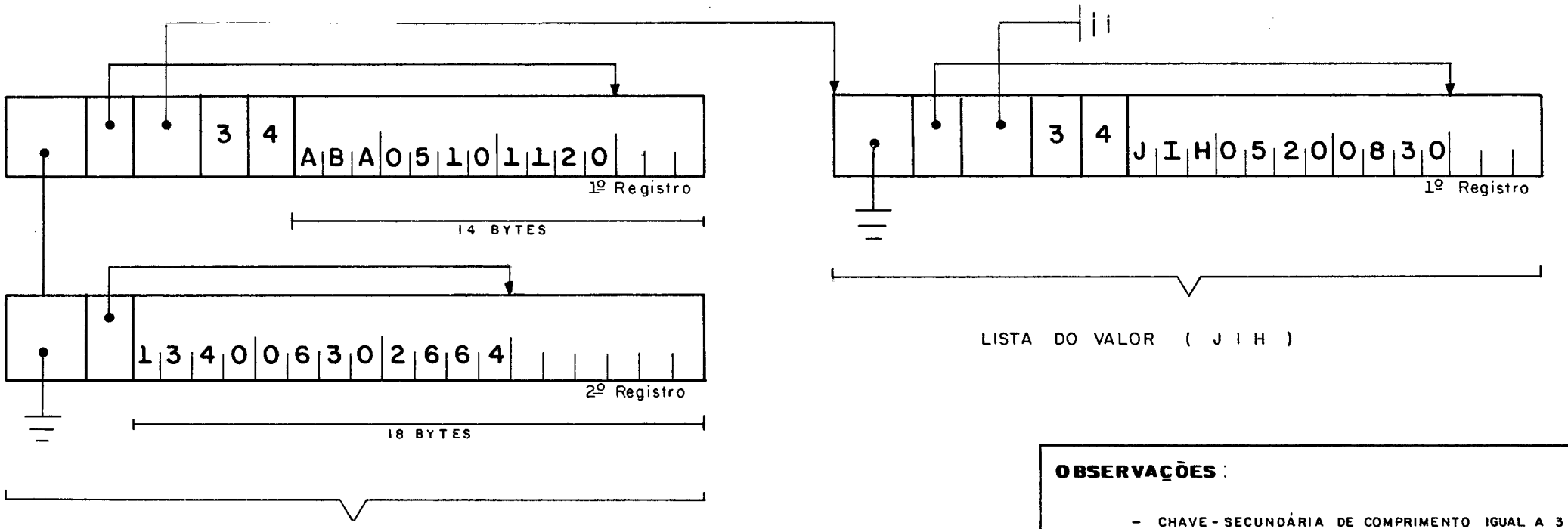
---

* - O Arquivo Invertido foi implementado com endereçamento simbólico (DECJLi);

** - O comprimento escolhido para os ensaios do modelo, foi 64 bytes.

FIGURA IV. 2

EXEMPLO DE LISTAS INVERTIDAS



OBSERVAÇÕES:

- CHAVE-SECUNDÁRIA DE COMPRIMENTO IGUAL A 3;
- CHAVE-PRIMÁRIA DE COMPRIMENTO IGUAL A 4;
- VALORES SINÔNIMOS "ABA" E "JIH";
- PARA O VALOR "ABA" EXISTEM 5 CHAVES-PRIMÁRIAS; E
- REGISTRO LI COM 21 BYTES

do disco. No modelo ensaiado os registros "Li" possuiam 64 bytes de comprimento.

- Número de Ocorrências da Chave-Primária - (Address) número máximo de chaves-primárias que um registro de lista invertida (Li) pode conter; é fornecido por parâmetro como o item anterior.

- Nome do Arquivo Li - (30 bytes) denominação por extenso do arquivo, colocado automaticamente.

- Quantidade de Registros Li Criados - (Address) é a quantidade de registros do arquivo, calculada automaticamente no instante de inicialização dos arquivos.

- Quantidade de Registros "Li" Utilizados - (Address) é a quantidade dos registros "Li" que estão efetivamente ocupados.

- Quantidade de Registros "Li" Disponíveis - (Address) é a quantidade de registros Li que estão disponíveis para uso, naquele instante.

- Data de Gravação - (6 bytes) é a cópia da data (ANO/MÊS/DIA) de gravação do arquivo Dicionário de Dados (DECJTB) o qual foi inicializado na mesma ocasião que o Arquivo de Listas Invertidas. Isso se justifica pelo fato de que, qualquer alteração no Dicionário de Dados que implica em alteração de Chave-secundária, obriga a que se recarregue todas as listas.

- Bytes Complementares - (bytes) são "bytes" complementares na definição desse "header", para o comprimento total de "Li".

. Primeiro Registro de Lista Invertida (Li)

Todas as listas invertidas, para cada valor assumido por cada Chave-secundária, tem seu primeiro registro formatado segundo a descrição abaixo:

- Apontador de Expansão (Link*) - variável address (2 bytes); tem como finalidade indicar, se houver, o endereço do próximo registro de expansão da "Li" que se refere a um determinado valor de uma chave-secundária. Se seu valor for zero não há registro de expansão.

- Quantidade de Posições Utilizadas - variável byte (1 byte); tem como finalidade indicar a quantidade de posições (bytes) ocupadas do "vetor para armazenamento de chaves". Essa quantidade comparada relativamente ao comprimento do registro "Li", indica o local para se inserir uma nova chave-primária ou que a capacidade desse registro foi esgotada.

- Apontador de Lista de Sinônimo - variável address (2 bytes); tem como finalidade indicar, se houver, o endereço do primeiro registro "Li" pertencente a um outro valor da mesma chave-secundária, sinônimo do primeiro valor pelo fato de gerarem mesma assinatura. Portanto, diz respeito ao segundo nível de solução de colisão tratado na seção (IV.2), no item "Endereçamento a Índice de Chave". Quando igual a zero, indica que não há subsequente lista de sinônimo.

- Comprimento da Chave-Secundária - variável byte (1 byte); informa o comprimento da chave-secundária a qual diz respeito a "Li".

---

* - KNUTH[2] se refere como "link variable" ou "pointer variable".

É útil ainda, para se determinar a primeira posição de primeira chave-primária armazenada no registro.

- Comprimento de Chave-Primária - variável byte (1 byte); corresponde ao número de caracteres da Chave - primária do Arquivo Mestre, a qual é de tamanho fixo. Sua finalidade é reduzir o número de consultas ao "Dicionário de Dados".

- Vetor para Armazenamento de Chaves - vetor "byte" de dimensão igual ao comprimento do registro Li menos 7 (sete) bytes, referentes aos campos anteriores. Destina-se a armazenar em suas primeiras posições o valor da chave-secundária a qual pertence tal "Li", enquanto as posições subsequentes estão reservadas para as chaves-primárias dos registros de dados que, para essa mesma chave-secundária, possuem o valor a que se refere a"Li".

. Registro de Expansão de Lista Invertida (Li)

Tem como finalidade expandir o comprimento das Li's que necessitam de uma maior capacidade de armazenamento de Chaves-primárias. Esse registro pode ocorrer em uma Li tantas vezes quanto necessária for. Ele possui três espécies de dados, como descrito abaixo:

- Apontador de Expansão (Link) - idêntico ao do primeiro registro da Li, descrito anteriormente.

- Quantidade de Posições Utilizadas - idêntico ao do primeiro registro da Li, descrito anteriormente.

- Espaço para Armazenamento de Chaves - vetor "byte" de dimensão igual ao comprimento do registro Li subtraído de 3 (três "bytes" correspondentes aos dois campos anteriores). Destina-se a armazenar apenas as chaves-primárias dos registros de dados que, para uma mesma chave-secundária, possuem o valor ao qual se refere a Li, ou seja, aquele mantido no seu primeiro registro.

IV.4 Rotinas de Tratamento das Listas Invertidas

As rotinas de tratamento das listas invertidas foram construídas de forma a tornar flexível a escolha do comprimento do registro Li, deixando ao projetista o encargo de escolher aquele que melhor se ajuste às necessidades da aplicação. No caso do POTi, esse comprimento está limitado a 512 "bytes" (1 Setor), sendo a criação do arquivo (DECJLi) realizada por um programa de uso geral (CRIE). O comprimento dos registros Li com que foi criado o arquivo (DECJLi) tem que ser fornecido como parâmetro, em tempo de inicialização, como foi mencionado na descrição do registro "header" na seção (IV.3).

É necessário lembrar que cada lista invertida está associada ao par:

. Identificação da Chave-secundária - definida pela sua posição relativa dentro do Dicionário de Dados, tendo-se utilizado apenas seu número de ordem; e

. Valor da Chave-secundária - que é entendido como sendo o valor representado pelos caracteres localizados nas posições destinadas a sua respectiva chave-secundária dentro da área de leitura do Registro Mestre (REGMEST).

Para isso, valeu-se de informações do Dicionário de Dados relativas a localização e comprimento de cada campo.

Na realidade, para escrever um programa que lide com as listas invertidas, é suficiente conhecer duas rotinas:

a) (Rotina ECISLi) aquela que fornecendo-se o par identificação da chave-secundária/valor da chave secundária, pode-se executar as três operações básicas às Li's, de acordo com a opção desejada. Quais sejam:

. Consultar (c) - apenas localiza o primeiro registro da Li definida pelo par acima, deixando-o na área de lei

tura;

. Incluir (I) - percorre a Li definida pelo par acima, até encontrar, ou adicionar, espaço para a inserção de uma nova chave-primária, caso essa chave-primária não exista na mesma Li.

. Excluir (E) - percorre a Li definida pelo par acima, até encontrar a chave-primária de mesmo valor, procedendo a exclusão desta, ou até concluir que tal chave-primária não existe na mesma Li.

É patente a impropriedade da opção "Alterar", considerando que tal fato só poderia advir de uma alteração do valor de uma chave-primária de um Registro Mestre, o que não deve ser permitido por ameaçar a integridade e segurança da base de dados.

b) (Rotina ECHDLi) - aquela que ao final do processamento grava o registro "header" atualizado, em função da mudança do estado de utilização do espaço reservado ao arquivo (DECJLi). Portanto, só requer uso obrigatório quando o programa permitir a opção "Incluir", pois apenas essa adquire registro do espaço disponível acoplando-os em alguma Li, enquanto não há a alternativa desses mesmos registros retornarem ao espaço disponível.

. Descrição da Rotina "ECISLi"

Para utilizar essa rotina* é necessário definir no módulo mais externo do programa algumas áreas de trabalho declaradas como "GLOBAIS", além de outros cujos endereços devem ser man

---

* - Para mais detalhes sobre a construção dessa rotina, veja-se no anexo sua codificação em PLTi.

tidos em variáveis declaradas como "GLOBAIS". Esse recurso permitiu uma considerável e importante economia de memória para a implementação do "Modelo".

Tais áreas são:

. "VETRTB" - variável "global" - vetor "Tabela de Definição do Registro Mestre" ou Dicionário de Dados, cuja dimensão deve ser no mínimo 16 (dezesseis) vezes o número de campos do Registro Mestre, mais 1 (um) byte para guardar a quantidade de entradas da tabela.

. "LISTAINVERTE" - área de leitura dos registros do arquivo (DECJLi), sobre os quais são construídas as listas invertidas (Li). Seu endereço deve ser guardado na variável "address" "BLP", definida como "global".

. "INDICEii" - área de leitura dos registros do arquivo (DECJii), sobre os quais são construídos os índices das chaves-secundárias que constituem o Diretório. Essa área tem dimensão fixada em 512 bytes. Seu endereço deve ser guardado na variável "address" "Bii", definida como "global".

. "REGMEST" - área para passagem dos valores das chaves, secundárias e primária, a qual deve coincidir com a dimensão dos registros do Arquivo Mestre (DECJ2Ø). Seu endereço deve ser guardado na variável "address" BMT, definida como "global".

. ARA133 - área de 133 bytes destinada a impressão de mensagens de erro. Esse vetor deve ser definido como variável "global".

O comando de chamada da rotina é da forma:

"CALL ECiSLi (RLi, CODATB, Rii, ALTER)";

Seus parâmetros tem as seguintes funções:

a) Na chamada - ao se fazer a chamada é obrigatório informar qual opção se deseja e para qual chave-secundária. Assim, precedendo ao comando CALL, devem ser atribuídos valores que traduzam o que se deseja, as variáveis abaixo:

CODATB - número de ordem do campo chave-secundária dentro do Dicionário de Dados, representado em hexadecimal; e

ALTER - com o código "C" para consultar ou "I" para incluir ou "E" para excluir, cujos significados descreveu-se anteriormente.

Quanto ao parâmetro Rii, deve ser atribuído o valor zero em "tempo de inicialização" do programa, sem qualquer outra atribuição de valor. Nas subsequentes chamadas contém o endereço do último registro lido do Diretório de Atributos (DECJii), podendo evitar uma leitura física de disco.

b) No Retorno - quando o controle do processamento retorna ao ponto de chamada, obtém-se como indicação de resultado:

. Procedimento bem sucedido - CODATB = # FF (hexadecimal) e RLi contém o endereço do último registro de lista processado, deixado na área de leitura, mas para as alternativas de inclusão ou exclusão (DECJLi) foi completada. Caso a alternativa tenha sido "Consultar" ou "EXCLUIR" e a lista procurada estivesse vazia, acarretaria RLi = 0.

. Procedimento mal sucessivo - de modo geral, se identifica pelo retorno na variável "CODATB" de valor hexadecimal igual a # FA, afora o caso em que deseja-se "INCLUIR" e o espaço do arquivo (DECJLi) encon

tra-se exaurido, não havendo registro Li para expansão de lista.

As demais alternativas, para quando CODATB = # FA são: RLi = 'B' (carater) - significa erro no código ALTER, ou seja, diferente de "C" ou "I" ou "E".

RLi = "*" (carater) - significa atributo declarado no "Dicionário de Dados" como não chave-secundária;

RLi = "C" (carater) - significa atributo declarado no Dicionário de Dados como chave-primária;

RLi = "L" (carater) - significa que o registro do Índice da Chave-secundária encontra-se cheio, não havendo disponibilidade de entrada para criação de uma nova lista, o que requer uma ampliação do Índice dessa chave-secundária;

RLi = "N" (carater) - significa atributo não existente no Dicionário de Dados;

RLi = "K" (carater) - significa que o valor do atributo encontra-se com carater branco; e

RLi = "X" (carater) - significa outras ocorrências anormais.

Feitas estas considerações, passa-se a descrever o tratamento* propriamente dito das listas invertidas, dado um par "identificação da chave-secundária/valor da chave-secundária.

---

* - As figuras (IV.1) e (IV.2) podem ser úteis a compreensão do tratamento aqui descrito.

Em uma primeira etapa, comum as três opções, é feito o reconhecimento da chave-secundária declarada, que consiste na consulta ao "Dicionário de Dados" pelo processo Acesso ao Dicionário de Dados descrito em (IV.1).

Caso seja constatado que a chave-secundária do par tenha sido declarada corretamente, são guardados seus parâmetros necessários a próxima etapa.

A segunda etapa consiste em localizar no "Diretório de Atributos", qual entrada corresponde ao valor atribuído à chave-secundária de acordo com a descrição dada em "Endereçamento a Índice de Chave" de (IV.2). Dessa etapa pode-se obter ou o endereçamento da "lista de listas" onde deve ser procurada a lista pertencente àquele valor, ou a indicação de que não existe "lista de lista" para tal valor, caso em que a entrada do Índice de Chave encontra-se vazia (lista vazia).

Para executar essas duas primeiras etapas foi escrita a subrotina (ECRATB), cuja codificação em PLTi encontra-se em anexo. Seu comando de chamada é:

```
CALL ECRATB (CODATB, RLi, Rii, iNiCAT, COMPAT, HASHii);
```

onde os parâmetros CODATB, RLi e Rii tem as mesmas funções mencionadas para (ECISLi).

Quanto aos demais tem-se:

- iNiCAT - retorna da chamada com a posição inicial do campo identificado pelo número de ordem fornecido em CODATB;

- COMPAT - retorna da chamada com o comprimento do mesmo campo identificado em CODATB; e

- HASHii - retorna da chamada com o valor de assinatura, determinado para o valor atribuído ao campo identificado em CODATB (chave-secundária), segundo o processo relatado em Endereçamento a Índice de Chave de (IV.2).

Considerou-se duas ocorrências de procedimento bem sucedido reconhecidas quando:

. CODATB = # FF e RLi > 0 (zero) - significa que existe uma "lista de lista" para o par chave-secundária/Valor da Chave-secundária, cujo endereço de início é deixado em RLi; e

. CODATB ≠ # FF e RLi = 0 (zero) - significa lista vazia para o par chave-secundária/valor da chave-secundária. Os valores atribuídos aos demais valores são dos tipos já mencionado, exceto para CODATB, o qual recebe o número de ordem da entrada disponível pertencente ao registro Índice da Chave (cujo endereço é deixado em RLi), de onde deve ser apontado o endereço da lista a ser criada, caso se deseje (opção incluir).

As demais combinações dos valores atribuídos a CODATB (# FA) e RLi siginificam procedimento mal sucedido, como visto para a chamada a (ECiSLi).

A terceira etapa do tratamento dado as listas invertidas, é executada apenas quando as duas primeiras tiverem sido bem sucedidas. Apresentando características distintas para cada opção da rotina (ECiSLi), julgou-se ser melhor para o entendimento descrever cada caso isoladamente, como segue:

- Opção "CONSULTAR"

. Se lista vazia (CODATB ≠ # FF e RLi = 0 (zero) - o processo de consulta é encerrado, sendo atribuído às variáveis de retorno CODATB = # FF e RLi = 0 (zero);

. Se "lista de listas" existe (CODATB = # FF e RLi > 0) - percorre a lista de listas a partir do endereço em RLi, consultando os primeiros registros das listas de cada sinônimo, comparando o valor da Chave-secundária com aquele valor que se encontra nas primeiras posições do

campo "Vetor para Armazenamento de Chaves" mencionado na descrição do "Primeiro Registro de Lista Invertida (Li)" na seção (IV.3). Existem duas alternativas como resultante: há lista e não há lista para o par chave-secundária/valor da chave-secundária. Em resumo, os passos para a terceira etapa quando tratando desta opção, considerando que a chamada (CALL) a rotina (ECRATB) tenha sido bem sucedida, são os seguintes:

1 - lê arquivo DECJLi para o endereço RLi;

2 - compara valor da chave-secundária com valor em "Vetor para Armazenamento de Chaves" de comprimento igual a COMPAT, ou seja,

Se(if) para todo (I) variando de zero a (COMPAT-1):
REGMEST (INICIAT + I) = VALORATRIB* (I)

faça: 2.1 - Condição VERDADEIRA - a RLi foi atribuído endereço da Li que satisfaz a igualdade, então faz:
CODATB = # FF; retorna;

caso contrário (else)

2.2 - Condição FALSA - consulta o "Apontador de Lista de Sinônimo" verificando se há outra lista:

2.2.1 - Condição FALSA - executa as atribuições RLi = 0 (zero) e CODATB = # FF; retorna;

---

* - VALORATRIB - nome do campo dado na declaração do registro Li (LISTA INVERTE) - os nomes de campo aqui utilizados encontram em anexo.

2.2.2 - Condição VERDADEIRA - faz RLi = "Apontador de Lista de Sinônimo" e volta para (1);

3 - fim

Portanto, se lista vazia ou lista não existente para o par chave-secundária/valor da chave-secundária, aos parâmetros resultantes da chamada à rotina (ECiSLi), são atribuídos os seguintes valores:

CODATB = # FF e RLi = 0 (zero)

- Opção INCLUIR

. Se lista vazia - utiliza os parâmetros CODATB, Rii e HASHii resultantes da chamada a (ECRATB), para inicializar uma nova lista. Tratando-se dessa ocorrência os passos são os seguintes:

1 - solicita da lista de espaço disponível o próximo registro Li, cujo endereço é fornecido na variável address "I" - (CALL PEDELi(I););

2 - caso I = 0 (arquivo DECJLi exaurido) faz CODATB="A"; e retorna;

3 - inicializa aquela designada entrada do registro "Índice da Chave" que encontra-se na área de leitura (iNDICEii) do arquivo DECJii (o número de ordem da entrada encontra-se em CODATB e valor da assinatura em HASHii); ou seja:

INDICEii (CODATB) = I;

INDICEii (CODATB + 128) = HASHii;

4 - regrava (REWRITE) o último registro lido do arquivo (DECJii) a partir da área INDICEii;

5 - prepara o primeiro registro Li da lista para gravar:

move brancos para registro Li (LISTAINVERTE);

move valor da chave-secundária REGMEST (INICAT) para VALORATRIB com comprimento igual a COMPAT;

inicialize "comprimento da chave-secundária" (COMPATRIB) e quantidade de posições utilizadas" (QTULiLiZLi) com valor de COMPAT;

inicialize com zero os apontadores de expansão (LKEXPANLi) e de lista de Sinônimo (PROXLiSiN);

move valor da chave-primária em REGMEST (VETRTB(5)) para Li em VALORATRIB (QTUTiLiZLi) com comprimento igual a VETRTB (8). (Comprimento da chave-primária) incrementa "quantidade de posições utilizadas" do comprimento da chave-primária QTUTiLiZLi=QTUTiLiZLi + VETRTB (8);

guarda o "comprimento da chave-primária" em COMPCHAV;

6 - grava o registro preparado no arquivo (DECJLi) a partir da área LISTAINVERTE no endereço I;

7 - atribui as variáveis de retorno os seguintes valores: RLi = I e CODATB = # FF;

8 - retorna; fim.

Se "lista de lista" existe - inicia realizando os mesmos passos da opção CONSULTAR, até localizar a lista que pertence ao par chave-secundária/valor da chave-secundária, ou até concluir que tal lista não existe. Novamente considera-se que a chamada a rotina (ECRATB)

tenha sido bem sucedida. Para facilitar a compreensão, apresenta-se a seguir uma descrição do algoritmo juntamente com um macrodiagrama da sequência lógica dos passos.

1 - lê registro do arquivo DECJLi de endereço RLi;

2 - compara o valor da chave-secundária mencionada na chamada à (ECiSLi) com o correspondente valor da Li lida; ou seja:

Se(if)

para toda (I) variando de zero a (COMPAT-I) REGMEST (INICCAT + I) = VALORATRIB (I)

Faz:

2.1 - Condição VERDADEIRA - trata-se da lista procurada;

- verifica se chave-secundária já encontra-se no registro Li;

2.1.1 - em caso afirmativo - faz CODATB= #FF; e retorna;

2.1.2 - em caso negativo - faz:
verifica se há espaço no registro Li para a nova chave-primária, ou seja: (QTUTiLiZLi + comprimento da chave-primária VETRTB(8) < (comprimento do registro Li (COMPLi) - 6);

2.1.2.1 - em caso afirmativo - faz move REGMEST (início chave-primária) para VALORATRIB (QTUTiLiZLi) com comprimento da chave-primária VETRTB(8);

- incrementa "quantidade de posições

utilizadas" do comprimento da chave-primária;
QTUTiLiZLi = QTUTiLiZLi + VETRTB(6);

- CODATB = # FF;

- regrava (REWRITE) o último registro lido e atualizado do arquivo (DECJLi) a partir da área LISTAINVERTE; e retorna;

2.1.2.2 - em caso negativo (não há espaço no registro Li) - verifica se há registro de expansão; (LKEXPANLi ≠ 0)

2.1.2.2.1 - em caso negativo (LKEXPANLi = 0) expande lista:

solicita da lista de espaço disponível o próximo registro Li, cujo endereço é fornecido em "I" - (CALL PEDELi (I));

- caso I = 0 (arquivo DECJLi exaurido) faz CODATB = "A"; e retorna;

- faz ligação para o novo registro de expansão da lista (LKEXPANLi = I)

- regrava (REWRITE) o último registro lido e atualizado do arquivo (DECJLi) a partir da área LISTAINVERTE;

- faz RLi = I, CODATB = # FF;

- prepara Li, guardando chave-primária;

- atribui zero ao apontado LKEXPANLi = 0 e ao contador de "quantidade de posições utilizadas" atribui o comprimento da chave-primária (QTUTiLiZ=LCHAVEP);

- grava o registro preparado no arquivo (DECJLi), a partir da área LISTAINVERTE;

- retorna;

2.1.2.2.2 - em caso afirmativo (há registro de expansão, ou seja LKEXPANLi≠0) faz RLi = LKEXPANLi e lê registro de (DECJLi) de endereço RLi;

2.1.2.2.2.1 - se a chave-primária a inserir já existe em Li, faz CODATB= #FF, e retorna;

2.1.2.2.2.2 - (chave-primária não se encontra em Li);

2.1.2.2.2.2.1 - se há espaço para inserir chave-primária, ou seja, (QTUTiLiZLi + LCHAVEP) < (COMPLi-3) guarda chave-primária no registro Li, incrementa "quantidade de posições utilizadas" do comprimento da chave-primária: (QTUTiLiZLi = QTUTiLiZLi + LCHAVEP), e CODATB = # FF;

- regrava (REWRITE) o último registro lido e atualizado do arquivo(DECJLi) da área LISTAINVERTE; e
- retorna;

2.1.2.2.2.2.2 - se não há espaço para inserir chave-primária volta a(2.1.2.2).

Caso contrário (else)

2.2 - Condição FALSA (não se trata da lista procurada)

2.2.1 - se há lista para sinônimo, ou seja (PROXLiSiN ≠ 0), faz
RLi = PROXLiSiN; e volta a (1);

2.2.2 - se não há lista para sinônimo, ou seja (PROXLiSiN = 0).

- solicita registro Li da lista de espaço disponível, cujo endereço é fornecido em "I" - (CALL PEDELi(I);
- caso I = 0 (arquivo DECJLi exaurido) faz DOCATB = "A"; e retorna;
- faz ligação para a nova lista de sinônimo (PROXLiSiN = I);
- regrava (REWRITE) o último registro lido e atualizado do arquivo (DECJLi) a partir da área LISTAINVERTE;
- faz RLi = I e CODATB = ## FF;
- prepara o primeiro registro Li da lista para gravar (mesmo passo (5) da ocorrência lista vazia descrito anteriormente);
- grava o registro preparado no arquivo (DECJLi) a partir da área LISTAINVERTE no endereço I;
- retorna;

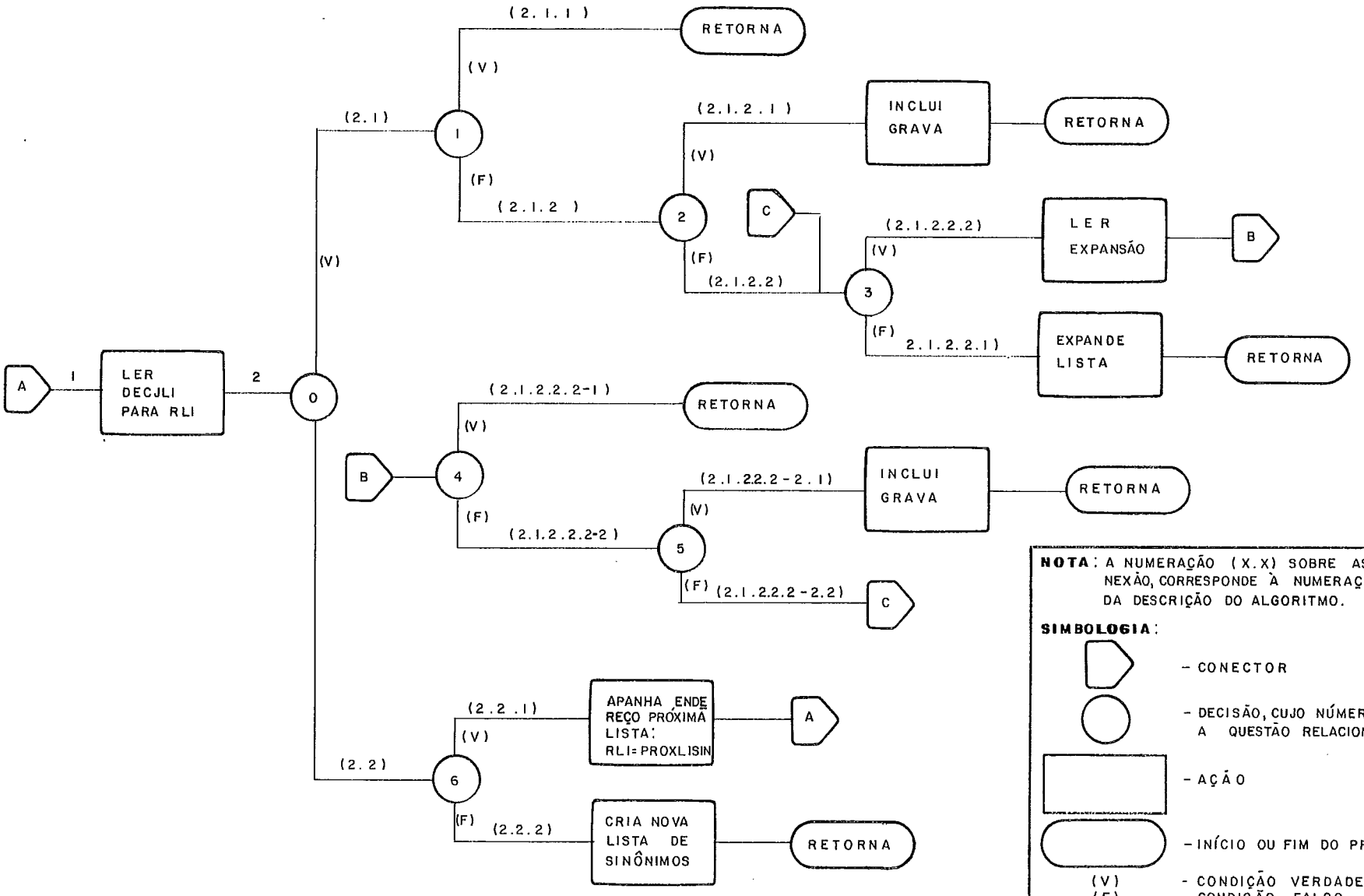
fim;

FIGURA IV.3

OPÇÃO INCLUIR

MACRO-DIAGRAMA DA SEQUÊNCIA LÓGICA DE PASSOS

(PARA OCORRÊNCIA "EXISTE LISTA DE LISTAS")



NOTA: A NUMERAÇÃO (X.X) SOBRE AS LINHAS DE CONEXÃO, CORRESPONDE À NUMERAÇÃO DOS BLOCOS DA DESCRIÇÃO DO ALGORITMO.

SIMBOLOGIA:

- CONECTOR
- DECISÃO, CUJO NÚMERO CORRESPONDE A QUESTÃO RELACIONADA NA LEGENDA
- AÇÃO
- INÍCIO OU FIM DO PROCESSO

(V) - CONDIÇÃO VERDADE

(F) - CONDIÇÃO FALSO

. Legenda da figura (IV-3)

- OPÇÃO INCLUIR

Macrodiagrama da Sequência Lógica de Passos para a Ocorrência "Existe Lista de Listas" (ou seja, quando a resposta da chamada à ECRATB indicar CODATB = ≠ FF e RLi > 0 (zero).

Relação das Questões:

0. Trata-se da lista procurada? - esta questão é traduzida pela comparação do valor da chave-secundária com o valor guardado no primeiro registro da lista (exemplo fig. IV.2).

1. A chave-primária que se deseja inserir, já se encontra no registro Li?

2. Há espaço no registro Li para se proceder a inserção da chave-primária?

   Se QTUTiLiZLi + VETRTB(8) < COMPLi - 6;

3. Essa lista tem continuação em registro Li de expansão?

   Se LKEXPANLi ≠ 0;

4. A chave-primária que se deseja inserir, já se encontra no registro Li de expansão;

5. Há espaço no registro Li de expansão para se proceder a inserção da chave-primária?

   Se QTUTiLiZLi + VETRTB(8) < COMPLi

6. Há outra lista para algum valor sinônimo do valor da chave-secundária em tratamento?

Vale observar que a inclusão de uma chave-primária na lista que diz respeito a seu par "chave-secundária/valor da chave-secundária", pode ser completada apenas quando tal chave-primária não existe nesta mesma lista (evitando duplicidade causada por algum erro do usuário). Quanto ao local da inserção pode se dar em alguma posição vazia intermediária à lista, em consequência de alguma anterior exclusão, ou pode ocorrer após a última chave-primária inserida. Por outro lado, as chaves-primárias, dentro das listas, não são mantidas ordenadas com relação aos seus valores. O esquema adotado de resolução dos operadores booleanos (por randomização das chaves-primárias), permitiu realizar as inserções segundo a ordem cronológica de chegada, aproveitando-se todos os espaços vazios intermediários dos registros Li's, evitando deslocamentos de dados e mantendo o custo de inserção em níveis mais baixos.

- OPÇÃO EXCLUIR

Prever tratamento para as duas alternativas possíveis chamada com resultado bem sucedido à rotina (ECRATB): "lista vazia" e "Existe lista de listas".

. Se lista vazia (CODATB ≠ # FF, RLi = 0 (zero) - o processo de exclusão é encerrado, sendo atribuído às variáveis de retorno os valores conforme segue:

CODATB = # FF e RLi = 0 (zero)

. Se "lista de listas" (CODATB = # FF e RLi > 0 (zero)

A exemplo desta alternativa para as opções anteriores, cabe a primeira etapa do processo localizar qual lista pertence ao par chave-secundária/valor da chave-secundária, caso a mesma exista (a figura IV.4) auxilia a compreensão.

1 - lê registro de endereço RLi pertencente ao arquivo (DECJLi);

2 - compara valor da chave-secundária mencionada na chamada a (ECiSLi) com valor em "Vetor para Armazenamento de Chaves" de Li, com comprimento igual a COMPAT:

Se (if)

para todo (I) variando de zero a (COMPAT-1) REGMEST (INICAT+I) = VALORATRIB (I)

Faz:

2.1 - Condição VERDADEIRA - (trata-se da lista procurada);

Se chave-primária de REGMEST encontra-se em Li na posição (I),

faz:

2.1.1 - move a última chave-primária do registro Li para a posição (I) e reduz o contador de posições utilizadas em Li de valor igual ao comprimento da chave-primária; ou seja:

move VALORATRIB (QTUTiLiZLi - VETRTB(8)) para VALORATRIB (I) de comprimento VETRTB (8); e

faz QTUTiLiZLi = QTUTiLiZLi - VETRTB(8));

regrava (REWRITE) registro Li no arquivo (DECJLi);

faz CODATB = # FF; e retorna;

caso contrário

2.1.2 - (Se a chave-primária não se encontra em Li)

Se existe registro de expansão para a lista,

faz

2.1.2.1 - lê próximo registro de expansão;

ou seja

faz: RLi = LKEXPANLi;

lê registro de endereço RLi pertencente ao arquivo DECJLi);

se chave-primária de REGMEST encontra-se em Li de expansão na posição (I)

faz:

2.1.2.1.1 - move a última chave-primária do registro Li para a posição (I) do mesmo e reduz o contador de posições utilizadas em Li de valor igual ao comprimento da chave-primária; ou seja:

move CHAVEIDLi (QTUTiLiZLi-VETRTB (8) para CHAVEIDLi (I) de comprimento VETRTB(8);

faz: QTUTiLiZLi = QTUTiLiZLi - VETRTB(8);

regrava (REWRITE) registro Li no arquivo (DECJLi)

faz CODATB = # FF; e retorna;

caso contrário

2.1.2.1.2 - (Chave-primária de REGMEST não se encontra em Li de expansão)

volta a 2.1.2;

caso contrário

2.1.2.2 - (não existe registro de expansão para a lista) - portanto, não há o que excluir.

faz: CODATB = # FF; e retorna;

caso contrário

2.2 - Condição FALSA - (Não se trata da lista procurada)

2.2.1 - Se há lista para sinônimo, ou seja

se PROXLISIN ≠ 0 (zero)

faz: RLi = PROXLiSiN; e volta a (1);

caso contrário

2.2.2 - Se não há lista para sinônimo (não há o que excluir), ou seja

se PROXLiSiN = 0 (zero)

faz: CODATB = # FF; e retorna;

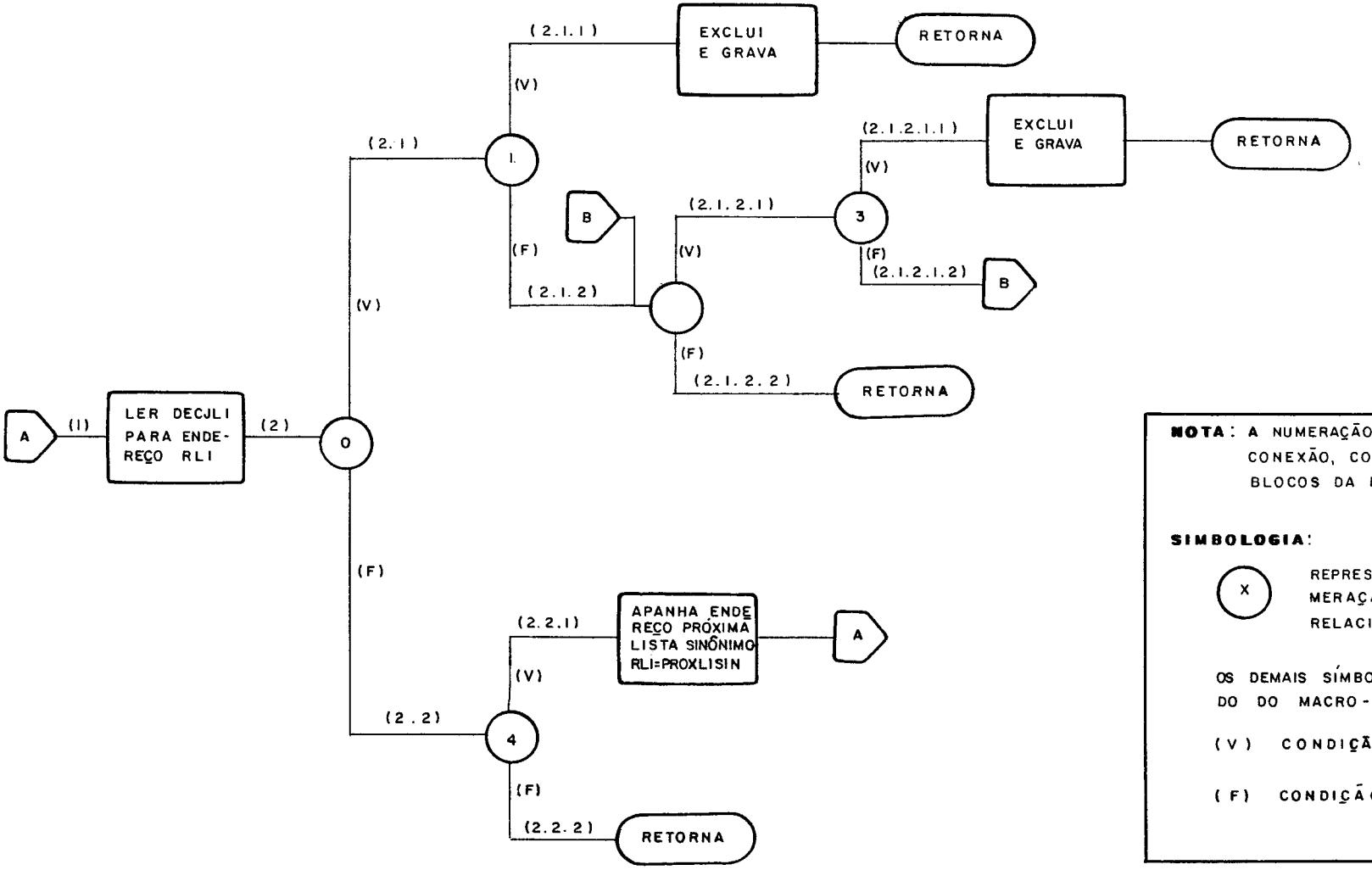
fim;

FIGURA IV. 4

OPÇÃO EXCLUIR

MACRO-DIAGRAMA DA SEQUÊNCIA LÓGICA DE PASSOS

(PARA OCORRÊNCIA "EXISTE LISTA DE LISTAS")



NOTA: A NUMERAÇÃO (X.X.X) SOBRE AS LINHAS DE CONEXÃO, CORRESPONDE À NUMERAÇÃO DOS BLOCOS DA DESCRIÇÃO DO ALGORITMO.

SIMBOLOGIA:

(X) REPRESENTA NÓ DE DECISÃO, CUJA NUMERAÇÃO CORRESPONDE ÀS QUESTÕES RELACIONADAS NA LEGENDA.

OS DEMAIS SÍMBOLOS TEM O MESMO SIGNIFICADO DO MACRO-DIAGRAMA DA OPÇÃO EXCLUIR.

(V) CONDIÇÃO VERDADE

(F) CONDIÇÃO FALSO

. Legenda da figura (IV.4)

- OPÇÃO EXCLUIR

Macrodiagrama da Sequência Lógica de Passos para a Ocorrência "Existe Lista de Lista".

(ou seja, quando a resposta da chamada à ECRATB indicar: CODATB = ≠ FF e RLi > 0 (zero).

Relação das Questões:

0. É a lista procurada? - Esta questão é traduzida pela comparação do valor da chave-secundária com o valor guardado no primeiro registro da lista (exemplo figura (IV.2)).

1. A Chave-primária que se deseja excluir (remover), já se encontra no registro Li?

2. A lista que está sendo tratada tem continuação em registro Li de expansão?

3. Chave-primária que se deseja excluir, encontra-se em registro Li de expansão?

4. Há outra lista para algum valor sinônimo do valor da chave-secundária em tratamento?

Salienta-se que a tentativa de exclusão de uma chave-primária que não exista na sua lista apropriada, não foi encarada como um fato que motive a interrupção do processo. Entretanto, tendo sido aceito como um fato normal, pode ser um recurso em favor da manutenção da integridade e segurança do sistema, pois em caso de repetições de transações por motivo de qualquer natureza, fortuito ou não, a fidelidade das listas em relação ao estado do Arquivo Mestre pode ser mantida.

Vale observar que não foi prevista na opção EXCLUSÃO o retorno de registro Li a lista de espaço disponível, o que poderia acarretar um melhor uso desse espaço. Julgou-se que o ganho em espaço pode ser irrelevante, considerando por um lado o aumento do nível de complexidade do controle do espaço disponível e por outro, o fato de que na grande maioria das vezes a quantidade de exclusões é bem inferior a quantidade de inclusões, ou, pelo menos, de ordem equivalente. Vale salientar como atenuantes, a facilidade oferecida pelo sistema operacional do POTi (SOCO) para acrescentar espaço a um arquivo em disco, bem como eventuais recargas das listas em consequência da necessidade de reajustes nos parâmetros do sistema. Aplicações das opções de uso da rotina (ECiSLi) serão exemplificadas mais adiante, como CONSULTAR no Analisador de Pedido, enquanto INCLUIR e EXCLUIR no processo de manutenção do Arquivo Mestre.

IV.5 Dimensionamento do Arquivo de Listas Invertidas (DECJLi)

O dimensionamento do Arquivo de Listas Invertidas depende de um conjunto de particularidades da aplicação em estudo (sistema em projeto). Em geral, deseja-se balancear (Trade-off) de um lado o tempo consumido em manutenção de listas e em tratamento das mesmas pela aplicação das "operações booleanas" e de outro lado a utilização do espaço reservado ao arquivo (DECJLi) em disco magnético. Esses dois fatores de custo do processo, são fortemente dependentes do comprimento do registro "Li" do arquivo de listas invertidas, sendo duas as causas principais:

- o tempo de manutenção de listas invertidas ou resolução de um "Pedido", cresce com o número de leituras em disco magnético, ou seja, com número de registros necessários a construção das listas de cada par chave-secundária / valor da chave-secundária. Esse número de registros representa a relação entre o comprimento das listas invertidas e o número de chaves-primária que podem ser armazenadas em cada registro;

- considerando que na prática encontra-se listas de comprimento bastante variado, quanto maior for o comprimento fixado para o registro "Li" (para atender ao requisito anterior) pior será o proveitamento da área reservada para o arquivo (DECJLi), ou seja, diminui sua taxa de ocupação.

Assim, ao se querer fixar o comprimento do registro "Li" é necessário, a priori, que se tenha avaliado as restrições de tempo e espaço em disco magnético com respeito à aplicação em estudo. Para se conseguir um bom dimensionamento, não é suficiente possuir sentimento (feeling) do problema, mas conhecer, entre outros dados, o comportamento da distribuição dos comprimentos das listas relativas aos pares chave-secundária / va-

lor da chave-secundária. Alguns aspectos do problema são aqui ressaltados.

Seja:

M - o número de entidades, que corresponde ao número de registro do Arquivo Mestre;

S - o número de chaves-secundárias escolhidas dentre todos os atributos (campos) das entidades;

vj - número de ocorrências de diferentes valores que a chave-secundária de ordem (j) tenha assumido, em seu domínio, entre todas as entidades sobre as quais se mantém registro no Arquivo Mestre. Corresponde à quantidade de listas relativas à chave-secundária (j);

$\overline{l}j$ - comprimento médio das listas relativas à chave-secundária de ordem (j), que é dado por:

$$\overline{l}j = \frac{M}{vj} ;$$

T_L - quantidade total de listas do Arquivo Invertido(DECJLi), dado por:

$$T_L = \sum_{j=1}^{S} vj;$$

Considerando que para cada chave-secundária, tem-se que armazenar em suas listas todas as (M) chaves-primárias do Arquivo-Mestre, sendo "S" o número de chaves-secundárias, o tamanho da lista média do Arquivo Invertido é então:

$$\overline{l} = \frac{S.M}{T_L} ;$$

Caso o analista se veja com uma aplicação cuja distribuição de comprimento das listas tem um comportamento tal que se possa considerar como uniforme (o que é uma situação muito especial), sugere-se adotar um comprimento para o registro "Li" que seja suficiente para armazenar $\overline{1}$ chaves-primárias, por ser uma alternativa favorável as restrições de tempo e rendimento de espaço mencionadas anteriormente.

Para os casos de distribuição não uniforme (mais frequentes) de comprimento de lista, sugere-se analisar tal distribuição construindo uma tabela semelhante a tabela (IV.1), a qual explana-se a seguir;

Suas colunas são:

- Comprimento da Lista (l_i) - significa a quantidade de chaves-primárias armazenadas na lista;

- Quantidade de Lista (Qi) - é o número de listas que possuem o mesmo tamanho, indicado na primeira coluna;

- Quantidade Acumulada de Listas - é a quantidade de listas de tamanho igual ou menor àquele indicado na mesma linha (coluna Tamanho da Lista). Por exemplo, para a linha "j" tem-se:

$$\sum_{i=1}^{j} Qi$$

- Total Acumulado de Chaves-Primárias - é a quantidade de chaves-primárias armazenadas nas listas de tamanho igual ou menor àquela indicada na mesma linha (coluna Tamanho da Lista). Por exemplo, para a linha "p" tem-se:

$$\sum_{i=1}^{p} Q_i \times l_i$$

TABELA IV.1*

Comprimento da Lista (l_i)	Quantidade de Lista (Q_i)	Quantidade Acumulada de Listas	Total Acumulado de Chaves-Primárias
1	Q_1	Q_1	Q_1
2	Q_2	Q_1+Q_2	$Q_1+Q_2 . Q_2$
3	Q_3	$Q_1+Q_2+Q_3$	$Q_1+2.Q_2+3Q_3$
.	.	.	.
.	.	.	.
.	.	.	.
l_p	Q_p	$\sum_{i=1}^{p} Q_i$	$\sum_{i=1}^{p} i.Q_i$
.	.	.	.
.	.	.	.
.	.	.	.
l_n	Q_n	T_L	S.M

A escolha de um valor (l_i) para comprimento do registro "Li", com o uso da Tabela (IV.1), pode ser realizada a partir do estabelecimento de um percentual (x%) do número de listas, que se espera ter comprimento tal que possa ser mantido, por exemplo, em um único registro.

---

* - A Tabela (IV.1) pode ser construída por programa, podendo ser o mais recomendável o próprio que carregar as listas invertidas, ou um programa auxiliar que possa rodar periodicamente.

Assim, esse percentual traduz a esperança matemática $(\frac{x}{100})$ de que uma lista, escolhida ao acaso, possa ser tratada com um único acesso a disco. Portanto, pode-se estimar a quantidade Q(x) esperada de listas contidas em um único registro pela expressão seguinte:

$$Q(x) = \frac{T_L \times x}{100}$$

Feito isto, procura-se na coluna "Quantidade Acumulada de Listas" o valor $(\sum_{i=1}^{p} Q_i)$ que mais se aproxima de Q(x), encontrando-se na mesma linha (p) o comprimento (l_p) que deve ser adotado para o registro "Li", para satisfazer, por aproximação, àquele percentual (x%). Tendo-se o l_p, pode-se calcular a quantidade mínima de registros que deve conter o arquivo DECJLi, a partir dos cálculos parciais das quantidades de listas cujos comprimentos estão nos intervalos conforme mostra a tabela (IV.2) abaixo:

TABELA IV.2

Intervalo de Comprimento de Listas (l_i)	Quantidade de Listas	Quantidade de Registros
$\ell_i \leq \ell_p$	P_1	P_1
$\ell_p < \ell_i \leq 2\ell_p$	P_2	$2P_2$
$2\ell_p < \ell_i \leq 3\ell_p$	P_3	$3P_3$
.	.	.
.	.	.
.	.	.
$(K-1)\ell_p < \ell_i \leq K\ell_p$	P_K	KP_K
TOTAL	T_L	$\sum_{i=1}^{K} iP_i$

Sendo o último intervalo determinado por l_n da Tabela (IV.1) para $(K-1).\ell p < \ell_n \leq K\ell p$

Dessa tabela pode-se tirar que a taxa de ocupação(x) para o total das listas invertidas construídas no arquivo (DECJLi) é:

$$x = \frac{S.M}{\left(\sum_{i=1}^{K} iP_i\right).\ell_p}$$

e que esse mesmo arquivo deve possuir uma quantidade (N) mínima* de registros dada por:

$$N = \sum_{i=1}^{K} iP_i$$

Vale mencionar, que ocorre na prática, não raras vezes, algumas chaves-secundárias aparecem com grande frequência na formulação de Pedidos. Dessa forma, existem mais variáveis para uma escolha minunciosa do comprimento do registro "Li", quando alguém deseja aperfeiçoar o tempo de resposta no processo interativo. Caso haja um pequeno número de chaves-secundárias com alta probabilidade de serem utilizadas na formulação de "Pedidos", na escolha de alterantiva deve-se levar em conta um fator de ponderação (que pode ser a esperança matemática) de modo a tender a reduzir o número de acessos a disco, o que certamente altera o fator de ocupação. Pode ser vantajoso considerar, no estudo de alternativas, apenas aquela ou aquelas chaves-secundárias que detêm a grande maioria das consultas, construindo-se para ela (ou elas) a tabela (IV.1) em adição, como meio auxiliar de análise, já que torna mais fácil a terefa de estimar os resultados para diversos (ℓi).

---

* - Caso o arquivo (DECJLi) esteja carregado, pode-se obter a quantidade de registros ocupados do próprio "header" desse arquivo.

Por fim, considerando que o método apresentado não se trata de um processo determinístico, deve ser enfatizado que o aperfeiçoamento do balanceamento tempo X fator de ocupação deve ser obtido por aproximação sucessiva, não apenas por simulação na época de implantação do sistema, mas por contínua avaliação do seu comportamento em função de mudanças em seu ambiente.

Salienta-se que até aqui, neste capítulo (IV), abordou-se os recursos utilizados para particionar o Arquivo Mestre, bem como o mecanismo de reconhecimento da lista invertida definida pelo par chave-secundária/valor da chave-secundária.

Entretanto, a interpretação de um Pedido formulado segundo a linguagem definida no Capítulo (III), chegando-se até a selecionar as chaves-primárias que integram o conjunto resultante desse Pedido, constitui o assunto da próxima seção (IV.6).

IV.6 Interpretação e Resolução de Pedidos

O processo de interpretação de Pedidos e seleção das chaves-primárias que constituem o conjunto resultante desse Pedido, é executado em duas etapas bem distintas.

Chamou-se a rotina* que executa todo esse processo de "ECPEDi", a qual utiliza-se de duas outras auxiliares para realizar aquelas etapas. A primeira foi implementada com a denominação de "ECANLS", tendo como função interpretar o Pedido escrito na linguagem defina no Capítulo (III), traduzindo-o para códigos dispostos em uma forma pós-fixada.

A segunda recebeu a denominação de "RESOLVE$EXPR", tendo como função executar as sucessivas reduções da expressão booleana, na forma pós-fixada construída pela etapa anterior, até encontrar o conjunto de chaves-primárias resultante do Pedido.

Essas rotinas são descritas, de forma condensada, a seguir:

IV.6.1 Descrição da Rotina "ECPEDi"

Essa é a rotina do modelo implementado que encerra todas as facilidades de comunicação com o usuário, para recuperação de informações por múltiplas-chaves. É simples de ser usada, bastando ao analista da aplicação conhecer as variáveis globais que a mesma requer, bem como o procedimento para utilizá-la, ao nível que se passa a discorre a seguir:

- VETRTB - (global) vetor sobre o qual é carregada a Tabela de Definição do Registro Mestre, mencionado na seção (IV.1).

---

* - As denominações das rotinas aqui referenciadas são aquelas que identificam suas respectivas "PROCEDURE's" escritas na linguagem PLTi, com cópia em anexo.

- ARA133 - (global) área de 133 "bytes" utilizada para impressão de mensagens;

- BMT - (Address global) indica endereço da área de leitura o Registro Mestre, já mencionado;

- BSRESM - (address global) indica o endereço da área onde é deixado o conjunto de chaves-primárias resultante do Pedido, área essa utilizada como interface com a rotina de recuperação direta ao Arquivo Mestre.

- POSFIX - (address global) vetor address definido para conter o Pedido na forma pós-fixada, utilizado como entrada para a segunda etapa do processo.

- ARQUIVOS - os arquivos utilizados pelas diversas etapas dessa rotina, os quais devem ser declarados no módulo principal do programa que chamar a rotina ECPEDi, são os seguintes:

  . DECJTB e DECJLi - arquivos em disco, já definidos anteriormente, os quais correspondem a Tabela de Definição do Registro Mestre e ao Arquivo de Listas Invertidas, respectivamente;

  . VIDE08 - arquivo em vídeo para exposição (display) de mensagens, em 80 (oitenta) posições (sem caracter de controle de espacejamento);

  . DECJRS - arquivo em disco, com 20 (vinte) registros de 512 bytes, utilizado como área auxiliar no processo de resolução da expressão booleana, na forma pós-fixada, de competência da rotina "RESOLVESEXPR";

  . TECLAD - arquivo em teclado, para digitação das respostas do usuário;

. IMPRES - arquivo em impressora, para impressão de linha de 132 posições, definida em área com 133 posições, sendo a primeira para controle do espacejamento vertical.

- Basicamente a rotina* ECPEDi se constitui de duas chamadas, à ECANLS e à RESOLVE$EXPR, na sequência descrita a seguir:

```
ECPEDi ; PROCEDURE;
   .          .
   .          .
   .          .

1 - Chama ECANLS, ou seja:

    | CALL ECANLS |

2 - Chama RESOLVESEXPR; ou seja:

    | CALL RESOLVESEXPR (ERRO); |

3 - Se erro dá mensagem de erro e volta ao passo 1;

    caso contrário

4 - Retorna; (ao ponto de chamada)

fim.
```

A descontinuidade do processamento, só pode se dar na primeira etapa (ECANLS), através de instrução do usuário nesse sentido.

---

* - Ver em anexo essas mesmas rotinas escritas em PLTi, caso deseje mais detalhes.

IV.6.2 Descrição da Rotina (ECANLS)

Essa rotina é interna à ECPEDi, sendo por sua vez constituído de dois módulos básicos. Um reconhece os operandos (representado nas expressões booleanas por condições atômicas), traduzindo-os em códigos sob a forma de endereços de listas invertidas.

O outro módulo reconhece os operadores booleanos, bem como o fim de expressão, traduzindo-os em códigos de 2 bytes (forma address). Os parênteses, utilizados para alterar as prioridades de resolução segundo as regras da linguagem definida no Capítulo (III), são interpretados seguindo essas mesmas regras.

Essa rotina é constituída dos seguintes passos:

1 - solicita Pedido, enviando a mensagem "TECLE SEU PEDIDO OU DESCONTINUI (D)";

2 - lê resposta do usuário:

2.1 - se for "D" descontinua o programa (EXIT);

3 - se o primeiro operando for "PA" (Pedido Anterior) mova o código (-1000) para área POSFIX e salte para o passo (5);

4 - chama rotina que analisa operando:

| CALL QUALIFICADOR (ERRO); |;

4.1 - se houver erro dá mensagem:

"ERRO-ESPERADO QUALIFICADOR", e volta ao passo (1);

5 - chama rotina que analisa operador:

| CALL OPERADOR (ERRO) |;

5.1 - se houve erro dá mensagem:

"ERRO-ESPERADO OPERADOR", e volta ao passo (1);

5.2 - se fim de Pedido retorna; (ao ponto da chamada-expressão correta)

6 - volta ao passo (4);

fim.

O esquema* adotado para transformar as expressões da forma in-fixada para pós-fixada é de fácil assimilação, por isso mesmo bastante divulgado. O controle de prioridade dos operadores, bem como das subexpressões parentesadas, é exercido com o uso de uma pilha onde são retidos temporariamente operadores de menor prioridade e sinais de "abre parênteses" "(".

Convencionou-se que a codificação de operandos e operadores na forma pós-fixada seria em variável address (hexadecimal), cujos valores tem os seguintes significados:

infixada		pós-fixada
Operador AND (representado por "&")	→	(-1 ou # FFFF);
Operador OR (representado por "!")	→	(-2 ou # FFFF);
Operador NOT (representado por "N")	→	(-3 ou # FFFD);
Fim de Pedido(representado por "?")	→	(# FFFØ);

---

* - Descrições detalhadas sobre o mencionado algoritmo encontram-se em FORSYTHE[36] e GAUTHIER[37].

infixada		pós-fixada
Operandos que representam condições atômicas na forma "código chave-secundária = valor"	→	endereço da lista cujo valor pode variar de zero a 32.767, sendo zero para lista vazia.
Operando "PA" (Pedido Anterior)	→	(-1000) que significa registro zero do arquivo DECJRS.

Para efeito de compatibilidade, o sinal "abre parêntese" quando guardado na pilha recebe o código (-6).

Alguns detalhes dos módulos de tratamento dos operandos e operadores são ressaltados a seguir, com o auxílio do exemplo abaixo:

Exemplo de Pedido:

```
                operador          operador          fim de expressão
                   ↓                 ↓                   ↓
  AØ3 = 512  &  (AØ4 = TB12  |  AØ4 = TB15)              ?

  Operando 1          Operando 2        Operando 3
      ou
Condição Atômica                subexpressão
```

Imagine-se que os endereços em hexadecimal dos inícios das listas correspondentes a cada um desses operandos sejam:

Operando 1 endereço # ØØ5Ø

Operando 2 endereço # ØØ5A

Operando 3 endereço # ØØA1

Pedido na forma pós-fixada

ØØ5Ø	ØØ5A	ØØA1	FFFE	FFFF	FFFØ	

. Análise de Operando

A análise de operandos tem início tomando-se o código da chave-secundária nele representada e consultando a Tabela de Definição do Registro Mestre, a fim de verificar a sua validade.

Nesse instante pode ser constatado dois tipos de erro* - código não declarado na Tabela ou existe o código, mas não foi definido como chave-secundária. Da Tabela também são obtidos a localização desse campo chave-secundária e seu comprimento dentro do Registro Mestre, para onde é movido o valor que compõe a condição-atômica, caso não tenha sido observado algum erro até esse ponto da expressão lógica.

Para obter o endereço da lista invertida correspondente ao par chave-secundária/valor da chave-secundária definido pela condição-atômica em análise, utiliza-se a rotina "ECISLI" para a opção CONSULTAR, atribuindo-se aos parâmetros os valores seguintes:

CODATB = número de ordem da chave-secundária; e

ALTER = 'C' - assim identificado o par que define a lista invertida desejada, pode-se evocar a rotina "ECISLP", já descrita:

[CALL ECISLI (RLI, CODATB, RII, ALTER);].

Caso o processamento tenha sido bem sucedido, o que é reconhecido pela condição (CODATB = ≠ FF), o endereço da lista invertida desejada encontra-se em RLI, o qual é acrescentado à área onde é construída a expressão pós-fixada, na posição apropriada.

---

* - Todos os erros de sintaxe são assinalados com "*" na posição em que foi detectado, sob o Pedido original.

Caso seja indicado um processamento mal sucedido, motivado por exemplo por valor indevido, retorna ao ponto de chamada com a indicação de ERRO.

. Análise de Operador

Essa rotina tem as atribuições seguintes:

- traduzir para a expressão pós-fixada a mesma sequência de reduções estabelecida pela forma in-fixada do Pedido. Isso significa determinar a posição apropriada de cada operador dentro da área reservada para a construção da expressão pós-fixada;

- reconhecer os símbolos operadores e fim de Pedido, segundo a definição da linguagem, e convertê-los aos respectivos códigos em hexadecimal;

- identificar erros na formulação do Pedido, tais como:

  . Código de operador não identificado;

  . Pedido incompleto, faltando "fecha parêntese" ou sinal de fim de Pedido.

Caso o processo seja bem sucedido, retorna ao ponto de chamada com o código (ERRO = # FF); em caso contrário o código de erro depende da ocorrência, quais sejam:

ERRO = 1 → não encontrado fim de Pedido;

ERRO = 7 → esperado operador ou "fecha parêntese";

ERRO = 8 → capacidade da área pós-fixada foi excedida;

ERRO = 9 → fim de Pedido inesperado, provavelmente falta "fecha parêntese".

IV.6.3 Descrição da Rotina RESOLVE$EXPR

Essa é uma rotina interna à ECPEDI, que tem como função (quando ECANLS conclui com sucesso a transformação do Pedido para a forma pós-fixada) executar as sucessivas reduções da expressão booleana, até encontrar o conjunto de chaves-primárias resultante do Pedido. Em linhas gerais, o processo consiste em percorrer a expressão pós-fixada empilhando os operandos, até que seja reconhecido um operador. Aí então, procede-se a redução conforme a função booleana, simbolizada pelo operador, sobre os dois últimos operando empilhados. A resultante da resolução de ordem "n" dentro da expressão é guardada no registro de ordem* (n+1) (ou endereço n em PLTi) do arquivo "DECJR$" e o endereço dessa resultante é colocado no topo da pilha de operandos. Para diferenciar dos operandos representados por endereços de listas invertidas o número de ordem "i" da resolução é empilhado na forma (-1000-i), para uso em resolução posterior.

Essa rotina pode ser descrita pelos seguintes passos:

1 - inicializa o CURSOR da área pós-fixada e TOPO da pilha com zero e número de resolução com (-1000);

[CSPOSFIX = Ø;
  TP = Ø;
  NRESOLUÇÃO = -1000;]

2 - percorre área pós-fixada até encontrar um não operando: sendo operando faz

2.1 - incrementa TOPO [TP = TP + 1;]

2.2 - coloca operando no topo da pilha;

---

* - Para o disco do POTI o primeiro registro tem endereço zero.

2.3 - incrementa CURSOR

[CSPOSFIX = CSPOSFIX + 1;]

2.4 - fim do bloco (2);

3 - se fim de expressão salte para o bloco (8);

4 - retira do topo da pilha os dois últimos operandos colocando-os em OPERANDO B e OPERANDO A mantendo a precedência;

5 - executa redução aplicando operador booleano conforme o código pontado pelo cursor;

```
[ DO CASE (I = AREAPOSFIX (CSPOSFIX) + 3);

      CALL  ROTNOT;
      CALL  ROTOR;
      CALL  ROTAND;

  END   DO-CASE; ]
```

6 - grava RESULTANTE da redução no registro (-NRESOLUÇÃO - 1000) do arquivo DECJRS; e

incrementa CURSOR;

[CSPOSFIX = CSPOSFIX + 1;]

7 - volta para o bloco (2);

8 - grava RESULTANTE da última resolução no registro zero do arquivo DECJRS; e retorna ao ponto de chamada;

fim;

O processo de redução pela aplicação dos operadores booleanos (AND, OR e NOT) é executado utilizando-se duas áreas auxiliares de 512 bytes cada, sendo os dois primeiros bytes reservados como contador de chaves-primárias inseridas, e os 510 bytes restantes destinados a conter as próprias chaves-primárias envolvidas no processo (essas áreas denominou-se de RESULTANTE e RESULTAUX).

O resultado de toda redução é sempre deixado em RESULTANTE.

Para qualquer dos três operadores booleanos é necessário prever as alternativas possíveis de valores que os operandos podem assumir, quais sejam:

- valor zero → lista invertida vazia;
- valor maior que zero → lista invertida não vazia; e
- valor menor que zero → resultante de redução anterior.

Existe um procedimento padrão para recuperar todas as chaves-primárias de uma lista invertida e reuní-las em uma das áreas, RESULTANTE ou RESULTAUX.

1 - prepara a área receptora inicializando contador de chaves-primárias com zero e movendo brancos para os demais 510 bytes.

2 - faz para toda chave-primária existente na lista invertida:

2.1 - aplica função hashing por dobramento (já mencionada) transformando a chave-primária em uma sucessão de bits representada na variável address KHASH; em seguida aplica função hashing por divisão para KHASH positivo:
[KHASH = ((KHASH MOD 510) (LCHAV)* LCHAV;]
onde 510 é o módulo da área onde se deseja mapear

e LCHAV é o comprimento da chave-primária (em KHASH resulta zero ou um múltiplo de LCHAV no intervalo 0 ≤ KHASH < 510

2.2 - procura na área (RESULTANTE ou RESULTAUX) um espaço vazio de comprimento LCHAV a partir da posição KHASH, considerando essa área como uma lista circular (o curso recebe incrementos de LCHAV);

2.3 - caso a área esteja cheia emite mensagem, e retorna ao ponto de chamada;

2.4 - move chave-primária para área com comprimento LCHAV;

2.5 - fim do bloco (2);

3 - retorna ao ponto de chamada;

fim.

Por julgar desnecessário tecer maiores detalhes, faz-se apenas alguns esclarecimentos sobre cada processo de redução relativos aos operadores booleanos.

. Processo de Redução pelo Operador AND

A rotina em PLTI para esse processo, deu-se o nome de ROTAND.

1 - verifica se algum dos operadores diz respeito a lista vazia; visto que resultaria em um conjunto vazio, poder-se-ia encerrar o processo de redução e retornar;

2 - caso exista um dos operandos que seja < zero, esse deve ser atribuído ao OPERANDOA (trata-se de resultante de resolução anterior);

3 - recupera OPERANDOA para área RESULTANTE;

3.1 - caso se trate de resultante de resolução anterior, deve simplesmente ler o respectivo registro do arquivo DECJRS para área RESULTANTE;

3.2 - caso se trate de lista invertida, aplica o algoritmo anterior de recuperação de chaves-primárias;

4 - para toda chave-primária pertencente ao conjunto definido pelo OPERANDOB é verificado se também pertence ao conjunto definido por OPERANDOA; em caso afirmativo tal chave-primária é mantida em RESULTAUX (usa mesmas funções hashing);

5 - copia RESULTAUX em RESULTANTE;

6 - retorna ao ponto de chamada;

fim.

. Processo de Redução pelo Operador OR

A rotina em PLTI para esse processo, deu-se o nome de ROTOR.

Inicialmente é verificado se ambos os operandos se referem a listas vazias, pois em caso afirmativo, pode-se dar por encerrado o processo. Do contrário as chaves-primárias pertencentes ao conjunto definido pelo OPERANDOA são recuperadas e mantidas na área RESULTANTE. Em seguida, a cada chave-primária do conjunto definido por OPERANDOB é aplicado o mesmo esquema de endereçamento por hashing. A partir do endereço gerado, é procurada na área RESULTANTE chave-primária de mesmo valor. Caso não exista (espaço em branco) é inserida em RESULTANTE a chave-primária do OPERANDOB e incrementado o contador de chaves-primárias dessa mesma área.

Findo tal conjunto de chaves-primárias é dado por encerrado o processo e o controle é retornado ao ponto de chamada.

. Processo de Redução pelo Operador NOT

Chamou-se de ROTNOT à rotina em PLTI para esse processo. O uso desse operando significa que se deseja todas as chaves-primárias pertencentes ao conjunto definido pelo OPERANDOA, mas, que não pertença ao conjunto do OPERANDOB.

Em consequência, verificando-se de início que o OPERANDOA define um conjunto vazio, o processo é dado por encerrado. Caso contrário, recupera-se o conjunto do OPERANDOB mantendo-o na área RESULTAUX. Em seguida, a cada chave-primária do conjunto definido por OPERANDOA é aplicado o mesmo esquema de endereçamento por hashing. A partir do endereço gerado, é procurado na área RESULTAUX chave-primária de mesmo valor. Caso não exista (espaço em branco) é inserida* (ou mantida) em RESULTANTE a chave-primária do OPERANDOA e incrementado (ou inalterado) o contador de chaves-primárias dessa mesma área.

---

* - Inserida se chave-primária do OPERANDOA estiver sendo recuperada diretamente de lista invertida.

V. MÓDULO DE RECUPERAÇÃO DE REGISTRO

V. MÓDULO DE RECUPERAÇÃO DE REGISTRO

A organização do Arquivo de Dados (Arquivo Mestre), bem como o seu mecanismo de recuperação de registro relativo a uma chave-primária dada, independem do esquema adotado para o Módulo Interpretador de Pedido e Seleção de Chaves-Primárias.

Lembramos que não existe um processo determinístico geral para se projetar ótimas organizações de arquivo. Entretanto, um conjunto de regras práticas pode auxiliar ao analista experiente a descobrir novas abordagens de projeto, além de sua vivência pessoal, bem como ajudar a um inexperiente a evitar projetos de pobres desempenhos. CARLIS e SEVERANCE[27] conceberam um conciso método de abordagem (framework) em que são expostos os essenciais princípios a considerar na elaboração de um projeto de "base de dados", sendo de fácil compreensão para sua efetiva aplicação. Tal trabalho "não pretende ser um manual para projetos de "base de dados" (data base)", dizem eles.

Entende-se aqui por "base de dados", de maneira simplificada mas não conflitante com definições mais completas, como sendo não apenas um depósito de informações, e sim uma coleção organizada de conjuntos de descrições de fatos relativos a entidades perceptíveis ao ser humano (pessoas, peças de máquinas, lugares em linhas aéreas, etc.). Um processo de delineamento de "base de dados" procura determinar a organização de dados que atenda alguns critérios de desempenho dentro das restrições conhecidas.

---

* - Alguém interessado em maiores detalhes sobre o assunto pode encontrar úteis e testados princípios reunidos nas obras LEFKOVITZ[4], MARTIN[6], YOURDON[7] e DATE[8], sem querer desmerecer outras.

São quatro os fatores básicos que definem tal problema:

- Os dados a serem armazenados - seus tamanhos, volume e volatilidade;

- A variedade de solicitação que condicionam a recuperação de dados - suas frequências, prioridades, requerimentos de respostas e critério de seleção de dados;

- O ambiente de armazenamento de dados - seus custos de operação e características de acesso; e

- O objetivo do delineamento do sistema - uma medida de desempenho combinada com um conjunto de restrições operacionais.

Vale observar que grande parte das informações requeridas por esses fatores devem ser providas pelas conclusões do primeiro estágio do delineamento de um sistema de informações, definido como "Problema Infológico" por Börge LANGEFORS[19], já mencionado.

Apesar do grande número de combinações possíveis com esses quatro fatores, são relativamente poucas as estratégias básicas de acesso a dados utilizadas para delinear uma "base de dados", embora variações de certas estruturas básicas possam representar múltiplas estruturas de arquivo. Entretanto, as características dos problemas e soluções afetam o desempenho do sistema de um modo complexo, cujas sutilezas de suas relações dasafiam à consecução de uma análise exata.

Ao delinear o Arquivo Mestre, tinha-se a considerar o carater interativo do Sistema de Recuperação de Informação em estudo, voltado para equipamento de porte semelhante ao (PDP11), contando com algumas de suas facilidades. Em circunstâncias como essa, tornam-se fatores de suma importância o aproveitamento de área em memória de acesso direto, bem como o tempo consumido em recuperação de registros desse arquivo, apesar de

que, sobretudo relativo a tempo, só o conhecimento da aplicação possibilita estabelecer limites de desempenho.

Diante dessas considerações optou-se por uma variação da organização com encadeamento em área separada, a qual usa o conceito do que se chamou de "Tabela de Espalhamento*", SOUZA[10].

Talvez a mais antiga referência a essa organização tenha sido MORRIS[34], a qual denominou de "Scatter Index Table", posteriormente comentada por SEVERANCE[28].

Nessa organização o método de endereçamento requer o uso de dois arquivos, como representado na figura (V.1). Um deles é o "Arquivo de Dados", no qual os registros de dados são armazenados e onde as colisões geradas pela função de endereçamento (hashing function) são resolvidas. O outro arquivo é uma "Tabela de espalhamento", que também chama-se aqui Tabela Índice, onde a função de endereçamento "mapea" as chaves-primárias dos registros de dados. Cada entrada ativa da tabela é um apontador para um registro do arquivo de dados, a qual só é inicializada após aquele registro ter sido armazenado em algum local convenientemente designado. Pode-se dizer, então, que cada entrada da tabela de espalhamento é um apontador ou cabeça de lista de sinônimos de registros de dados, podendo essa lista, em determinado instante, encontrar-se vazia (entrada com valor zero).

A especial vantagem desse processo é tornar a localização física de um registro de dados independente do endereço estabelecido pela função de endereçamento e portanto do seu identificador.

Dessa propriedade pode-se apontar como relevante a possibilidade de aproveitamento total da área destinada ao arquivo de

---

* - Na figura (V.1) chamou-se de Tabela Índice.

FIG. V.1

MÓDULO DE RECUPERAÇÃO DE REGISTROS DE DADOS

TABELA ÍNDICE (MÓDULO=10)

0	1	2	3	4	5	6	7	8	9	
R1	R4	R5	R7	00	R8	00	00	R11	00	

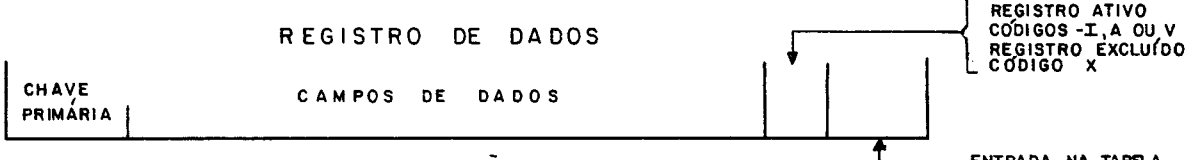
ESPAÇO DISPONÍVEL

R15	38

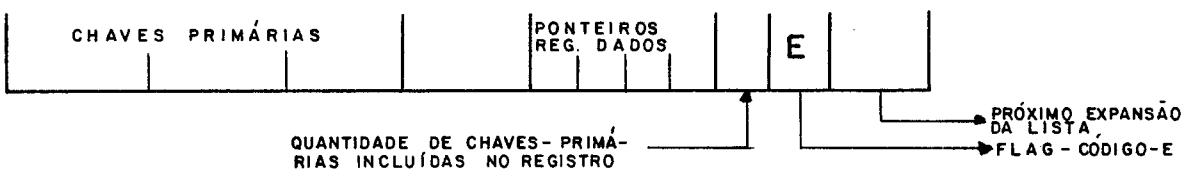
ARQUIVO DE DADOS

R1 0900/JOSÉ 100	R2 0910/ANTONIO 100	R3 0080/BRUNA 100	R4 0501/GABRIELA 101
R5 0502/JOÃO 102	R6 0522/ENIO 102	R7 0603/MARCOS 103	R8 0015/MARIA 105
R9 0105/LUZIA 105	R10 0215/LUIZ 105	R11 0018/MANOEL 108	R12 0068/FABRÍCIO 108
R13 0098/RITA 108	R14 1028/BRENO 108	R15	R16
R17	R18	R19	R20
R21	R22	R23	R24
R25	R26	R27	R28
R29	R30	R31	R32
R33	R34	R35	R36
R37	R38	R39	R40
R41	R42	R43	R44
R45	R46	R47	R48
R49	R50	R51	R52

REGISTRO DE DADOS



REGISTRO EXPANSÃO DE LISTA



dados, armazenando-se serialmente (em posições consecutivas) os registros de dados pela ordem cronológica de chegada. Isso contrasta com o disperdício de espaço causado pelos hiatos gerados pelas funções de endereçamento, quando adota-se o endereçamento direto ao arquivo de dados, obrigando a que o fator de ocupação não se aproxime de 1 (um), pois nesse caso o número de colisões adquiriria uma alta taxa de crescimento. Na grande maioria das vezes é impraticável encontrar uma função que gere endereços segundo uma distribuição uniforme, sem colisões, sem hiatos dentro do domínio de endereços do arquivo de dados limitado ao espaço que se dispõe, a partir de chaves-primárias de valores aleatórios ou mesmo com algum vício. Por esse esquema o disperdício pode se tornar intolerável quando o registro de dados atinge grande dimensão.

Considerando que as entradas da tabela índice podem ser formadas apenas de uma palavra do computador (uma variável address em PLTi), para um mesmo espaço tem-se uma quantidade maior de entradas, resultando em listas de sinônimos muito mais curtas.

A tabela de espalhamento tem duas aparentes desvantagens:

- um adicional espaço requerido pela tabela; e
- um adicional acesso à tabela, o qual é requerido para toda recuperação de dados.

Com relação ao espaço, em geral se obtém um ganho ao comparar o tamanho da tabela com o espaço salvo no arquivo de dados, pela alocação serial dos registros, conforme mencionou-se há pouco.

Quanto ao custo total de acesso, há situações em que a tabela pode ser mantida na memória, levando a que se obtenha número de acesso médio menor comparado a outra alternativa, quando nessas situações as listas de sinônimos são mais curtas.

SEVERANCE[28] mostra um exemplo para o qual o uso da Tabela de espalhamento é vantajoso em espaço e acesso relativos à memória secundária (disco no caso aqui tratado).

Ainda, a tabela de espalhamento pode ser de grande utilidade quando se deseja recuperar diretamente por mais de um identificador do registro de dados. Como por exemplo, em um arquivo de empregados alguém desejasse recuperar dados utilizando a matrícula ou o número do CIC (CPF) da Receita Federal.

Nesse caso, naturalmente para cada chave de recuperação se teria que ter uma tabela e listas de sinônimos independentes.

Por fim, vale destacar que o esquema em Tabela de Espalhamento permite que se trabalhe com registros lógicos de comprimento variável, ainda como consequência da localização física independer da chave-primária (identificador). Esse artifício, em muitas aplicações, pode contribuir para salvar uma quantidade maior de espaço da memória secundária, quando se comparando com a alternativa de acessar diretamente ao arquivo de dados, pois, nesse caso, os registros obrigatoriamente são definidos com dimensão igual ao tamanho máximo requerido.

V.1 Descrição do Arquivo "Tabela de Espalhamento" ou "Tabela Índice" (DECJØ1)

O Arquivo "Tabela Índice" tem como finalidade manter os ponteiros ou cabeças de listas de sinônimos, construídas no "Arquivo de Dados", como foi dito no início deste capítulo. Dada sua importância no processo de manutenção de recuperação de registros de dados, ele foi implementado (como nome externo DECJØ1) com registros de 512 bytes (1 setor) sendo o primeiro (registro zero) denominado header o qual tem a função de identificar o arquivo e conter várias informações de controle atualizados. Os demais registros são destinados a conter ponteiros, cujo número máximo de entradas é 255.

Cabe ao usuário, em um dos primeiros passos da implantação de uma aplicação, formatar* a Tabela Índice, fornecendo como parâmetro a dimensão da Tabela. As informações mais relevantes estão explicitadas a seguir:

. Dados do Registro Header

TABGERAÇÃO - é o número de ordem da geração da Tabela; quando o arquivo é formatado inicialmente, o número dado é 1 (um);

TFLAG - indica se houve alteração na dimensão da Tabela; código A-alterado recentemente; código V - estado antigo, significando que todos os passos de inicialização foram executados;

---

* - Para isso utiliza-se o programa ECCJØ1, o qual funciona em um processo interativo com o usuário.

BLOCAGEM - é o número de entradas, desejadas, por bloco ou registro físico da Tabela, cujo valor máximo é igual 255.

COMPRIMENTO - é o comprimento da Tabela ou a quantidade total de seus ponteiros (foi implementado com o máximo de 5.865 entradas para 23 blocos);

QREGISTRAB - quantidade de registros físicos requeridos para conter toda tabela; é calculado por | COMPRIMENTO ÷ BLOCAGEM | + 1 incluindo o registro zero, sendo que se a divisão for exata, arredonda-se para o inteiro imediatamente superior;

PONTDISP - apontador do início do espaço disponível do Arquivo Mestre (foi implementado apontando para o último registro cedido, sendo inicializado com zero;

QUANTDISP - quantidade de Registros de Dados a ser cedido às listas de sinônimos pelo espaço disponível (inicializado com o número total do Arquivo Mestre menos um);

QUANTEXP - quantidade de registros utilizados como expansão de lista de sinônimos;

DATAUTIL - data da última utilização do espaço disponível (DD/MM/AA);

DATALTER - data da última vez em que foi definida ou alterada a dimensão da Tabela Índice; e

NREGTABH - número de ordem do registro, que sendo o "header" é igual a zero.

Esse registro "header" possui 434 bytes não utilizados, dos quais se pode dispor para inclusão de novos campos caso seja desejado.

. Dados do Registro de Ponteiros (REGPONT)

PONTEIRO - é um vetor de 255 entradas (address) utilizadas como ponteiro ou cabeça de lista de sinônimo; e

NREGTAB - é o número de ordem o registro de ponteiro.

. Método de Endereçamento ao Arquivo Tabela Índice

Como foi dito na explanação inicial deste capítulo, o endereçamento à entrada apropriada da Tabela Índice para uma dada Chave-primária é realizado através do cálculo da posição relativa dessa entrada ao início da Tabela.

Esse cálculo é feito pela aplicação de uma função de transformação chave-a-endereço, chamada função de endereçamento, a qual transforma cada valor do domínio da chave-primária, "mapeando" sobre a região de endereços possíveis da Tabela.

Para a escolha da função de endereçamento, são válidas as mesmas considerações da seção (IV.2), sendo que aqui se trata com chave-primária em lugar de chave-secundária.

Lembra-se que, mesmo sendo numérica* a chave-primária escolhida para a aplicação, é necessário o analista observar se em seu domínio existem valores superiores ao limite (32.767) de

---

* - Na aplicação simulada adotou-se chave-primária numérica, com valores não superiores a 32.767.

valores positivos que uma variável address pode representar em hexadecimal no POTi. Em caso afirmativo, antes do mapeamento sobre a região de endereços possíveis da Tabela Índice, o valor da chave-primária deve ser normalizado dentro daquele domínio da variável address, podendo ser adotado o mesmo procedimento descrito na seção (IV.2).

Para função de endereçamento optou-se pelo método de "Divisão" tendo como razões principais:

- em uma aplicação de base de dados, o tempo requerido para desenvolver a transformação de chave-a-endereço é insignificante em comparação ao tempo requerido para acessar o registro (ou bloco) endereçado; e

- estudos de LUM[31] indicam que a técnica relativamente simples da função de endereçamento por "Divisão" prover um bom desempenho geral, a qual é ligeiramente equivalente àquela esperada de uma transformação com a distribuição uniforme.

O cálculo do endereço E(Ki) para a chave-primária K_i, pela técnica de "Divisão", resume-se em determinar o resto da divisão tendo como módulo o COMPRIMENTO da Tabela Índice definida no registro "header". Assim, pode-se ter a expressão:

$E(K_i) \leftarrow K_i$ mod COMPRIMENTO;

Para localizar esse endereço na Tabela Índice é necessário determinar qual registro do arquivo "DECJØ1" deve ser lido, sendo seu número de ordem $R(K_i)$ dado por:

$$R(K_i) \leftarrow \lfloor E(K_i) \div \text{BLOCAGEM} \rfloor + 1 \quad (**);$$

dentro desse registro, a entrada que corresponde ao apontador

---

** - Os símbolos $\lfloor\ \rfloor$ representam arredondamento para o inteiro imediatamente inferior.

da lista a que pertence a chave-primária (K_i), é localizado na posição Ap (K_i) dada por:

$$A_p(K_i) \leftarrow E(K_i) \quad \underline{mod} \quad \text{BLOCAGEM};$$

Só então, pode-se efetivamente consultar a Tabela Índice da chave-primária (K_i).

V.2 Descrição do Arquivo Mestre (DECJ2Ø)

O Arquivo Mestre ou Arquivo de Dados tem como finalidade manter as informações relevantes das entidades relacionadas com aquela aplicação em estudo, com vistas as necessidades do usuário final.

Como representado na figura (V.1), esse arquivo é carregado inicialmente de forma a que todos os registros de uma mesma lista de sinônimos ocupem posições contíguas dentro do mesmo arquivo. Posteriormente, as inserções de novos registros em uma lista de sinônimos são realizadas, tendo-se como recurso o uso de um registro de extensão, o qual aponta ao local onde efetiva-se a desejada inserção, local esse designado pelo controle de espaço disponível.

Esse arquivo se constitui de três registros, o header, registro de dados e registro de extensão, os quais passa-se a descrever:

. Dados do Registro Header

Esse registro tem como função manter informações necessárias ao controle e segurança de sua compatibilidade com a Tabela Índice.

Em suas primeiras posições é conveniente manter informações para a identificação do arquivo. Os demais campos são:

DIMENSAOM - quantidade de registro criados para o arquivo DECJ2Ø, incluindo o registro header;

TTALREGDADO - total de registros criados para o espaço disponível, utilizados pelos registros dados;

NGERAMEST - número de ordem da geração do Arquivo Mestre, sempre incrementado quando há atualização do mesmo;

NGERATABR - número de geração da Tabela Índice que o indexa;

MODCOMPRIMTAB - comprimento da Tabela Índice ou módulo para cálculo de endereçamento às entradas da Tabela; e

BLOCATABM - número de entradas por registro do Arquivo Tabela Índice.

As demais posições são livres para o analista fazer o uso que desejar.

Registro de Dados

Os campos desse registro são definidos na Tabela de Definição do Registro Mestre, sendo eles de três tipos básicos:

- Chave-primária, com base no identificador da entidade, escolhido;

- Dados, sendo alguns escolhidos como chave-secundária; e

- Campo controle, o qual contém o endereço relativo da entrada (ponteiro) na Tabela Índice para ele designada pela função de endereçamento, com base em sua chave-primária.

CAMPO FLAG - indica o estado do registro sendo (I), registro recém incluído, requerendo que chaves-secundárias sejam incluídas nas listas invertidas;

(A), registro recém alterado, com listas invertidas não atualizadas;

(V), registro com listas invertidas atualizadas;

(X), registro excluído logicamente.

. Registro Expansão de Lista

É utilizado para possibilitar a inserção de registros em uma lista de sinônimos, em posições não contíguas com essa mesma lista, após a carga (inicialização ou reorganização) do Arquivo Mestre. O primeiro registro de expansão de uma lista é a ela associado pela sua localização na última posição da lista, além de guardar chaves-primárias que geram o endereço e seu ponteiro de início. Os demais registros de expansão de uma lista são encadeados entre si. Seus campos são:

CHAVEXPAN - é um vetor onde são guardadas as chaves primárias dos registros apontados por esse registro de expansão;

PONTEXPAN - é um vetor onde são guardados os correspondentes apontadores aos registros inseridos posteriormente à carga do Arquivo Mestre;

QUANTEXPAN - indica a quantidade de registros de dados apontados pelo registro de expansão;

FLAGEXPAN - código "E" que identifica um registro expansão; e

ENDCONTINUA - endereço do próximo registro de expansão da mesma lista. Quando contém valor zero indica que não há um próximo registro de expansão.

. Métodos de Endereçamento ao Arquivo Mestre

São quatro os processos para se selecionar com os registros de dados do Arquivo Mestre:

- Consulta;
- Inserção;
- Alteração; e
- Exclusão.

Veja-se a seguir como procedem tais funções do Sistema.

. Consulta

Essa rotina recebeu a denominação de ROTRECUPERAR quando programada em PLTi. Dada uma chave-primária, ela tenta localizar no Arquivo Mestre um registro que tenha idêntica chave-primária. Caso exista, o mesmo é deixado na área de leitura (REGMEST), retornando ao ponto de chamada o endereço de leitura desse registro, no parâmetro utilizado nessa chamada. Caso não tenha sido localizado, é passado em um dos parâmetros o carater "*" e em outro parâmetro é colocado zero quando a lista é vazia e quando não é colocado o endereço do último registro lido pelo qual foi concluída a busca sem sucesso.

Essa rotina pode ser resumida nos seguintes passos, considerando que tenha sido fornecida uma chave-primária "K" qualquer (denomina-se "K" de argumento de pesquisa:

CONSULTA

1 - calcula o endereço na Tabela Índice do ponteiro de lista de sinônimos para o valor da chave-primária fornecida;

2 - se lista está vazia, atribui aos parâmetros a indicação de busca sem sucesso com lista vazia, e retorna;

3 - caso a lista não seja vazia, caminha sobre podendo ocorrer:

3.1 - encontrar registro de dados com chave-primária de valor igual ao argumento de pesquisa "K"; atribui ao parâmetro endereço do registro encontrado e retorna;

3.2 - não encontra registro tendo percorrido toda a lista; atribui o endereço do último registro lido e "*" a cada parâmetro; e retorna.

fim.

As três outras rotinas são iniciadas com uma chamada a CONSULTA, a fim de localizar o registro, deixando sua cópia na área de leitura, para sofrer algum tratamento que se deseje, ou constatar erro da transação; como por exemplo inclusão para registro já existente, etc.

. Inserção

O processo de inserção pode ser resumido nos seguintes passos:

1 - chama rotina CONSULTA para a chave-primária fornecida;

2 - se a lista está vazia faz:

pede registro do espaço disponível,
insere o novo registro, e
coloca endereço na entrada apropriada da Tabela Índice,
retorna;

3 - se lista não vazia e já existe registro com tal chave-primária, dá mensagem de erro; retorna;

4 - se lista não vazia;

consulta último registro da cadeia;

4.1 - se não for registro de expansão de lista, faz expansão

solicita registro do espaço disponível;

copia último registro da lista para novo local;

formata expansão sobre último registro da lista, quando chave-primária e endereço do registro salvo no passo anterior;

solicita registro do espaço disponível;

insere novo registro;

guarda chave-primária e endereço no registro expansão e grava;

retorna;

4.2 - se for registro de expansão não cheio;
faz

solicita registro do espaço disponível;

insere novo registro;

guarda chave-primária e endereço no registro expansão e regrava;

retorna;

4.3 - se for registro de expansão cheio e existir outro registro de expansão, lê o próximo e volta a (4.2).

4.4 - se for registro de expansão cheio e não existir outro registro de expansão para a lista faz

solicita registro do espaço disponível;

guarda endereço de expansão (regrava);

formata próximo expansão (lê próximo); e volta a 4.2.

fim.

A cada registro incluído no Arquivo Mestre, torna-se necessário criar referências ao mesmo no Arquivo de Listas Invertidas que digam respeito a cada campo definido como chave-secundária na Tabela de Definição do Registro Mestre.

Assim, no procedimento designado nos passos (2), (4.1) e (4.2) como "insere", estando preparada a área de gravação (REGMEST) com o registro a ser inserido, deve-se realizar sucessivas chamadas a rotina de inclusão nas listas invertidas, de acordo com o que descreveu-se no capítulo (IV.4). Nesse caso, as inserções nas listas são realizadas pela rotina ECISLI para a opção INCLUIR (atribuindo-se ao parâmetro ALTER ← 'I') e atribuindo ao parâmetro CODATB valores de 2 a n, sendo n o número de ordem do último campo definido no Dicionário de Dados.

Caso a inserção do registro seja realizada em uma modalidade interativa, em que o usuário fornece os valores de campo por campo, torna-se mais prático que cada chamada à rotina ECISLI seja após o movimento de cada valor para sua respectiva posição na área (REGMEST).

. Alteração

A rotina de alteração do conteúdo de um campo qualquer de um registro do Arquivo Mestre identificado por sua chave-primá

ria, pode ser resumida nos seguintes passos:

1 - chama rotina CONSULTA para a chave-primária dada;

2 - se não foi localizado o registro, dá mensagem de erro e retorna;

3 - se foi localizado o registro

altera o campo desejado na área (REGMEST) e regrava o registro;

fim.

A alteração de um campo do Registro de Dados, caso esse campo tenha sido designado como chave-secundária, implica na atualização de duas listas invertidas, quais sejam:

. exclusão da respectiva chave-primária desse Registro de Dados da lista definida pelo par chave-secundária /antigo valor da chave-secundária; e

. inclusão da respectiva chave-primária desse Registro de Dados na lista definida pelo par chave-secundária/novo valor da chave-secundária.

Portanto, isso requer que sejam realizadas duas chamadas a rotina ECISLI, atribuindo-se a CODATB o número de ordem do campo:

. uma com a opção EXCLUIR (ALTER ← 'E'), que antecede ao movimento do novo campo para REGMEST; e

. uma com a opção INCLUIR (ALTER ← 'I'), que segue ao movimento do novo valor para REGMEST.

. Exclusão

A exclusão de um registro do Arquivo Mestre é efetivada logicamente, assinalando-se com o carater (X). O espaço que é liberado só pode ser reaproveitado por um registro de expansão ou quando ocorrer uma realocação. Para manter as listas invertidas atualizadas, deve-se realizar sucessivas chamadas à rotina ECISLI, para a opção EXCLUIR, (ALTER ← 'E'), atribuindo ao parâmetro CODATB valores de 2 a n, sendo n o número de campos do Registro de Dados definido no Arquivo Dicionário de Dados.

V.3 Dimensionamento da Tabela Índice do Arquivo Mestre

Existem diversos fatores que podem influenciar no dimensionamento da Tabela Índice, visto que se deseja recuperar informações em um tempo de resposta dentro de certos limites, dando um bom aproveitamento aos recursos de uma pequena máquina.

Estudo minuncioso sobre esse tema foi feito considerando as listas não extendidas, o qual encontra-se no Anexo (I) "Dimensionamento de Tabela de Índices", tendo-se chegado às conclusões abaixo, partindo das seguintes definições:

M - quantidade de registros do arquivo de dados;

N - quantidade de entradas da Tabela Índice;

B - número de registros do arquivo de dados por setor;

$L = \frac{M}{N}$ comprimento médio das listas de sinônimos.

Concluiu-se que para o intervalo $1 \leq L \leq B + 1$ o valor esperado E(C) do número de acessos ao Arquivo Mestre é dado por:

$$E(C) = 1 + \frac{L - 1}{2B} \qquad \text{(expressão 1.4 do Anexo I)}$$

Quando $L = B + 1$ tem-se

$$E(C) = 1,5;$$

Enquanto para o intervalo $B \leq L \leq 2B + 1$ quando $L = 2B + 1$ acarreta $E(C) = 2$.

Resumindo, do estudo encerrado no Anexo I concluiu-se que, para um dado M, caso seja possível manter a Tabela Índice na memória para algum valor de sua dimensão $N = \frac{M}{L}$, de modo

que $1 \leq L \leq 2B + 1$, o número esperado de acesso a disco permanece no intervalo $1 \leq E(C) \leq 2$, ou seja, o valor máximo esperado para E(C) no intervalo é igual ao melhor caso (L = 1) para a alternativa de se manter a tabela em disco com dimensão $N \geq M$.

Verificou-se ainda que, para processadores lentos, quando a pesquisa linear sobre as listas de sinônimos pode consumir tempo não desprezível em relação ao tempo total de recuperação (para L grande) tem-se como limite para manter a tabela na memória um valor de N tal que:

$$L = \frac{2B + 1}{xB + 1} \quad \text{onde}$$

"x" a relação entre o tempo t consumido para consultar cada registro da lista de sinônimos, na memória, e o tempo T_s médio de leitura e transmissão à memória de cada setor do disco.

O sistema implementado oferece, como flexibilidade, a possibilidade de ajustar a dimensão da Tabela Índice às condições que se deseja trabalhar. Para isso, é suficiente fornecer a dimensão desejada como parâmetro do programa (ECCJØ1) que inicializa a Tabela e executa a rotina de carga do Arquivo Mestre.

Para arquivos pouco voláteis as conclusões a que se chegou são pouco modificadas, pois para listas pouco extendidas executa-se apenas uma leitura a mais.

VI. DETALHES E COMENTÁRIOS SOBRE A IMPLEMENTAÇÃO NO POTI

VI. DETALHES E COMENTÁRIOS SOBRE A IMPLEMENTAÇÃO NO POTI

Para implementação do modelo aqui projetado foi utilizado o equipamento POTI, desenvolvido no Laboratório Digital do NCE, com as características detalhadas no capítulo (II).

Foi simulada uma aplicação com o objetivo de se constatar a exatidão das rotinas e fornecer alguns indicadores, com os quais se possa fazer previsão de desempenho para outras aplicações.

Vale salientar que a linguagem PLTI mostrou-se apropriada para o desenvolvimento de software dessa natureza, bem como a facilidade de over-lay do sistema operacional SOCO foi decisiva, para viabilizar a implementação desse modelo, caso contrário não se dispunha de memória suficiente.

A aplicação simulada foi definida com:

- arquivo de dados com registros de 32 bytes;
- 360 registros identificados com chaves-primárias individuais;
- registro de lista invertida (Li) com 64 bytes; e
- 5 campos de chave-secundária, para os quais foram construídas listas invertidas.

Quanto aos dados utilizados, não se teve a preocupação de gerar uma distribuição uniforme de valores, pois desejava-se observar o comportamento em situações desfavoráveis, mesmo porque na prática é muito comum que as chaves de identificação e valores de atributos possuam algum vício, ou tendência a aglomerações.

VI.1 Carga dos Arquivos

O processo de carga dos arquivos de dados e listas invertidas, foi realizado em vários passos, compreendendo aqui cada passo um programa para esse fim.

. Geração dos Endereços de Entradas da Tabela Índice

Tomando por módulo o comprimento da Tabela Índice, informado em seu registro header, aplica-se a função endereçamento por Divisão sobre as chaves-primárias de cada registro, gravando-se em outro arquivo cujos registros são acrescidos desse endereço calculado. Nesse instante é criado o header do Arquivo Mestre tendo inicializados aqueles campos que dependem da Tabela Índice, sobretudo necessários a verificação da compatibilidade dos mesmos em tempo do processamento interativo. Esse passo tem ainda a função de eliminar os registros assinalados como excluídos logicamente (FLAG = 'X').

. Inicialização do Arquivo Mestre (DECJ2Ø)

Esse passo se resume em classificar o arquivo saída do passo anterior pelo campo de controle inicializado com os endereços calculados, de forma que todos registros de chaves-primárias sinônimas ocupem posições contíguas e em ordem ascendente. Para permitir que o registro header seja localizado como primeiro registro do arquivo, a classificação obedeceu a seguinte prioridade de campos e ordem ascendente de valores:

. campo controle ENDINDADO;

. campo FLAG; e

. campo chave-primária.

. Inicialização de Tabela Índice

Percorrendo-se o Arquivo Mestre inicializado, cada lista de sinônimo constatada tem seu endereço de início carregado na entrada (ponteiro) apropriada da Tabela Índice. Nesse instante os headeres são atualizados com informações a respeito do número de registros do Arquivo Mestre, registros ativos e disponíveis.

. Inicialização das Listas Invertidas (DECJLI)

Também é realizada percorrendo cada registro do Arquivo Mestre, além de consultar todos os campos definidos no Dicionário de Dados executando, para aqueles designados como chave-secundária, o processo de inclusão pelo uso da opção INCLUIR da rotina ECISLI vista no capítulo (IV).

Esse último passo mostrou-se lento, como era previsto, pois para os dados da aplicação simulada, seu tempo de duração foi de 35 minutos.

Acredita-se que uma ligeira modificação na rotina ECISLI, apenas para a execução da fase de inicialização das listas invertidas, possa salvar uma razoável parcela desse tempo. Essa modificação constaria da eliminação do teste de duplicidade de chave-primária, necessária nas inclusões pelo processo interativo para evitar danos causados por falha do usuário. Entretanto, sendo esses passos aqui descritos realizados automaticamente, tais falhas não podem ocorrer.

Como atenuante à lentidão da carga das listas, lembramos como foi dito no capítulo (III), que a recarga dessas listas invertidas só se torna necessária caso haja alguma alteração no conjunto de chaves-secundárias, como inclusão de uma nova ou

exclusão* de outra, ou ainda pela alteração do módulo do índice de uma chave-secundária.

Essa propriedade foi adquirida quando optou-se pelo endereçamento simbólico (chave-primária em lugar de endereço físico) das listas invertidas para os registros de dados.

Mesmo quando indispensável for a execução de recarga das listas invertidas, tal inconveniente pode ser ameno considerando-se que a mesma pode ser executada em ocasião anterior e não conflitante com o exercício da atividade fim a que se destina a aplicação como instrumento.

---

* - A exclusão não necessariamente implica em recarga de lista, pois pode ser feita apenas alterando seu código no Dicionário de Dados, mas deve liberar bastante espaço do arquivo.

VI.2 Processo Interativo de Atualização

Desenvolveu-se um programa (ECCJ33) que permitiu consultar ou atualizar o Arquivo Mestre de modo interativo.

Foram realizados testes para diversas condições, constatando-se que os tempos de execução para consulta a um registro, dada sua chave-primária, mostraram-se bons considerando as alternativas de uso do equipamento POTI: mesmo em condições desfavoráveis como a consulta ao último registro de uma lista com 15 registros, sendo dois de expansão, conseguiu-se uma resposta em aproximadamente 3 segundos (tempo que pode ser considerado bom em um processo interativo).

Os processos de inclusão de novos registros e alteração de campos, são recomendáveis (e por isso foram adotados) que sejam executados campo a campo, intercalando tempos consumidos pelo computador e pelo usuário, no mesmo instante que podem ser realizadas críticas sobre os dados e por outro lado podem ser atualizadas as listas invertidas, conforme descreveu - se no capítulo (V) sobre atualizações do Arquivo Mestre.

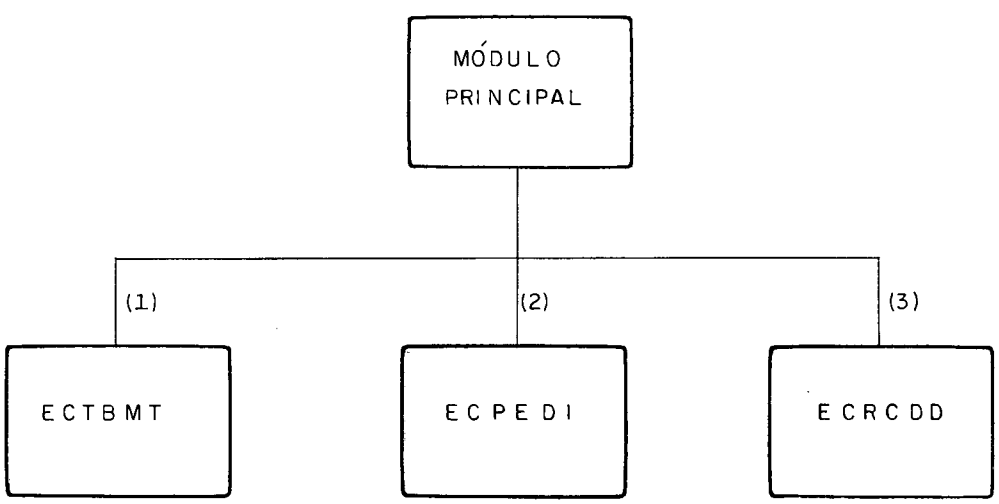
Lembrou-se ser obrigatório, ao encerrar o processamento desta etapa, o programa executor gravar os headeres atualizados quanto a atualização de espaço disponível.

VI.3 Processo Interativo para Formulação de Pedidos

O processo interativo para recuperação de informação por múltiplas-chaves é representado por um programa (ECCJ60) constituído de rotinas básicas definidas anteriormente. Devido à modularidade conseguida nesse projeto, a construção desse programa tornou-se bastante simples, fato da maior importância, quando lembrado que ele encerra o objetivo principal deste trabalho.

No início de seu processamento (SETUP) é feita a verificação da compatibilidade entre o Arquivo Diretório (DECJII) com o arquivo (DECJLI), bem como do Arquivo Tabela Índice com o Arquivo Mestre.

Caso não haja incompatibilidade, o processo continua carregando a Tabela Dicionário de Dados no vetor VETRTB (rotina ECTBMT). Essa rotina foi colocada no mesmo nível de over-lay que "ECPEDI" e a rotina de recuperação de registros do Arquivo Mestre (ECRCDD). A estrutura de over-lay projeta (e recomenda) é a seguinte:



Naturalmente, que no módulo principal se encontram as áreas de inter-face entre os módulos do segundo nível.

A sequência de passos é a seguinte:

1 - carrega o módulo ECTBMT;

2 - executa o módulo ECTBMT;

3 - carrega o módulo ECPEDI;

4 - executa o módulo ECPEDI;

4.1 - solicita Pedido;

4.2 - analisa Pedido;

4.3 - executa resolução do Pedido deixando chaves-primárias na área RESULTANTE;

5 - se deseja refinar Pedido volta para (4);

6 - carrega o módulo ECRCDD;

7 - executa o módulo ECRCDD para cada chave-primária em RESULTANTE

7.1 - expõe no VIDEO registro recuperado

8 - retorna a (3);

fim.

O encerramento do processo é comandado pelo usuário em resposta a pergunta formulada no módulo ECPEDI.

Esse processo foi submetido a exaustivos testes objetivando averiguar seu corretismo, ao mesmo tempo que se avaliava seu desempenho.

Testou-se um Pedido envolvendo 5 (cinco) operadores, consequentemente 5 (cinco) reduções, bem como a quantidade superior a 100 (cem) chaves-primárias, apanhadas das listas envoldidas, mas resultando apenas 17 (dezessete) dessas chaves-primárias. O tempo consumido foi aproximadamente 16 (dezesseis) segundos (talvez o pior caso para o modelo simulado), para o processo de redução do Pedido.

É de se observar que o tempo consumido pela resolução de um Pedido não depende do tamanho do Arquivo Mestre, necessariamente, mas do número de operadores da expressão lógica e do comprimento das listas envolvidas.

Exemplo de saída do programa ECCJ6Ø

```
TABELA DEFINICAO DO REG MESTRE DE DADOS -ARQ  ECCJ20
QUANT. BYTES UTEIS 032 COM 010 CAMPOS./DATA REF. 800516
*                      *          *      *
REFER DESCRICAO.....  TIPO LOCAL COMP V. BASE MODULO

 C01  CHAVE PRIMARIA    C     001   05   00000   000
 A02  ATRIBUTO UM       1     006   03   00000   128
 A03  ATRIBUTO DOIS     0     009   05   00000   000
 A04  ATRIBUTO TRES     1     014   04   00000   383
 A05  ATRIBUTO TRES     1     018   04   00000   383
 A06  ATRIBUTO CINCO    1     022   04   00000   128
 A07  ATRIBUTO SEIS     0     026   02   98005   000
 A08  ATRIBUTO SETE     1     028   03   00000   127
 A09  CAMPO CONTROLE    C     031   02   00000   000
TECLE SEU PEDIDO OU DESCONTINUI(D)
A02 = 770 & A08 = 771 ?
    2 ENTIDADES RESULTANTES DO SEU PEDIDO = SE DESEJA REFINAR TECLE (S)
TECLE SEU PEDIDO OU DESCONTINUI(D)
A02 = 110 & (A08 = 129 & A04 = 5050) ?
    1 ENTIDADES RESULTANTES DO SEU PEDIDO = SE DESEJA REFINAR TECLE (S)
TECLE SEU PEDIDO OU DESCONTINUI(D)
```

VI.4 Sugestões para o Aperfeiçoamento de Desempenho

O desempenho do modelo de sistema de recuperação de informações aqui tratado, pode ser afetado pelas chaves-secundárias escolhidas, pela dimensão dos registros com que forem criados os arquivos, especialmente listas invertidas (DECJLI), índice de Chave-secundária e a Tabela índice (DECJØI), e ainda pela própria formulação do Pedido.

Para aperfeiçoar ou manter o nível de desempenho do sistema, é recomendável periodicamente avaliar a distribuição dos comprimentos de listas, conforme descrito na seção (IV.5) (isso deve ser feito por um programa que tenha essa especial função). Essa avaliação pode levar a que seja alterado o comprimento do registro LI, ou alterar o conjunto de chaves-secundárias.

Por outro lado, caso seja observada a ocorrência de muitas colisões no Arquivo Diretório (DECJII) ou no Arquivo LI(DECJLI), deve ser ampliado o tamanho dos índices dessas chaves-secundárias.

Há casos de chaves-secundárias que possuem uma pequena quantidade de diferentes valores, como por exemplo o atributo sexo, resultando em pequeno grupo de longas listas. Caso sejam de relevante interesse para frequentes formulações de Pedidos em conjunção com uma outra chave-secundária, pode-se declarar no Dicionário de Dados essas duas chaves como sendo uma única - colocando os campos em posições contíguas sobre o Registro Mestre e declarando uma chave-secundária que abranja esses dois campos. Assim, resultam listas menos longas, elimina-se uma resolução nos Pedidos que envolvem as duas chaves-secundárias e consequentemente salva-se tempo na Resolução do Pedido.

Quanto a formulação de Pedido deve-se observar alguns detalhes:

- O Pedido deve ser formulada pela expressão reduzida, evitando que se façam reduções em número desnecessário.

As expressões abaixo são equivalentes, sendo as condições atômicas representadas por A_1, A_2, B_1 e B_2:

a) $(A_1 \ \& \ B_1)$ OR$(A_1 \ \& \ B_2)$?

b) $A_1 \ \& \ (B_1$ OR $B_2)$?

Para a expressão (a) seriam realizadas 3 reduções enquanto para (b) seriam 2, apesar do número resultante de registros a recuperar ser o mesmo.

Tendo-se conhecimento que uma condição atômica possui uma lista invertida muito grande, de tal forma que todas suas chaves-primárias não possam ser contidas na área de resolução de Pedido, essa chave-secundária não deve ser o primeiro operando da expressão ou subexpressão.

Na expressão (b) acima, pode ocorrer que o resultado intermediário da subexpressão $(B_1$ OR $B_2)$ exceda a capacidade da área auxiliar de resolução (RESULTANTE no exemplo implementado, pode conter até 102 chaves-primárias). Nesse caso seria recomendável a expressão (a). E ainda, se em (a) a concatenação das duas subexpressões exceder a capacidade da área auxiliar, mas cada uma isolada não, sugere-se formular dois Pedidos distintos, para cada subexpressão.

Deixou-se de desenvolver as rotinas para seleção por operadores relacionais (maior do que) e formatação, após a recuperação no Arquivo Mestre, dos registros selecionados através de um Pedido na forma em que foi implementado. Entretanto, o

caráter experimental deste trabalho não ficou prejudicado, visto que necessitava-se conhecer o comportamento do modelo simplificado, para então avaliar os benefícios do investimento em sofisticação.

Por outro lado, vale acrescentar que o Módulo de Recuperação de Registros não foi desenvolvido com facilidades de uso geral mas, pode ser aproveitado com algumas adaptações como da função de endereçamento as características da chave - primária da aplicação, bem como seu armazenamento no registro de expansão.

Uma outra alternativa de Tabela de Espalhamento pode ser tentada, resolvendo-se as colisões na própria Tabela, pela técnica de endereçamento aberto eliminando a necessidade de registro de extensão. Porém essa alternativa só parece interessante quando não for conveniente, ou possível, manter toda Tabela na memória.

Como já mencionado, o módulo de recuperação de Registro, não necessariamente deve ser semelhante ao implementado.

VII CONCLUSÕES

VII. CONCLUSÕES

Julgou-se que o modelo implementado para recuperação de informação por múltiplas-chaves, mostrou-se amplamente eficaz em relação aos objetivos preconizados, dentro das restrições que lhe foram impostas, permitindo manter níveis de desempenho plenamente satisfatórios.

A sua implementação é simples, exigindo um esforço mínimo de analista e programador.

Requer cuidados especiais no dimencionamento dos arquivos pertinentes ao Módulo Interpretador de Pedidos e Seleção de Chaves-Primárias, devendo se observar as recomendações do Capítulo IV. Cuidado também deve ser devotado à escolha das chaves-secundárias, cujos aspectos de ordem comum tratados no Capítulo II, podem auxiliar a tomada de decisão, levando-se em conta restrições do próprio modelo.

Por fim, lembra-se que ao pretender dotar esse modelo com características de uso geral e dada a diversificação de peculiaridades das aplicações, o atingimento e manutenção de níveis de desempenho dependerá do analista acompanhar a evolução do comportamento da aplicação em uso, bem como do conhecimento das facilidades para ajustes disponíveis no modelo de recuperação de informações que encerra este trabalho.

A N E X O S

ANEXO I - DIMENSIONAMENTO DA TABELA DE ÍNDICES DO ARQUIVO MESTRE

ANEXO I

DIMENSIONAMENTO DA TABELA DE ÍNDICES DO ARQUIVO MESTRE

Existem diversos fatores que podem influenciar no dimensionamento da Tabela de Índices, visto que se deseja recuperar informações em um tempo de resposta dentro de certos limites, dando um bom aproveitamento aos recursos de uma pequena máquina.

Ao analisar as alternativas, deve-se considerar os seguintes fatores:

M - quantidade de registros do arquivo de dados (quantidade de entidades sobre as quais se devem manter informações em um arquivo mestre);

N - dimensão da Tabela de Índices (quantidade de apontadores de listas de sinônimos da Tabela de Índices);

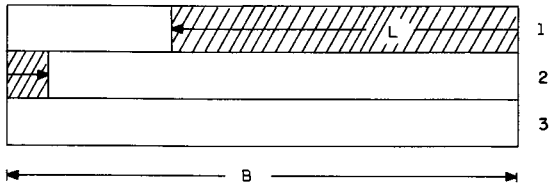
$L = \frac{M}{N}$ (I.1) comprimento médio das listas de sinônimos;

B - fator de bloco do arquivo mestre (no TI é limitado ao número de registros por setor);

Pi - podendo uma lista de sinônimos de comprimento L estar distribuída ao longo de consecutivos blocos do arquivo mestre, "Pi" representa a probabilidade de se localizar no bloco de ordem i, que contém, em relação à L, um elemento pertencente a L. Supõe-se que tanto a escolha da entidade a qual corresponde o registro a localizar, como a posição do primeiro registro da lista dentro do primeiro dos blocos que a contém, são ocorrências aleatórias, cujos resultados são equiprováveis.

PRIMEIRA ALTERNATIVA - $L \leq B + 1$

Para $L \leq B + 1$, a lista L estará distribuída no máximo em dois blocos, dependendo da posição em que inicia a lista, em relação ao início do primeiro bloco que contém L. A figura abaixo mostra um exemplo deste caso.



Para este caso particular pode-se demonstrar que as probabilidades do registro desejado estar no primeiro bloco (P_1) ou no segundo bloco (P_2) são expressas por:

$$P_1 = 1 \frac{L - 1}{2B} \qquad (1.2)$$

$$P_2 = \frac{L - 1}{2B} \qquad (1.3)$$

Demonstração:

$$P_1 = P_1 + P_2 + P_3 + \ldots + P_i + \ldots + P_L$$

p_i - é a probabilidade do registro procurado ser o de ordem "i" da lista de comprimento L, e ele estar armazenado no primeiro bloco (bloco 1) que contém L.

Defina-se dois eventos:

X = [o registro procurado ser i-ésimo da lista]

Y = [o i-ésimo da lista estar no 1º bloco]

Considerando-se os eventos X e Y acima pode-se dizer que a probabilidade pi definida acima é igual a probabilidade de ocorrerem os dois eventos X e Y simultaneamente. Isto é:

$$pi = Pr(X \cap Y) = Pr(X) \times Pr(Y|X)$$

Como se pode ver, os eventos X e Y são independentes. Assim:

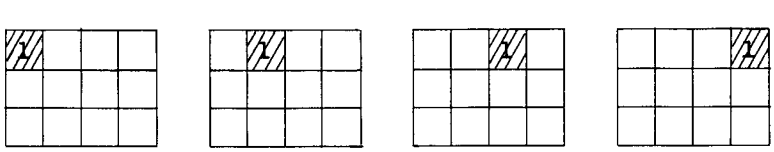
$$pi = Pr(X \cap Y) = Pr(X) \times Pr(Y)$$

Resta agora definir Pr(X) e Pr(Y). Como se pode deduzir a probabilidade de que o registro procurado seja o i-ésimo registro de lista de L elementos é dada por $Pr(X) = \frac{1}{L}$, visto que todos os L elementos da lista são equiprováveis.

Com relação à probabilidade do evento Y = Pr (Y), vamos deduzir sua expressão através dos exemplos práticos abaixo:

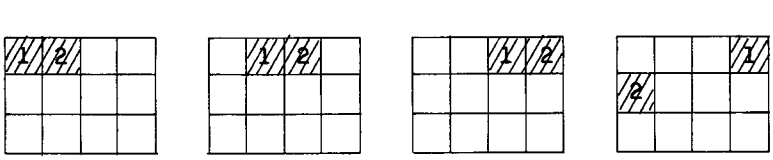
1) CASO I - L < B:

a) L = 1:



$$P_r(1) = \frac{4}{4} = 1$$

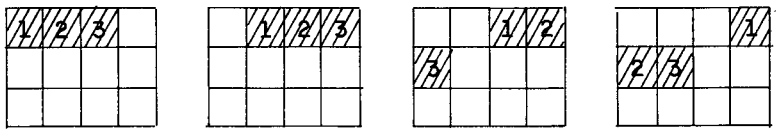
b) L = 2:



$$P_r(1) = \frac{4}{4} = 1$$

$$P_r(2) = \frac{3}{4} = \frac{3}{4}$$

c) L = 3:



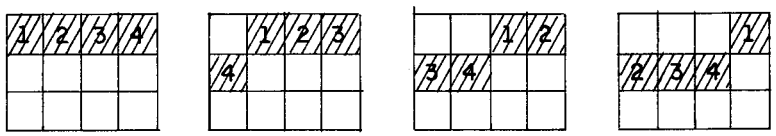
$$P_r(1) = \frac{4}{4} = 1$$

$$P_r(2) = \frac{3}{4} = \frac{3}{4}$$

$$P_r(3) = \frac{2}{4} = \frac{1}{2}$$

II) CASO II - L = B:

d) L = 4:



$$P_r(1) = \frac{4}{4} = 1$$

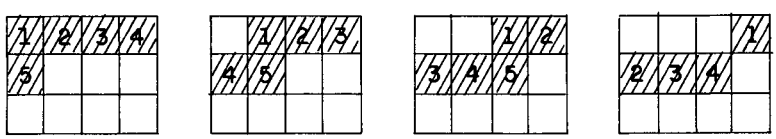
$$P_r(2) = \frac{3}{4} = \frac{3}{4}$$

$$P_r(3) = \frac{2}{4} = \frac{1}{2}$$

$$P_r(4) = \frac{1}{4} = \frac{1}{4}$$

III) CASO III - L > B → L = B + 1:

e) L = 5:

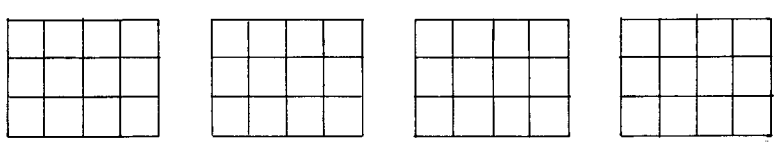


$$P_r(1) = \frac{4}{4} = 1$$

$$P_r(2) = \frac{3}{4} = \frac{3}{4}$$

$$P_r(3) = \frac{2}{4} = \frac{1}{2}$$

$$P_r(4) = \frac{1}{4} = \frac{1}{4}$$



$$P_r(5) = \frac{0}{4} = 0$$

Observando-se os exemplos práticos pode-se deduzir que:

$$Pr(Y = i) = \frac{B - i + 1}{B}$$

Assim, pi será dada por:

$$pi = \frac{1}{L} \times \frac{B - i + 1}{B}$$

Deste modo:

$$P_1 = \frac{1}{L} \times \frac{B}{B} + \frac{1}{L} \times \frac{B-1}{B} + \frac{1}{L} \times \frac{B-2}{B} + \ldots + \frac{1}{L} \times \frac{B-(i-1)}{B} +$$

$$+ \ldots + \frac{1}{L} \times \frac{B-(L-1)}{B} \quad \text{ou}$$

$$P_1 = \frac{1}{LB} \{B + (B - 1) + (B - 2) + \ldots + [B - (i - 1)] + \ldots +$$

$$+ [B - (L - 1)]\} \therefore$$

$$P_1 = \frac{1}{LB} \times \left(\frac{(B + B - L + 1)}{2}\right) = \frac{2B - L + 1}{2B} \quad \therefore$$

$$P_1 = 1 - \frac{L - 1}{2B} \qquad \text{c.q.d}$$

Considerando que $L \leq B + 1$, pode-se concluir que a probabilidade do registro estar sobre o segundo bloco é:

$$P_2 = 1 - P_1 = \cancel{1} - \cancel{1} + \frac{L - 1}{2B} \quad \text{ou}$$

$$P_2 = \frac{L - 1}{2B} \qquad \text{c.q.d.}$$

VALOR ESPERADO DO NÚMERO DE ACESSOS AO ARQUIVO MESTRE

Considerando-se que uma lista de comprimento L esteja distribuída sobre n blocos consecutivos e que para acessar ao j-ésimo bloco são necessárias "j" leituras físicas ao disco hospedeiro, e sendo P_j a probabilidade de que um registro escolhido aleatoriamente esteja no mesmo j-ésimo bloco, o valor esperado do número de acessos ao "arquivo mestre", E(C), é dado por (KNUTH[1], pg396):

$$E(C) = 1 \times P_1 + 2 \times P_2 + 3 \times P_3 + \ldots + j \times P_j + \ldots + n \times P_n$$

para essa alternativa em que $L \leq B + 1$.

Tem-se

$$E(C) = 1 \times P_1 + 2 P_2 = P_1 + 2 (1 - P_1) = 2 - P_1$$

Substituindo P_1 pelo seu valor apresentado em (1.2) deste anexo, tem-se

$$E(C) = 2 - (1 - \frac{L - 1}{2B}) \rightarrow E(C) = 1 + \frac{L - 1}{2B} \qquad (1.4)$$

Como se pode observar, para um mesmo tamanho de bloco B, o valor esperado do número de acessos diminui, tendendo para 1 ($E(C) \rightarrow 1$), a medida que o comprimento da lista, L, tende para 1 ($L \rightarrow 1$), ou seja, quando cada um dos M registros do Arquivo Mestre possui um distinto apontador na "Tabela Índice".

A seguir, avalia-se o comportamento do número médio (valor esperado) de acessos ao Arquivo Mestre para os valores extremos

de L: $\underline{L = 1}$ e $\underline{L = B + 1}$

a) $\underline{L = 1}$:

$$E(C) = 1 + \frac{1 - 1}{2B} = 1 + \frac{0}{2B} \rightarrow E(C) = 1$$

Neste caso (L = 1), torna-se necessário o dimensionamento de uma "Tabela de Índices" com uma dimensão de tal ordem que poderia desaconselhar a sua manutenção na memória principal, provocando um mal aproveitamento do espaço em disco.

Deste modo, considerando-se que a função "Hashing" não produzisse sinônimos na Tabela de Índices, e que os endereços das entradas na Tabela de Índices gerados pela mesma função obedecessem uma distribuição uniforme, a dimensão daquela tabela seria:

$$N = \frac{M}{L} \rightarrow N = M$$

Por outro lado, caso a Tabela de Índices não possa ser mantida na memória principal, o número médio de acessos cresceria para:

$$E(C)_1 = 2$$

b) $\underline{L = B + 1}$

$$E(C) = 1 + \frac{B + 1 + 1}{2B} = 1 + \frac{B}{2B} \rightarrow E(C) = 1,5$$

Como se observa, o número médio de acessos aqui é maior do que o caso anterior, mantendo-se, em ambos os casos, a Tabela de Índices na memória principal. No entanto, este tamanho (L = B + 1) da Lista de Sinônimos permite, muito mais facilmen

te, que seja mantida a Tabela de Índices na memória principal, dado a sua menor dimensão requerida.

Neste último caso, o tamanho daquela Tabela seria:

$$N = \frac{M}{L} \quad \text{ou} \quad N = \frac{M}{B+1}$$

Analisando-se os resultados apresentados acima pode-se concluir que a escolha do comprimento L da Lista de Sinônimos, e consequente dimensão da "Tabela de Índices", requer uma análise de "Trade-off" entre o espaço de memória principal requerido (e disponível) para manutenção daquela tabela e o número médio de acessos à Tabela.

Do exposto pode-se concluir o seguinte:

(1) Caso a dimensão da Tabela de Índices seja tal que permita o seu armazenamento na memória principal, quando $L = 1$, a opção mais vantajosa para o comprimento da Lista de Sinônimos seria adotar $L = 1$, e, consequentemente, o comprimento da Tabela de Índices seria $N = M$;

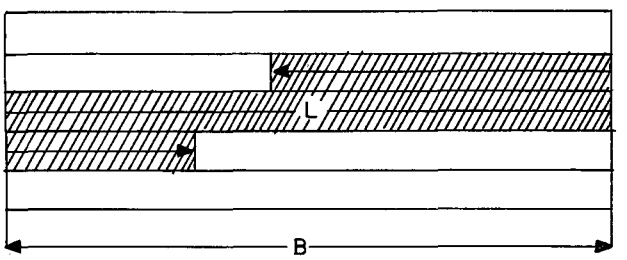
(2) Caso a dimensão da Tabela de Índices seja tal que não permita seu armazenamento na memória principal, quando $L = 1$, porém seja viável quando $L = B + 1$, deve-se optar pela alternativa de utilizar o comprimento da Lista de Sinônimos $L = B + 1$, com a Tabela de Índices na memória. Neste caso, o comprimento recomendado para a Tabela de Índices seria $N = \frac{M}{B+1}$.

(3) Caso sua dimensão torne inviável seu armazenamento na memória principal, para qualquer das duas alternativas limites de L, a Tabela de Índices será colocada em disco, e a alternativa escolhida para comprimento da Lista de Sinônimos deverá ser $L = 1$, com consequente comprimento daquela tabela igual a $N = M$.

O sistema implementado oferece, como flexibilidade, a possibilidade de ajustar as condições em que se deseja trabalhar, sendo necessário apenas fornecer ao programa inicializador da Tabela de Índices do Arquivo Mestre a dimensão da mesma, através de parâmetro (por teclado), e executar sua rotina de carga.

SEGUNDA ALTERNATIVA - $B \leq L \leq 2B + 1$

Para $B \leq L \leq 2B + 1$ a lista de comprimento L estará distribuída no máximo em três blocos, dependendo da posição em que L inicia em relação ao início do primeiro bloco que a contém.



O valor esperado do número de acessos ao "Arquivo Mestre" será dado então, por (KNUTH[1], pag. 396):

$$E(C) = P_1 + 2 \times P_2 + 3 \times P_3$$

Demonstra-se, para este caso particular, que as probabilidades do registro desejado estar no primeiro bloco (P_1), no segundo bloco (P_2), ou no terceiro bloco (P_3), são dados por:

$$P_1 = \frac{B + 1}{2L} \qquad (1.5) \qquad P_2 = \frac{1}{2LB} \left| B(B - 1) + (3B - L + 1)(L - B) \right| \qquad (1.6)$$

$$\text{e} \quad P_3 = \frac{1}{2LB} \times (L - (B + 1))\ (L - B) \qquad (1.7)$$

Demonstrações:

(1) Cálculo de P_1:

Chamemos de "pi" a probabilidade do registro procurado ser o i-ésimo elemento da Lista de Sinônimos de comprimento "L", e ele encontrar-se armazenado, no primeiro bloco que contém a lista.

Como já foi demonstrado pi é dada por:

$$pi = \frac{1}{L} \times \frac{B - i + 1}{B}$$

Seja P_1, a probabilidade de se encontrar no bloco onde se inicia a Lista L (bloco 1), um elemento da Lista de Sinônimos de tamanho L. Essa probabilidade será dada por:

$$P_1 = \sum_{i=1}^{L} \frac{1}{L} \cdot \frac{B - i + 1}{B} = \sum_{i=1}^{B} \frac{1}{L} \cdot \frac{B - i + 1}{B} + \sum_{i=B+1}^{L} \frac{1}{L} \cdot \frac{B - i + 1}{B}$$

O segundo somatório tem valor nulo, visto que, para todo i > B torna-se impossível o registro procurado encontrar-se no primeiro bloco. Assim:

$$P_1 = \sum_{i=1}^{B} \frac{1}{L} \cdot \frac{B - i + 1}{B} = \frac{1}{L.B} \Sigma\, B - i + 1 = \frac{1}{LB} \left[B + (B-1) + \ldots + 2 + 1 \right]$$

$$P_1 = \frac{(B + 1) . \cancel{B}}{2L\cancel{B}} \quad \text{ou} \quad P_1 = \frac{B + 1}{2L} \qquad (1.5) \qquad \text{c.q.d.}$$

(2) Cálculo de P_2:

Seja P_2, a probabilidade de se encontrar no segundo dos blocos onde se distribui a lista L, um dado elemento pertencente à Lista de Sinônimos. Seja agora "pi" a probabilidade do registro procurado ser o de ordem "i" da lista de comprimento L, e dele estar armazenado no segundo bloco (bloco 2) em que se encontra distribuída a mesma lista.

Defina-se os dois eventos:

X = [o registro procurado ser o i-ésimo da lista]

Y = [o i-ésimo da lista estar no 2º bloco]

Considerando-se os eventos X e Y acima, pode-se dizer que a probabilidade "pi" definida anteriormente é igual a probabilidade de ocorrerem os dois eventos X e Y simultaneamente. Isto é:

$$pi = Pr(X \cap Y) = Pr(X) \times Pr(Y/X)$$

sendo os eventos X e Y independentes, pois estar o i-ésimo da lista no segundo bloco independe dele ser o registro procurado, pode-se escrever

$$pi = Pr(X \cap Y) = Pr(X) \times Pr(Y)$$

Resta definir Pr(X) e Pr(Y) como já foi dito, $Pr(X) = \frac{1}{L}$, visto que todos dos "L" elementos da lista são equiprováveis.

Seguindo o mesmo processo indutivo utilizado para a alternativa em que L < B, discutida anteriormente, chegou-se a conclusão, tomando-se os dois intervalos abaixo, que:

(a) Para $1 \leq i \leq B$ tem-se $Pr(Y = i) = \frac{i - 1}{B}$ e

(b) Para $B + 1 \leq i \leq L$ tem-se $Pr(Y = i) = \frac{2B - (i - 1)}{B}$

Sendo $(i - 1)$ e $[2B - (i - 1)]$, as quantidades de ocorrências favoráveis do elemento de ordem i pertencente àqueles respectivos intervalos, encontrar-se no segundo bloco (Bloco 2).

E por conseguinte pode-se escrever que:

$$P_2 = \sum_{i=1}^{L} pi = \sum_{i=1}^{B} pi + \sum_{i=B+1}^{L} pi$$

Considerando-se o intervalo definido em (a) e a expressão anterior de "pi" em função Pr(X) e Pr(Y), vem:

$$\sum_{i=1}^{B} pi = \sum_{i=1}^{B} Pr\,(X = i) \times Pr\,(Y = 1) = \sum_{i=1}^{B} \frac{1}{L} \times \frac{i - 1}{B}$$

$$= \frac{1}{LB} \left[0 + 1 + 2 + 3 + \ldots + (B - 2) + (B - 1) \right] = \frac{1}{LB} \times \left[\frac{(B - 1)B}{2} \right]$$

ou seja:

$$\sum_{i=1}^{B} pi = \frac{B\,(B - 1)}{2LB}$$

Da mesma maneira, considerando-se agora o intervalo (b) e a expressão anterior de "pi" em função de Pr (X) e Pr (Y), vem que:

$$\sum_{i=B+1}^{L} pi = \sum_{i=B+1}^{L} Pr(X) \times Pr\,(Y) = \sum_{i=B+1}^{L} \frac{1}{L} \times \frac{2B - (i - 1)}{B} \quad e$$

para tanto:

$$\sum_{i=B+1}^{L} pi = \frac{1}{LB} \left[B + (B-1) + (B-2) + \ldots + (2B - (L-1)) \right] \therefore$$

$$\sum_{i=B+2}^{L} pi = \frac{1}{2LB} (3B - L + 1) (L - B)$$

Consequentemente, para "P_2" pode-se escrever:

$$P_2 = \frac{1}{2LB} \times \left| B (B-1) + (3B - L + 1) (L - B) \right| \quad (1.6) \quad \text{c.q.d.}$$

(3) Cálculo de P_3:

Seja P_3, a probabilidade de se encontrar no terceiro dos blocos onde se distribui a lista L, um dado elemento de L (Lista de Sinônimos). Seja agora "pi" a probabilidade do registro procurado ser o de ordem "i" da lista de comprimento L, e dele estar armazenado no terceiro bloco (bloco 3), em que se encontra distribuída aquela lista.

Defina-se os dois eventos:

X = [o registro procurado ser o i-ésimo da lista]

Y = [o i-ésimo da lista estar no 3º bloco]

Considerando-se os eventos X e Y acima, pode-se dizer que a probabilidade "pi", definida para "P_3", é igual a probabilidade de ocorrerem simultaneamente os eventos X e Y, os quais são independentes de modo análogo ao que foi visto quando dos cálculos de P_1 e P_2. Assim pode-se escrever

$$pi = Pr (X \quad Y) = Pr (X) \times Pr (Y)$$

ainda, quando dos cálculos de P_1 foi visto que $Pr(X=1)=\frac{1}{L}$, pois todos os elementos de "L" são equiprováveis.

Resta, então, definir-se Pr(Y). Seguindo o mesmo processo indutivo utilizado para a alternativa em que $L < B$, discutida anteriormente, chega-se a conclusão, tomando-se os dois intervalos abaixo, que:

a) $Pr(Y = i) = \frac{i - (B + 1)}{B}$ para $B + 1 \leq i \leq L$; e

b) $Pr(Y = i) = 0$ para $1 \leq i \leq B$

Sendo "$i-(B+1)$" o número de ocorrências favoráveis do evento "Y" no intervalo (a).

Por conseguinte, pode-se escrever que:

$$P_3 = \sum_{i=1}^{L} pi = \sum_{i=1}^{B} pi + \sum_{i=B+1}^{L} pi = 0 + \sum_{i=B+1}^{L} pi$$

Considerando-se a expressão anterior de "pi" em função de Pr(X) e Pr(Y), vem:

$$P_3 = \sum_{i=B+1}^{L} pi = \sum_{i=B+1}^{L} Pr(X=i) \times Pr(Y=i) = \sum_{i=B+1}^{L} \frac{1}{L} \times \frac{i-(B+1)}{B} =$$

$$= \frac{1}{LB} \times [\, 0 + 1 + 2 + \ldots + (L - (B + 1)) \,]$$

Então:

$$P_3 = \frac{1}{2LB} \times (L - (B + 1)) \times (L - B) \qquad (1.7) \qquad \text{c.q.d.}$$

Como verificação final, pode-se ver que a soma das três probabilidades P_1, P_2, P_3 é igual a 1 (um), demonstrando a correção das expressões daquelas probabilidades.

Discussão de Alternativas

Conforme observou-se anteriormente, caso se tenha que manter a tabela em disco, em situação ideal (L = 1 e consequentemente o comprimento da tabela N = M), o número esperado de acessos a disco seria minimizado (E (C) = 2) para cada recuperação (um acesso ao arquivo Tabela de Índices e um outro ao Arquivo Mestre - ver figura (III.1) - Capítulo (III).

Então, torna-se desejável determinar o limite da amplitude de L, para a qual pode a tabela ser mantida na memória em situação não pior que E(C) = 2.

Ora, para $B \leq L \leq 2B + 1$, tem-se:

$$E(C) = P_1 + 2P_2 + 3P_3 \quad \text{ou seja} \quad E(C) = P_1 + 2P_2 + 3[1-(P_1+P_2)]$$

Para E(C) = 2 e fazendo-se as simplificações na expressão acima, tem-se:

$$2P_1 + P_2 = 1$$

Como $P_1 + P_2 + P_3 = 1$, vem que: $2P_1 + P_2 = P_1 + P_2 + P_3$, assim, neste caso particular, deduz-se que $P_1 = P_3$.

Então, substituindo-se os valores de P_1 e P_2 dados respectivamente pelas expressões (I.5) e (I.6) deste anexo, faz-se:

$$1 = 2 \times \frac{B+1}{2L} + \frac{1}{2LB} \quad | \; B(B-1) + 3B - L + 1)(L-B) \; | \; \therefore$$

$$2LB = 2B^2 + 2B + B^2 - B + 3LB - L^2 + L - 3B^2 + LB - B \; \therefore$$

$$\therefore L(L - 2B - 1) = 0 \quad \text{cujas raízes são:}$$

$L = 0$ (solução que é despresível)

e

$$L = 2B + 1 \qquad (1.8)$$

Então, pode-se concluir que, para um dado M, sendo toda a tabela mantida na memória de modo que sua dimensão $N = \frac{M}{L}$ permaneça no intervalo definido por $1 \leq L \leq 2B + 1$, o número esperado de acessos a disco é mantido no intervalo $1 \leq E(C) \leq 2$, ou seja, o valor máximo esperado para "C" (pior caso para aquele intervalo de L) é igual ao melhor caso ($L = 1$), para o qual a tabela encontra-se em disco, conforme já mencionado.

Entretanto, para processadores lentos, a pesquisa linear sobre a lista pode consumir tempo não desprezível em relação ao tempo total de recuperação, quando L assume valores relativamente grandes.

Considerando-se:

- T_D - tempo médio de leitura em disco e transmissão à memória, englobando os tempos de SEEK + 1/2 rotação do disco + transmissão à memória + overhead do sistema consumido pelas operações de controle (SEVERANCE[27]);
- t - tempo para consultar cada registro da lista de sinônimos, já na memória; e
- T_L - tempo médio consumido para, aleatoriamente, localizar registros de L. ($T_L = 1/2 \times L \times t$).

Analisa-se qual o L máximo que permite manter-se a tabela na memória em condições favoráveis, ou iguais, em relação àquela mais favorável quando a "Tabela de Índices" é mantida em disco, ou seja E(C) = 2;

Então, no limite, pode-se igualar os tempos consumidos em cada caso:

$$1/2\ L.t.f\ E(C)\ .T_D = 2T_D$$

onde E(C) aqui representa o número médio (valor esperado) de acessos ao "Arquivo Mestre".

Daí, pode-se escrever que:

$$E(C) = 2 - \frac{t.L}{2T_D}$$

Fazendo $X = t/T_D$, tem-se:

$$\overline{C} = 2 - \frac{x.L}{2}$$

considerando a expressão anterior que determina E(C) quando $B \leq L \leq B + 1$, pode-se fazer:

$$E(C) = P_1 + 2P_2 + 3P_3 = 2 - \frac{x.L}{2}$$

somando e subtraindo P_1 e desmembrando $3P_3$ em $2P_3 + P_3$, tem-se:

$$2(P_1 + P_2 + P_3) + P_3 - P_1 = 2 - \frac{x.1}{2}, \text{ ou}$$

$$P_1 - P_3 = \frac{x.L}{2}$$

Entretanto, conforme as expressões (I.5) e (I.7), pode-se dizer que:

$$P_1 - P_3 = \frac{B+1}{2L} - \frac{1}{2LB} \times (L - (B+1))\ (L - B) \therefore$$

$$P_1 - P_3 = \frac{1}{2LB}\ (< (2B + 1) - L^2), \text{ consequentemente}$$

$$\frac{xL}{2} = \frac{1}{2LB}\ (< (2B + 1) - L^2) \therefore$$

$$L\ (< (xB + 1) - (2B + 1)) = 0 \quad \text{o que resulta}$$

$$L = 0 \quad \text{(solução despresada) e}$$

$$L = \frac{2B + 1}{xB + 1}$$

Observe-se que, sendo "x" muito pequeno, ou seja $t << T_D$, é recomendável manter a "Tabela de Índices" na memória, com dimensão tal que garanta um comprimento médio de lista de sinônimos não superior a $(L \leq 2B + 1)$, reafirmando conclusão anterior, encerrada na expressão (I.8) deste anexo.

ANEXO II - ROTINAS EM PLTI

```
000A 1  [VETRTB (1) BYTE,(BLI,BII,BMT,B133) ADDRESS; ]
000A 1  ECRATB : PROCEDURE (ENT,RLI,RII,INICAT,COMPAT,HASH)
000A 1                 RESULT(ENT,RLI,RII,INICAT,COMPAT,HASH);
001B 2  % AUTOR EDUARDO CARNEIRO CAMPELO
001B 2  % OBJETIVO-RECEBENDO EM ENT O NUM.ORDEM DA ENTRADA A' TABELA DE DEFINI-
001B 2  %        CAO REG.MESTRE O QUAL CORRESPONDE AO ATRIBUTO QUE ESTA' SENDO
001B 2  %        TRATADO, PROCURA LOCALIZAR O PRIMEIRO REG. DA LI A QUE PERTEN-
001B 2  %        CE ,ATRAVES DE CONSULTA INTERMEDIARIA AO CORRESPONDENTE REG. DE
001B 2  %        DECJII. PARA ISSO USA FUNCAO HASHING SOBRE O VALOR DO ATRIBUTO,
001B 2  %        BEM COMO O MODULO EM (DF.REG.MESTRE) PARA TAL ATRIBUTO.
001B 2  %        VER FINALIDADE DOS PARAMETROS EM SUAS DECLARACOES ABAIXO:
001B 2  %        OPCOES DE RESPOSTA :      1- ENT = #FF PROCEDIMENTO BEM SUCEDIDO
001B 2  %        RLI > 0  ENDERECO DA LI INICIALIZADA, RESULTANDO PARA OS OUTROS
001B 2  %        PARAMETROS ...  RII - N.REG EM DECJII / INICAT = INICIO DO ATRI-
001B 2  %        BUTO /COMPAT = COMPRIMENTO DO ATRIBUTO /  HASH = RESULTADO  DA
001B 2  %        FUNCAO HASHING SOBRE O ATRIBUTO //      2- ENT = #FA....PROCEDI-
001B 2  %        MENTO MAL SUCEDIDO, RESULTANDO ...RLI = 'N' ATRIBUTO NAO EXIS-
001B 2  %        TENTE / RLI = 'C' CHAVE PRIMARIA SENDO INICAP E COMPAT CARREGA-
001B 2  %        DOS CORRETAMENTE / RLI = '@' OUTROS ATRIBUTOS NAO CHAVE SECUN-
001B 2  %        DARIA, INICAT E COMPAT CARREGADOS / RLI='K' VALOR ATRIBUTO COM
001B 2  %        ESPACO BRANCO / RLI='L' RII CHEIO E ATRIBUTO NAO ENCONTRADO;
001B 2  %             3- ENT <> #FF E RLI=0  LISTA VAZIA, ENT CONTEM POSICAO EM
001B 2  %        RII (REG.TAB.INDICE PARA LI) ONDE DEVE SER APONTADA A FUTURA LI
001B 2  %        (O HASH DEVE SER COLOCADO EM RLI+128); RII,INICAT E COMPAT SAO
001B 2  %        CARREGADOS. //////////
001B 2  %*************************************************************************
001B 2  %
001B 2  DECLARE
001B. 2      ENT          BYTE,       %NUM. DE ORDEM DA ENTRADA TABELA
001B. 2      RLI          ADDRESS,    %N. REG DECJLI
001B. 2      RII          ADDRESS,    %N. REG DECJII
001B. 2      INICAT       ADDRESS,    %POS. INICIAL DO ATRIBUTO
001B. 2      COMPAT       ADDRESS,    %COMPRIMENTO DO ATRIBUTO
001B. 2      AREAII  BASED BII (255) ADDRESS, % BUFFER DECJII
001B. 2      AREAMT  BASED BMT (31)  BYTE,    % BUFFER DECJ20 MESTRE
001B. 2  %   VETRTB  BASED BTB (320) BYTE,    % VETOR TABELA DFREG.MESTRE
001B. 2                                       % POS.ZERO DA' N.ENTRADAS
```

```
001B. 2       AREA133 BASED B133 (132) BYTE,
001B. 2       IX              ADDRESS,     %DESLOCAMENTO DA ENTRADA DA TABELA
001D. 2       HASH            ADDRESS,
001D. 2       COMP            ADDRESS,
001F. 2       INIC            ADDRESS,
0021. 2       MODII           ADDRESS,     % MODULO PARA DECJII
0023. 2       II              ADDRESS,
0025. 2       III             ADDRESS,
0027. 2       KI              ADDRESS,
0029. 2       JI              ADDRESS;
002B. 2   DECLARE
002B. 2       (DECJII, IMPRES)              FILE,
002B. 2       (RETORNO, ERRO1, VAZIO, BUSCALI, REPETE, ERRO2, ERRO3)          LABEL;
002B. 2   %-------------------------------------------------------------------------
002B 2    ECCG    : PROCEDURE (ENTB, INICATB, COMPATB)
002B 2                RESULT (ENTB, INICATB, COMPATB);
0036  3   DECLARE
0036. 3       ENTB            BYTE,
0036. 3       (CGAT, INICATB, COMPATB)  ADDRESS;
0038. 3       IF  VETRTB(0) < ENTB THEN DO;
0045  5       ENTB = #FA;
004B  5       RETURN;
0053  5   END;
0053  4     ELSE
0056  4       CGAT = ((ENTB - 1)* 16) +1;
0066  4       MOVE VETRTB(CGAT + 4) TO INICAT FOR 2;
0076  3       MOVE VETRTB(CGAT + 6) TO COMPAT FOR 2;
0086  3   END  PROCEDURE-ECCGAT;
008E  2   %-------------------------------------------------------------------------
008E  2   ECCGAT: PROCEDURE (ENTB,  INICATB,  COMPATB) ENTRY
008E  2                 RESULT (ENTB,  INICATB,  COMPATB);
00A3  3   DECLARE
00A3. 3       ENTB BYTE,
00A3. 3       (INICATB,  COMPATB) ADDRESS;
00A3. 3       CALL ECCG (ENTB,  INICATB,  COMPATB);
```

```
00B2 3  END;
00BA 2  %-------------------------------------------------------------------------
00BA 2  ECHASH : PROCEDURE(HASHSAI)    RESULT(HASHSAI);
00C1 3  %               FAZ HASHING DO VALOR DO ATRIBUTO QUALQUER QUE SEJA SEU
00C1 3  %               COMPRIMENTO, PARA UM CAMPO ADDRESS (DOIS BYTES), USANDO
00C1 3  %               OU EXCLUSIVO(XOR) DESLOCAMENTO DE UM BIT A ESQUERDA EM
00C1 3  %               NUMERO IGUAL AO DE CARACTERES DO ATRIBUTO.
00C1 3  %
00C1 3  DECLARE
00C1 3      (RETORNOHS)         LABEL,
00C1 3      HASHSAI      ADDRESS,
00C1 3      IH           ADDRESS; % COMPRIMENTO ATRIB  OU INDICA ERRO=#FFFF
00C3 3  %-------------------------------------
00C3 3      IH =0;
00C9 3      HASHSAI = AREAMT(INIC);
00D4 3  DO WHILE    IH < COMP  AND AREAMT(INIC + IH) <> ' ';
00ED 4      HASHSAI = (HASHSAI <- 1) XOR AREAMT(INIC + IH);
0103 4      IH = IH + 1;
010D 4      IF  AREAMT(INIC+IH) <> ' ' THEN GO TO RETORNOHS;
0121 5      MOVE ' ERRO- COMPRIMENTO ATRIBUTO MENOR QUE DEFINIDO-' TO AREA133;
0159 4      MOVE AREAMT(INIC) TO AREA133(48) FOR IH;
016A 4      WRITE IMPRES FROM AREA133;
0171 4      HASHSAI = #FFFF;
0178 4  RETORNOHS :
0178 4  END  DO-WHILE;
017B 3  END   PROCEDURE-ECHASH;
0181 2  %-------------------------------------------------------------------------
0181 2      CALL  ECCG  (ENT, INICAT, COMPAT);
0190 2      IF  ENT= #FA  THEN DO;
0199 4      RLI = 'N';
019F 4      ENT, RII, INICAT, COMPAT = #FA;
01B1 4      GO TO RETORNO;
01B4 4  END  DO-IF;
01B4 3      IX =((ENT - 1) * 16) + 1;
01C4 2      IF VETRTB(IX + 3) = 'C' THEN
```

```
01D4 3  DO;
01D4 4      RLI ='C';
01DA 4      ENT.RII = #FA;
01E4 4      MOVE VETRTB(IX + 4) TO INICAT FOR 2;
01F4 4      MOVE VETRTB(IX + 6) TO COMPAT FOR 2;
0204 4      GO TO RETORNO;
0207 4  END  DO;
0207 3      IF VETRTB(IX+3) <>'1'THEN
0217 3  DO;
0217 4      RLI = '*';
021D 4      ENT.RII = #FA;
0227 4      MOVE VETRTB(IX + 4) TO INICAT FOR 2;
0237 4      MOVE VETRTB(IX + 6) TO COMPAT FOR 2;
0247 4      GO TO RETORNO;
024A 4  END  DO;
024A 3  ELSE
024D 3      MOVE VETRTB(IX+4) TO INIC FOR 2;
025D 3      MOVE VETRTB(IX+6) TO COMP FOR 2;
026D 2      JI = 0;
0273 2  DO WHILE  JI < COMP;
027D 3      IF AREAMT(INIC + JI) = ' ' THEN JI = 1000;
0295 4      ELSE   JI = JI + 1;
02A2 4  END  DO-WHILE;
02A5 2      IF JI = 1000 THEN  GO TO ERRO3;
02B2 3      CALL ECHASH(HASH);
02BB 2      MOVE VETRTB(IX + 10) TO MODII FOR 2;
02CB 2      II = HASH MOD MODII;
02D6 2      III = II MOD 128;
02E0 2      II = II / 128;
02EA 2      MOVE VETRTB(IX + 12) TO KI FOR 2;    %PRIMEIRO REG DA TAB. LI ATRIB
02FA 2      MOVE VETRTB(IX + 14) TO MODII FOR 2;%QUANT. REG TAB. IND.  LI
030A 2      IF (II+1) > MODII THEN GO TO ERRO1;
031A 3      II = (II + KI);
0325 2      SEEK DECJII AT II;
032C 2      READ DECJII INTO AREAII;
```

```
0333 2      JI = III;
033A 2  REPETE :
033A 2      IF AREAII(III) = 0 THEN GO TO VAZIO;
034A 3      IF AREAII(III + 128) = HASH THEN GO TO BUSCALI;
035E 3      III = III + 1;
0368 2      IF III = JI THEN GO TO ERRO2;
0375 3      IF III> 127 THEN III = 0;
0384 3      GO TO REPETE;
0387 2  VAZIO :
0387 2      RLI = 0;
038D 2      RII = II;
0394 2      ENT = III;
039B 2      INICAT = INIC;
03A2 2      COMPAT = COMP;
03A9 2      GO TO RETORNO;
03AC 2  BUSCALI :
03AC 2      RII = II;              % N.REG DE DECJII
03B3 2      ENT = #FF;             % LOCAL EM REG DECJII CONTEM APONT LI
03B9 2      INICAT = INIC;
03C0 2      COMPAT = COMP;
03C7 2      RLI = AREAII(III);   % PRIMEIRO REG LI DA LISTA
03D2 2      GO TO RETORNO;
03D5 2  ERRO1 :
03D5 2      MOVE '1N.REG II RESERVADOS INCOMPATIVEL COM MODULO PARA ATR-'
03D5 2          TO AREA133;
0414 2      JI = 56;
041A 2  DO WHILE ENT > 0;
0423 3      AREA133(JI) = (ENT MOD 10) + '0';
0434 3      ENT = ENT / 10;
043E 3      JI = JI - 1;
0448 3  END  DO-WHILE;
044B 2      WRITE IMPRES FROM AREA133;
0452 2      EXIT;
0453 2  ERRO2 :
0453 2      MOVE '1ATRIBUTO NAO LOCALIZADO - REG DECJII ENCONTRA-SE CHEIO-'
```

```
0453 2                  TO AREA133;
0494 2       JI = 59;
049A 2  DO WHILE  ENT > 0;
04A3 3       AREA133(JI) = (ENT MOD 10) + '0';
04B4 3       ENT = ENT / 10;
04BE 3       JI = JI - 1;
04C8 3  END  DO-WHILE;
04CB 2       WRITE IMPRES FROM AREA133;
04D2 2       ENT = #FA;
04D8 2       RLI = 'L';
04DE 2       GO TO RETORNO;
04E1 2  ERRO3 :
04E1 2       ENT = #FA;
04E7 2       MOVE ' ERRO-ATRIBUTO NAO PODE TER ESPACO BRANCO -.....' TO AREA133;
0520 2       WRITE IMPRES FROM AREA133;
0527 2       RLI = 'K';
052D 2  RETORNO :
052D 2  END  PROCEDURE-ECRATB;
```

0 - PROBLEMA NA CRIACAO DO ARQUIVO : DEIXA OBJETO NA AREA DE TRABALHO <-

```
000A 1  [VETRTB (1) BYTE,(BLI,BII,B133) ADDRESS; ]
000A 1  ECTBMT : PROCEDURE;
000F 2  % AUTOR EDUARDO CARNEIRO CAMPELO JR.
000F 2  %       SUBROTINA PARA CARREGAR E LISTAR TABELA DE DEFINICAO DO REG.
000F 2  %       MESTRE DE DADOS DO ARQUIVO DECJ20 - A TABELA E'LIDA DO ARQ.
000F 2  %       DECJTB.   CADA ENTRADA NA TABELA E'CONSTITUIDA DE 16 BYTES ASSIM
000F 2  %       DISTRIBUIDOS : SIMBOLO=3 BYTES,TIPO=1 B,INICIO=COMPRIMENTO=
000F 2  %       BASE=MODULO=REG LI=QT REG LI= ADDRESS NESTA ORDEM DENTRO DO VE-
000F 2  %       TOR TABELA;   VERIFICA COMPATIBILIDADE DOS ARQUIVOS DECJTB  E
000F 2  %       DECJLI.
000F 2  DECLARE
000F. 2 %-------------BII                    % LOCAL DO REG TABELA-BASED S/ REG II
000F. 2 %-------------BLI                    % LOCAL DO REG LISTA INVERTIDA-BASED,
000F. 2     IX                  ADDRESS INITIAL(00),
0011. 2     KX                  BYTE    INITIAL(0),
0012. 2     TX                  ADDRESS,
0014. 2     KQTREGTAB           ADDRESS,
0016. 2     IV                  ADDRESS,    %NUMERO DE ENTRADAS DA TABELA
0018. 2     LOCTAB              ADDRESS,
001A. 2 %   VETORTD   (320)     BYTE,       % N. ENTRADAS GUARDA NA POS.  ZERO
001A. 2     IMPRE  LITERALLY 'IMPRES',
001A. 2 1  LINHA133 BASED B133,
001A. 2     2 CONT              BYTE,
001A. 2     2 ESCRITA (131)     BYTE;
001A. 2 DECLARE
001A. 2     (IMPRES,DECJTB,DECJLI)      FILE;
001A. 2 %-----------------------------LINHAS DO CABECALHO RELATORIO-TABELA
001A 2  DECLARE
001A. 2     LINHA1      (51) BYTE INITIAL('TABELA DEFINICAO DO REG MESTRE DE',
003B. 2                                   ' DADOS -ARQ  ECCJ20'),
004E. 2     LINHA2      (51) BYTE INITIAL('QUANT. BYTES UTEIS XXX COM XXX CAMPOS',
004E. 2                                   '/DATA REF. XXXXXX'),
0082. 2     LINHA3      (51) BYTE INITIAL('REFER DESCRICAO.....  TIPO LOCAL COMPR',
0082. 2                                   ' V. BASE MODULO'),
                                                          ^
```

```
153 - NEM TODAS AS POSICOES FORAM INICIALIZADAS : ADVERTE                <-

00B6. 2       (ERRO1, SEGUEDO)          LABEL;
00B6. 2   %     ESTRUTURA DOS ARQUIVOS
00B6 2    DECLARE
00B6. 2   1   DEFINREGMT BASED   BII,
00B6. 2       2 NORDEMMT        ADDRESS,          %NUMERO DE ORDEM - CARATER
00B6. 2       2 SIMBATRIBMT (2) BYTE,             %SIMBOLO DO ATRIBUTO
00B6. 2       2 DENOMINATB  (14)BYTE,             %DENOMINACAO DO ATRIBUTO
00B6. 2       2 INICATR         (2)   BYTE,       %POSICAO INICIAL DO ATRIBUTO
00B6. 2       2 INICATRH              ADDRESS,    %POSICAO INICIAL EM HEXA
00B6. 2       2 COMPRIMMT       (1)   BYTE,       %COMPRIMENTO DO CAMPO DO ATRIBUTO
00B6. 2       2 ESPECIEMT             BYTE,       %ESPECIE DO ATRIBUTO
00B6. 2       2 BASEMT          (4)   BYTE,       %BASE SE ESPECIE=3
00B6. 2       2 MODTABLI        (2)   BYTE,       %MODULO TAB INDICE DE L. INVERTIDA
00B6. 2       2 COMPRIMMTH            ADDRESS,    %COMPRIMENTO DO CAMPO EM HEXA
00B6. 2       2 BASEMTH             ADDRESS,      %BASE EM HEXA
00B6. 2       2 MODTABLIH           ADDRESS,      %MODULO TAB INDICE LI HEXA
00B6. 2       2 ENDTABLI            ADDRESS,      %APONTA REG TAB INDICE LI DO ATRIBUTO
00B6. 2       2 QTREGINDLI          ADDRESS,      %QUANT. RESERVADA DE REG DO DECJII
00B6. 2       2 TAMPAOMT    (4)     BYTE,         %DISPONIVEL PARA ACRESCENTAR CAMPOS
00B6. 2   1  HEADTABDF             BASED BII,
00B6. 2       2 QTREGTABDF (2)      BYTE,         %QUANTIDADE REGISTROS DA TABELA S/HEADR
00B6. 2       2 COMPREGMT  (2)      BYTE,         %COMPRIMENTO DO REG MESTRE
00B6. 2       2 COMPMAXATB (2)      BYTE,         %COMPRIMENTO MAXIMO DE ATRIBUTO
00B6. 2       2 QTREGTABDFHX        ADDRESS,      %QTREGTABDF EM HEXA S/ HEADER
00B6. 2       2 COMPMAXATBHX        ADDRESS,      %COMPMAXTB EM HEXA
00B6. 2       2 COMPREGMTHX         ADDRESS,      %COMPREGMT EM HEXA
00B6. 2       2 DATAGRVTAB (5)      BYTE,         %DATA (AAMMDD) DE GRAVACAO DA TABELA
00B6. 2       2 NOMEARQTBDF (27)    BYTE,         %NOME DO ARQ.  DECJTB
00B6. 2       2 TAMPAOTABDF (1)     BYTE,         %ESPACO COMPLEMENTO PARA 51 BYTES
00B6. 2   %-----------------------------------------
00B6 2    %--------------ARQUIVO LISTA INVERTIDA - DECJLI
00B6 2    %               REG  64  BYTES
```

```
00B6 2  DECLARE
00B6. 2 %------------PRIMEIRO REG.  DE LISTA DE CADA VALOR DE ATRIBUTO
00B6. 2 1    LISTAINVERTE      BASED  BLI,
00B6. 2      2 TAMPAOLI   (2) BYTE,          %ESPACO REDEFINIDO
00B6. 2      2 PROXLISIN      ADDRESS,       %APONTA PROX LI DE VALOR SINONIM
00B6. 2      2 COMPATRIB      BYTE,          %COMPRIMENTO DO VALOR DO ATRIBUTO
00B6. 2      2 COMPCHAVE      BYTE,          %COMPRIMENTO DA CHAVE PRIMARIA
00B6. 2      2 VALORATRIB (56)BYTE,          %VALOR ATRIBUTO REDEFINIDO
00B6. 2 %-------------------REDEFINICAO REG EXPANSAO
00B6. 2 1    LISTAINVERT2       BASED BLI,
00B6. 2      2 LKEXPANLI      ADDRESS,       %APONTA PROX REG EXPANSAO DA LI
00B6. 2      2 QTUTILIZLI     BYTE,          %QUANT. CAMPO CHAVE ENTIDADE UTILIZA
00B6. 2      2 CHAVEIDLI (60)BYTE,           %IDENTIFICADOR ENTIDADE = VALOR ATRIB
00B6. 2                                      %ISTO E' CHAVE PRIMARIA
00B6. 2 %-------------------REDEFINICAO PARA HEADER
00B6. 2 1    HEADLISTAINVERT    BASED BLI,
00B6. 2      2 COMPLI         ADDRESS,       %COMPRIMENTO DO REG LI
00B6. 2      2 OCORRCHENT     ADDRESS,       %NUM OCORRENCIAS CAMPO CHAVE ENTIDADE
00B6. 2      2 NOMEARQLI (29)BYTE,
00B6. 2      2 QTREGLI        ADDRESS,       %QUANT. REG LI CRIADOS COM HEADER
00B6. 2      2 QTREGUTILI     ADDRESS,       %QUANT. REG LI UTILIZADOS
00B6. 2      2 QTDISPLI       ADDRESS,       %QUANT. REG LI A UTILIZAR
00B6. 2      2 DATAGRTABLI(5)BYTE,           %(AAMMDD) DE GRAVACAO DO DECJTB ORIGEM
00B6. 2      2 TAMPAOHDLI (17)BYTE;          %COMPLEMENTO PARA 51 BYTES
00B6. 2 %-------------------------------------------------------------------------
00B6 2  %-------------------INICIO PROCESSAMENTO
00B6 2.     READ DECJTB INTO HEADTABDF;
00BD 2      READ DECJLI INTO HEADLISTAINVERT;
00C4 2  DO  TX=0 TO 5;
00CF 3      IF DATAGRTABLI(TX) <> DATAGRVTAB(TX) THEN KX=1;
00E9 4  END  DO;
00EA 2      IF  KX = 1  THEN
00F3 3  DO;
00F3 4      MOVE 'ARQUIVOS DECJTB E DECJLI COM DATAS DIFERENTES' TO ESCRITA;
012A 4      WRITE IMPRE FROM LINHA133;
```

```
0131 4       EXIT;
0132 4   END  DO;
0132 3       KQTREGTAB = QTREGTABDFHX;
013A 2       MOVE LINHA1 TO ESCRITA FOR 52;
0144 2       WRITE IMPRE FROM LINHA133;
014B 2       CONT = ' ';
0151 2       MOVE LINHA2 TO ESCRITA FOR 52;
015B 2       MOVE COMPREGMT(0) TO ESCRITA(10) FOR 3;
016C 2       MOVE QTREGTABDF(0) TO ESCRITA(26) FOR 3;
017C 2       MOVE DATAGRVTAB(0) TO ESCRITA(46) FOR 6;
018D 2       WRITE IMPRE FROM LINHA133;
0194 2       MOVE CONT TO ESCRITA(0) FOR 132;
01A1 2       ESCRITA(0),ESCRITA(22),ESCRITA(33),ESCRITA(39) = '*';
01C3 2       WRITE IMPRE FROM LINHA133;
01CA 2       MOVE LINHA3 TO ESCRITA FOR 52;
01D4 2       WRITE IMPRE FROM LINHA133;
01DB 2       MOVE CONT TO ESCRITA(0) FOR 60;
01E8 2       IV = 1;
01EE 2       CONT = '0';
01F4 2   DO  IX=2 TO QTREGTABDFHX;
0201 3       READ DECJTB INTO DEFINREGMT EOF ERRO1;
020C 3       MOVE SIMBATRIBMT(0) TO ESCRITA(1) FOR 3;
021D 3       MOVE DENOMINATB(0) TO ESCRITA(6) FOR 15;
022E 3       MOVE ESPECIEMT TO ESCRITA(23);
023D 3       MOVE INICATR(0) TO ESCRITA(28) FOR 3;
024E 3       MOVE COMPRIMMT(0) TO ESCRITA(34) FOR 2;
025F 3       MOVE BASEMT(0) TO ESCRITA(39) FOR 5;
0270 3       MOVE MODTABLI TO ESCRITA(47) FOR 3;
027E 3       WRITE IMPRE FROM LINHA133;
0285 3       CONT = ' ';
028B 3       IF IV > 320 THEN GO TO ERRO1;
0299 4       MOVE SIMBATRIBMT(0) TO VETRTB(IV) FOR 3;
02AA 3       VETRTB(IV+3) = ESPECIEMT;
02B9 3       MOVE INICATRH TO VETRTB(IV+4) FOR 2;
02CA 3       MOVE COMPRIMMTH TO VETRTB(IV+6) FOR 2;
```

```
02DB 3       MOVE BASEMTH TO VETRTB(IV+0) FOR 2;
02EC 3       MOVE MODTABLIH TO VETRTB(IV+10) FOR 2;
02FD 3       MOVE ENDTABLI TO VETRTB(IV+12) FOR 2;
030E 3       MOVE QTREGINDLI TO VETRTB(IV+14) FOR 2;
031F 3       IV = IV + 16;
0329 3       GO TO SEGUEDO;
032D 3   ERRO1 :
032D 3       MOVE 'ERRO TAMANHO DA TABELA- > 20 OU > QUANT. REG INFORMADA' TO
0364 3                                              ESCRITA;
036C 3       WRITE IMPRE FROM LINHA133;
0373 3       EXIT;
0374 3   SEGUEDO :
0374 3   END  DO;
0375 2       VETRTB(0) = KQTREGTAB - 1;  % GUARDA N. ENTRADAS DA TABELA
0382 2   END  PROCEDURE-ECTBMT;
0383 1   %
```

```
000A 1  ['VETRTB (1) BYTE, (BLI, BII, BMT, BTB) ADDRESS, ARA133(1) BYTE; ]
000A 1  ECISLI : PROCEDURE (RLI, CODATB, RII, ALTER)  RESULT(RLI, CODATB);
0017 2  %      CONSULTA, INSERE OU EXCLUI EM LISTA INVERTIDA (LI) CHAVE
0017 2  %      DO REGISTRO EM TRATAMENTO CONFORME INDICAR ALTER (C/I/E)
0017 2  %-------------------------------------------------------------
0017 2  %
0017 2  %-------------ESTRUTURA DO ARQUIVO LISTA INVERTIDA - DECJLI
0017 2  %              REG.   DE 64 BYTES
0017 2  DECLARE
0017 2  %----------PRIMEIRO REGISTRO DE CADA LISTA PARA CADA VALOR DE ATRIBUTO
0017 2  1 LISTAINVERTE  BASED BLI,        % VARIAVEL GLOBAL
0017 2    2 TAMPAOLI   (2)  BYTE,         % ESPACO REDEFINIDO
0017 2    2 PROXLISIN       ADDRESS,      % APONTA INICIO PROX LI DE VALOR SINONIM
0017 2    2 COMPATRIB  BYTE,              % COMPRIMENTO DO VALOR DO ATRIBUTO
0017 2    2 COMPCHAV   BYTE,              % COMPRIMENTO DA CHAVE PRIMARIA
0017 2    2 VALORATRIB (56) BYTE,         % VALOR ATRIBUTO
0017 2  %----------REDEFINICAO
0017 2  %----------REGISTRO DE EXPANSAO DE LISTA INVERTIDA (LI)
0017 2  1 LISTAINVERTE2     BASED  BLI,
0017 2    2 LKEXPANLI       ADDRESS,      % APRONTA PROX REG EXPANSAO DA LI
0017 2    2 QTUTILIZLI BYTE,              % QUANT DE CAMPO CHAVE ENTIDADE UTILIZA
0017 2    2 CHAVEIDLI  (60) BYTE,         % IDENTIFICADOR ENTIDADES = VALOR ATRIB
0017 2                                    % ISTO E' CHAVE PRIMARIA
0017 2  %----------REGISTRO RHEADER
0017 2  1  HEADLISTAINVERT,
0017 2    2 COMPLI          ADDRESS,      % COMPRIMENTO DO REG LI
0019 2    2 OCORRCHENT      ADDRESS,      % NUM. MAX.  OCORR.  CAMPO CHAVE ENTIDADE
001B 2    2 NOMEARQLI (29)  BYTE,
0039 2    2 QTREGLI         ADDRESS,      % QUANTIDADE REG LI CRIADOS C/ HEADER
003B 2    2 QTREGUTILI      ADDRESS,      % QUANT.  REG LI UTILIZADOS
003D 2    2 QTDISPLI        ADDRESS,      % QUANT.  REG LI A UTILIZAR
003F 2    2 DATAGRTABLI (5) BYTE,         % (AAMMDD) DE GRAVACAO DO DECJTB ORIGEM
0045 2    2 TAMPAOHDLI (17) BYTE;         % COMPLEMENTO PARA 64 BYTES
0057 2  %
0057 2  DECLARE
```

```
0057. 2  %----------ARQUIVO INDICE PARA LISTA INVERTIDA   -  DECJII
0057. 2  %           REG. DE 512 BYTES  -  4 BYTES/ENTRADA
0057. 2      INDICEII  BASED BII (255) ADDRESS,  %AREA DE LEITURA DE REG II
0057. 2      REGMEST   BASED BMT (31)  BYTE;     %AREA DE LEITURA DO REG MESTRE
0057. 2  %
0057 2   %
0057 2   %----------------------VARIAVEIS DE USO DIVERSO
0057 2   DECLARE
0057. 2      (I,K,J)           ADDRESS,
005D. 2      RLI    .          ADDRESS,     %REG. ONDE DEVE INICIAR A PROCURA
005D. 2      CODATB            ADDRESS,     %N. ORDEM DO ATRIB. NA TABELA DFMT
005D. 2      RII               ADDRESS,     %REG. II DO ATRIB. (TAB. IND. LI)
005D. 2      INICAT           ADDRESS,     %INICIO DO ATRIBUTO
005F. 2      COMPAT            ADDRESS,     %COMPRIMENTO DO ATRIBUTO
0061. 2      HASHII            ADDRESS,     %PARA COMPARAR E INSERIR EM REG II
0063. 2      KEY               BYTE   INITIAL ('1'),
0064. 2      ALTER             BYTE,        % =I INCLUI, =E EXCLUI, =C CONSULTA
0064. 2      LCHAVEP           BYTE,
0065. 2      INICCHP           ADDRESS,
0067. 2      (LISTVAZIA,RETORNOISLI,REPETE1,INSERC1,FALTALI,REPETE2)      LABEL,
0067. 2      (CONSULTNEGA,CONSULTAFIR,RETORNOIS,INSERE2)                  LABEL;
0067. 2  %----------------------DECLARACAO DE ARQUIVOS
0067 2   DECLARE
0067. 2      (DECJLI,DECJII,IMPRES)                                  FILE;
0067. 2  %
0067 2   %----------------------DECLARACOES  DE  ROTINAS
0067 2   %
0067 2   ECRATB  : PROCEDURE(ENT,B1,B2,B3,B4,HASH) PLTI
0067 2             RESULT   (ENT,B1,B2,B3,B4,HASH); END;
0067. 2  %--------------------------
0067 2   PEDELI  : PROCEDURE (ENDLI) RESULT (ENDLI);
006E  3  DECLARE
006E. 3      ENDLI   ADDRESS;
006E. 3      IF QTDISPLI = 0  THEN
0077 4   DO;
```

```
0077 5       ENDLI = 0;
007D 5       RETURN;
0083 5   END;
0083 4       ENDLI,QTREGUTILI = QTREGUTILI + 1;
0091 3       QTDISPLI = QTDISPLI - 1;
009B 3   END  PROCEDURE-PEDELI;
00A1 2   %-------------------------------------------------------------------
00A1 2   ECHDLI : PROCEDURE  ENTRY;
00B0 3   %       USO OBRIGATORIO EM TODAS AS ROTINAS DE FIM DE PROGRAMAS QUE
00B0 3   %       FACAM ATUALIZACAO DE LI'S PELO USO DA PROCEDURE ECISLI
00B0 3   %*******************************************************************
00B0 3   %
00B0 3       SEEK   DECJLI  AT  0;
00B6 3       WRITE  DECJLI  FROM  HEADLISTAINVERT;
00BD 3   END  PROCEDURE-ECHDLI;
00BE 2   %
00BE 2   %===================================================================
00BE 2   %
00BE 2   %-------------------------------------------------------------------
00BE 2   %-----------------------------INICIO  DO  PROCESSAMENTO
00BE 2       IF KEY = '1' THEN DO;
00C7 4       SEEK DECJLI AT 0;
00CD 4       READ DECJLI INTO HEADLISTAINVERT;
00D4 4       KEY ='*';
00DA 4           END  DO;
00DA 3       IF ALTER <> 'E'AND ALTER <>'I' AND ALTER <> 'C' THEN
00F1 3   DO;
00F1 4       CODATB = #FA;
00F7 4       RLI = 'B';
00FD 4       RETURN;
0104 4   END;
0104 3       CALL ECRATB(CODATB,RLI,RII,INICAT,COMPAT,HASHII);
011C 2       IF  CODATB <123 AND RLI=0  THEN GO TO LISTVAZIA;
012F 3       IF  CODATB=#FA AND (RLI='N' OR RLI='C' OR RLI='*' OR RLI='K'
014C 2           OR RLI='L') THEN RETURN;
```

```
0162 3     ELSE
0165 3       IF  CODATB=#FF AND RLI > 0 THEN GO TO REPETE1;
0178 4       CODATB = #FA; RLI = 'X';
0184 2       RETURN;
018B 2   %-------------------------------LISTA EXISTE
018B 2   REPETE1 :
018B 2       SEEK DECJLI AT RLI;
0192 2       READ DECJLI INTO LISTAINVERTE EOF FALTALI;
019C 2       K = 0;
01A2 2   DO I=0 TO (COMPAT - 1);
01B1 3       IF VALORATRIB(I) <> REGMEST(INICAT + I) THEN K=1;
01CE 4   END  DO;
01CF 2       IF K=1 THEN                  % NAO ENCONTROU ?
01D8 3   DO;                                          % DO-1
01D8 4       RLI = PROXLISIN;
01E0 4       IF RLI<>0 THEN GO TO REPETE1;
01EC 5   DO;                                          % DO-2
01EC 5       IF ALTER='C' THEN GO TO CONSULTNEGA;
01F8 6       CALL  PEDELI(I);
0201 5       IF I=0 THEN
020A 6   DO;                                          % DO-3
020A 7       CODATB = 'A';                           %INSERCAO NAO COMPLETADA-FALTA LI
0210 7       RETURN;
0217 7   END  DO-3;
0217 6       PROXLISIN = I;
021F 5       REWRITE DECJLI FROM LISTAINVERTE;
0226 5       RLI = I;
022D 5       GO TO INSERE1;                % PRIMEIRO REG PARA TAL VALOR A INSERIR
0230 5   END  DO-2;
0230 4   END  DO-1;
0230 3       IF ALTER='C' THEN GO TO CONSULTAFIR;
023C 3   %--------------PARA TAL VALOR A LISTA JA7 FOI INICIALIZADA; VERIFICA SE
023C 3   %--------------CHAVE JA EXISTE, CASO CONTRARIO INSERE SE HOUVER ESPACO.
023C 3       MOVE VETRTB(5) TO INICCHP FOR 2;% LOCAL INICIO CHAVE PRIMARIA
0248 2       LCHAVEP = VETRTB(8);          % COMPRIMENTO CHAVE PRIMARIA
```

```
0252 2       IF LCHAVEP <> COMPCHAV  THEN    % COMPRIMENTO CHAVE CORRETO ?
025D 3   DO;
025D 4       MOVE '1ERRO- DIVERGENCIA COMPRIMENTO CHAVE PRIMARIA DECJTB E DECJLI'
025D 4                TO ARA133;
02A3 4       MOVE 'REQUER RECARGA DE DECJLI- OCORR.REG .... ' TO ARA133(62);
02D7 4       K,I =0;
02E1 4   DO   WHILE RLI > 0;
02EA 5       ARA133(102 - I) = (RLI MOD 10) + '0';
02FE 5       RLI = RLI / 10;
0308 5       I = I+1;
0312 5   END  DO-WHILE;
0315 4       WRITE IMPRES FROM ARA133;
031C 4       EXIT;
031D 4   END  DO;
031D 3       I = COMPATRIB;
0325 2       K=1;
032B 2   DO  WHILE I < QTUTILIZLI AND K=1; % SE K=0 CHAVE EXISTE
033D 3       K=0;
0343 3     DO  J=0 TO (COMPCHAV - 1);
0353 4       IF VALORATRIB(I+J) <> REGMEST(INICCHP + J) THEN K=1;
0374 5     END;
0375 3       I = I + COMPCHAV;
0381 3   END  DO-WHILE;
0384 2       IF  K=0  THEN
038D 3   DO;                                  % CHAVE JA EXISTE NA LISTA
038D 4       IF ALTER = 'E' THEN
0396 5     DO;
0396 6       MOVE VALORATRIB(QTUTILIZLI - COMPCHAV) TO VALORATRIB(I-COMPCHAV)
03B1 6                                                FOR COMPCHAV;
03B6 6       QTUTILIZLI = QTUTILIZLI - COMPCHAV;
03C4 6       REWRITE DECJLI FROM LISTAINVERTE;
03CB 6     END;
03CB 5       CODATB = #FF;
03D1 4       RETURN;
03D8 4   END  JA  EXISTE;
```

```
03D8 3  %----------------CHAVE NAO EXISTE -VERIFICA SE HA' ESPACO PARA INSERI-LA
03D8 3  %                NO PRIMEIRO REG DA LISTA. SE FOR O CASO(ALTER='I').
03D8 3      IF ALTER = 'I' THEN          % QUER INSERIR ?
03E1 3          IF (QTUTILIZLI + LCHAVEP) < (COMPLI - 6)  THEN
03F3 4  DO;                              % HA' ESPACO
03F3 5      MOVE REGMEST(INICCHP) TO VALORATRIB(QTUTILIZLI) FOR LCHAVEP;
0407 5      QTUTILIZLI = QTUTILIZLI + LCHAVEP;
0414 5      CODATB = #FF;
041A 5      REWRITE     DECJLI  FROM LISTAINVERTE;
0421 5      RETURN;
0428 5  END;
0428 4  REPETE2 :
0428 2      IF LKEXPANLI = 0 THEN        % FIM DE LISTA REQUER EXPANDIR
0431 3  DO;
0431 4      IF ALTER = 'E' THEN          % NAO HA' O QUE EXCLUIR
043A 5    DO;
043A 6      CODATB = #FF;
0440 6      RETURN;
0447 6    END;
0447 5      CALL PEDELI(I);
0450 4      IF I= 0 THEN
0459 5    DO;
0459 6      CODATB = 'A';                % NAO HA' REG LI PARA EXPANDIR
045F 6      RETURN;
0466 6    END;
0466 5      LKEXPANLI = I;
046D 4      REWRITE DECJLI FROM LISTAINVERTE;
0474 4      RLI = I;
047B 4      SEEK DECJLI AT I;
0482 4      READ DECJLI INTO LISTAINVERTE  EOF FALTALI;
048C 4      CHAVEIDLI(0) = ' ';
0496 4      MOVE CHAVEIDLI(0) TO CHAVEIDLI(1) FOR (COMPLI - 4);
04AB 4      LKEXPANLI.QTUTILIZLI = 0;
04B6 4      GO TO INSERE2;           % INSERE REG EXPANSAO PARA VALOR JA'EXISTENTE
04B9 4  END  REQUER-EXPANDIR;
```

```
04B9 3  ELSE                                % COMPRIMENTO CHAVE CORRETO
04BC 3      RLI = LKEXPANLI;
04C3 3      SEEK DECJLI AT RLI;
04CA 2      READ DECJLI INTO LISTAINVERTE EOF FALTALI;
04D4 2  %-----------------VERIFICA SE JA' EXISTE TAL CHAVE NA LISTA;
04D4 2      K = 1;
04DA 2      I = 0;
04E0 2  DO WHILE I< QTUTILIZLI AND K=1;% SE K=0 CHAVE EXISTE
04F2 3      K=0;
04F8 3    DO J=0 TO (LCHAVEP - 1);
0507 4        IF CHAVEIDLI(I + J) <> REGMEST(INICCHP + J)  THEN  K=1;
0528 5    END;
0529 3      I = I + LCHAVEP;
0534 3  END DO-WHILE;
0537 2      IF K=0  THEN
0540 3  DO;                                % CHAVE JA' EXISTE
0540 4      IF  ALTER = 'E' THEN
0549 5    DO;
0549 6      MOVE CHAVEIDLI(QTUTILIZLI - LCHAVEP) TO CHAVEIDLI(I-LCHAVEP)
0562 6                                                          FOR LCHAVEP;
0566 6      QTUTILIZLI = QTUTILIZLI - LCHAVEP;
0573 6    END;
0573 5      CODATB = #FF;
0579 4      RETURN;
0580 4  END;
0580 3  %-------------VERIFICA SE HA' ESPACO PARA INSERCAO
0580 3  INSERE2 :
0580 2      IF (QTUTILIZLI + LCHAVEP) < (COMPLI-3)  THEN
0592 3  DO;
0592 4      MOVE REGMEST(INICCHP) TO CHAVEIDLI(QTUTILIZLI) FOR LCHAVEP;
05A6 4      QTUTILIZLI = QTUTILIZLI + LCHAVEP;
05B3 4      CODATB = #FF;
05B9 4      REWRITE DECJLI FROM LISTAINVERTE;
05C0 4      RETURN;
05C7 4  END;                                % INSERCAO COMPLETADA
```

```
05C7 3  ELSE
05CA 3      GO TO REPETE2;
05CD 3  %----------------------ALTERNATIVAS PARA CONSULTA
05CD 3  CONSULTNEGA :
05CD 2      CODATB = #FF;                %NAO HA' LISTA PARA TAL VALOR ATRIBUTO
05D3 2      RETURN;
05DA 2  CONSULTAFIR :                    %ENCONTRADO PRIMEIRO LI DE TAL VALOR
05DA 2      CODATB = #FF;
05E0 2      RETURN;
05E7 2  LISTVAZIA :
05E7 2      IF ALTER = 'C' OR ALTER = 'E' THEN GO TO CONSULTNEGA;
05FA 3      CALL PEDELI(I);
0603 2      IF I=0 THEN
060C 3  DO;
060C 4      CODATB = 'A';
0612 4      RETURN;
0619 4  END;
0619 3      INDICEII(CODATB) = I;
0624 2      INDICEII(CODATB + 128) = HASHII;
0632 2      REWRITE DECJII FROM INDICEII;
0639 2  INSERE1 :
0639 2      SEEK DECJLI AT I;
0640 2      CHAVEIDLI(0) = ' ';
064A 2      MOVE CHAVEIDLI(0) TO CHAVEIDLI(1) FOR (COMPLI - 4);
065F 2      MOVE REGMEST(INICAT ) TO VALORATRIB(0) FOR COMPAT;
0671 2      COMPATRIB.QTUTILIZLI = COMPAT;
067E 2      LKEXPANLI.PROXLISIN = 0;
0689 2      MOVE VETRTB(5) TO INICCHP FOR 2;
0695 2      LCHAVEP = VETRTB(8);
069F 2      MOVE REGMEST(INICCHP) TO VALORATRIB(QTUTILIZLI) FOR LCHAVEP;
06B3 2      QTUTILIZLI = QTUTILIZLI + LCHAVEP;
06C0 2      COMPCHAV = LCHAVEP;
06C8 2      WRITE DECJLI FROM LISTAINVERTE;
06CF 2      RLI = I;
06D6 2      CODATB = #FF;
```

```
06DC 2       RETURN;
06E3 2  FALTALI :
06E3 2       MOVE ' ERRO-DECJLI MENOR QUE INDICADO POR ECRATB' TO ARA133;
0716 2       WRITE  IMPRES FROM ARA133;
071D 2       CODATB = 'A';
0723 2       RLI = I;
072A 2  RETORNOIS :
072A 2       CODATB = #FA;
0730 2       RLI = 'B';
0736 2  RETORNOISLI :
0736 2  END   PROCEDURE-ECISLI;
073D 1  %_
```

```
000A 1  [VETRTB (1) BYTE, ARA133(1) BYTE, (BMT, BSRESU)ADDRESS, POSFIX(40) ADDRESS,
000A 1                BLI ADDRESS; ]
000A 1  ECPEDI : PROCEDURE;
000F 2  DECLARE
000F. 2      CSINFIX          BYTE INITIAL(0),     % CURSOR AREA IN-FIXADA
0010. 2      CSPOSFIX         BYTE INITIAL(0),     % CURSOR AREA POST-FIXADA
0011. 2      (VIDEO8, IMPRES, TECLAD, DECJRS, DECJLI)    FILE;
0011. 2  ECCGAT : PROCEDURE(ENTX, INICX, COMPX)
0011 2            RESULT    (ENTX, INICX, COMPX);
001C 3  DECLARE
001C. 3      ENTX          BYTE,
001C. 3      (INICX, COMPX, CGAT) ADDRESS;
001E. 3      IF VETRTB(0) < ENTX THEN DO;
002E 5       ENTX = #FA; RETURN;       END;
0039 4       ELSE
003C 4            CGAT = ((ENTX - 1)* 16) + 1;
004C 4            MOVE VETRTB(CGAT + 4) TO INICX FOR 2;
005C 3            MOVE VETRTB(CGAT + 6) TO COMPX FOR 2;
006C 3  END       PROCEDURE-ECCGAT;
0074 2  ECANLS : PROCEDURE;
0079 3  %AUTOR- EDUARDO CARNEIRO CAMPELO JR. ***********************************
0079 3  %        SUBROTINA QUE ANALISA EXPRESSAO LOGICA TRANSFORMANDO-A PARA*
0079 3  %        FORMA POST-FIXADA, SUBSTITUINDO ATRIBUTO POR ENDERECO  DE  *
0079 3  %        LISTA INVERTIDA E OPERADORES POR SEUS CODIGOS BYTE.        *
0079 3  %                                                                  *
0079 3  %******************************************************************
0079 3  DECLARE
0079. 3      AREA80  (80)     BYTE,
00CA. 3      PILHA(20)        ADDRESS INITIAL(42*' '),% PILHA DE OPERADORES E'('
00F4. 3      TOPILHA          ADDRESS,
00F6. 3      REGMEST  BASED BMT (31) BYTE,
00F6. 3      (REPETE, QUALIFIC, OPERA1)              LABEL,
00F6. 3      (ENTB, ALTER, ERRO)    BYTE,
00F9. 3      (I, K, J)        ADDRESS,
00FF. 3      RLI              ADDRESS,     %REG.  ONDE DEVE INICIAR A PROCURA
```

```
0101 3       CODATB              ADDRESS,     %N. ORDEM DO ATRIB.  NA TABELA DFMT
0103 3       RII                 ADDRESS,     %REG. II DO ATRIB. (TAB. IND. LI)
0105 3       INICAT              ADDRESS,     %INICIO DO ATRIBUTO
0107 3       COMPAT              ADDRESS,     %COMPRIMENTO DO ATRIBUTO
0109 3       HASHII              ADDRESS,     %PARA COMPARAR E INSERIR EM REG II
010B 3       KEY                 BYTE    INITIAL ('1');
010C 3   %
010C 3   %-----------------------DECLARACOES  DE  ROTINAS
010C 3   %
010C 3   ECISLI  : PROCEDURE(RLI,CODATB,RII,ALTER) PLTI
010C 3             RESULT    (RLI,CODATB); END;
010C 3   %-----------------------------
010C 3   TRANSHX : PROCEDURE (SAI)              RESULT (SAI);
0113 4   DECLARE
0113 4       SAI      ADDRESS;
0113 4       SAI = 0;
0119 4       CSINFIX = CSINFIX + 1;
0123 4       SAI = AREA80(CSINFIX) - '0';
0131 4       CSINFIX = CSINFIX + 1;
013B 4       SAI = (SAI*10) + AREA80(CSINFIX) - '0';
0150 4       CSINFIX = CSINFIX +1;
015A 4   END  PROCEDURE-TRANSHX;
0160 3   %-------------------------------------------------------------
0160 3   SEMBRANCO : PROCEDURE;
0165 4   DO  WHILE  AREA80(CSINFIX) = ' ' AND CSINFIX < 80;
0179 5       CSINFIX = CSINFIX + 1;
0183 5   END  DO-WHILE;
0186 4   END  SEMBRANCO;
0187 3   %-------------------------------------------------------------------------
0187 3   %ECCGAT  : PROCEDURE (ENTB,INICAT,COMPATB)  PLTI
0187 3   %              RESULT    (ENTB,INICAT,COMPATB); END;
0187 3   %-------------------------------------------------------------
0187 3   DUPLOWRITE : PROCEDURE;
018C 4       WRITE VIDEO8 FROM AREA80;
```

```
0193  4        MOVE AREA80 TO ARA133(1) FOR 80;
019F  4        WRITE IMPRES FROM ARA133;
01A6  4    END  DUPLOWRITE;
01A7  3    LIMPA80 : PROCEDURE;
01AC  4        AREA80(0) = ' ';
01B5  4        MOVE AREA80(0) TO AREA80(1) FOR 79;
01C4  4    END  LIMPA80;
01C5  3    %=============================================================================
```

```
01C5 3  QUALIFICADOR : PROCEDURE(ERROQ)   RESULT(ERROQ);
01CC 4  DECLARE
01CC 4      ERROQ        BYTE,
01CC 4      (REPETE)                          LABEL;
01CC 4      ERROQ = 0;
01D2 4  REPETE :
01D2 4      CALL SEMBRANCO;
01D8 4      IF  AREA80(CSINFIX) = '(' THEN
01E5 5  DO;
01E5 6      TOPILHA = TOPILHA + 1;
01EF 6      IF TOPILHA > 20 THEN DO; ERROQ = 6; END;
01FE 7      PILHA(TOPILHA) = -6;
0209 6      CSINFIX = CSINFIX + 1;
0213 6      CALL  SEMBRANCO;
0219 6      IF  CSINFIX < 80 THEN GO TO  REPETE;
0225 7    ELSE   DO; ERROQ = 1; RETURN; END;
0231 7  END;
0231 5      IF  AREA80(CSINFIX) <> 'A' THEN DO; ERROQ=2; RETURN; END;
024A 5      CALL TRANSHX(CODATB);
0253 4      IF  CODATB > VETRTB(0) OR CODATB < 2 THEN DO; ERROQ=3; RETURN; END;
0273 5      CALL  SEMBRANCO;
0279 4      IF  CSINFIX > 79 THEN DO; ERROQ=1; RETURN; END;
028E 5      ENTB = CODATB;
0295 4      CALL ECCGAT(ENTB,INICAT,COMPAT);
```

```
02A4 4       IF  ENTB = #FA  THEN DO; ERROQ = 3; RETURN; END;
02B9 5       IF  AREA80(CSINFIX) <> '=' THEN DO; ERROQ=4; RETURN; END;
02D2 5       CSINFIX = CSINFIX + 1;
02DC 4       CALL SEMBRANCO;
02E2 4       IF  CSINFIX > 79 THEN DO; ERROQ=1; RETURN; END;
02F7 5       MOVE AREA80(CSINFIX) TO REGMEST(INICAT) FOR COMPAT;
0309 4       CSINFIX = CSINFIX + COMPAT;
0314 4       ALTER = 'C';
031A 4       CALL ECISLI(RLI,CODATB,RII,ALTER);
032C 4       IF  CODATB = #FF THEN
0335 5  DO;
0335 6       POSFIX(CSPOSFIX) = RLI;
0340 6       CSPOSFIX = CSPOSFIX + 1;
034A 6       RETURN;
0350 6  END;
0350 5    ELSE
0353 5         IF CODATB = #FA  AND  RLI = 'K'  THEN
0363 6    DO;
0363 7         CALL  LIMPA80;
0369 7         AREA80(CSINFIX) = '*' ;  ERROQ = 2;              % ????????
0379 7         CALL DUPLOWRITE; RETURN;
0385 7    END;
0385 6         ELSE     ERROQ = 2; RETURN;
0394 4  END  PROCEDURE-QUALIFICADOR;
0394 3  %=============================================================
```

```
0394 3  OPERADOR : PROCEDURE(ERROP)   RESULT(ERROP);
039B 4  DECLARE
039B 4      ERROP        BYTE,
039B 4      OPERA        ADDRESS,
039D 4      (REPETE,REPETE1,REPETE2)                          LABEL;
039D 4      ERROP = 0;
03A3 4  REPETE1 :
03A3 4      CALL SEMBRANCO;
03A9 4      IF  CSINFIX > 79 THEN DO; ERROP=1; RETURN; END;
03BE 5      IF  AREA80(CSINFIX) = ')' THEN
03CB 5  DO;
03CB 6      IF  TOPILHA < 1 THEN DO; ERROP = 7; RETURN;%ESPERADO OPERRANDO OU
03E0 8                           END;                        %EXCESSO DE ')'
03E0 7      IF  PILHA(TOPILHA) =(-6)THEN DO; ERROP=7; RETURN; END;
03FA 7  REPETE2 :
03FA 6      POSFIX(CSPOSFIX) = PILHA(TOPILHA);
0409 6      CSPOSFIX = CSPOSFIX + 1;
0413 6      IF  CSPOSFIX > 40 THEN DO; ERROP=8; RETURN; END;% EXCEDEU CAPAC. POSFX
0428 7      TOPILHA = TOPILHA - 1;
0432 6      IF  TOPILHA < 0 THEN DO; ERROP=8; RETURN; END;% > INDEVIDO
0447 7      IF  PILHA(TOPILHA) <>(-6)THEN GO TO REPETE2;
0458 7      TOPILHA = TOPILHA - 1;
0462 6      CSINFIX = CSINFIX + 1;
046C 6      GO TO REPETE1;
046F 6  END;
046F 5      IF  AREA80(CSINFIX) = '&' THEN OPERA = -1;
0483 5      ELSE
0486 5          IF AREA80(CSINFIX) = '!' THEN OPERA = -2;
049A 6          ELSE
049D 6              IF AREA80(CSINFIX) ='N' THEN OPERA = -3;
04B1 7              ELSE
04B4 7                  IF AREA80(CSINFIX) = '?' THEN
04C1 8                  DO;
04C1 9                    IF TOPILHA < 0 THEN
04CA A                  DO;
```

```
04CA B                    POSFIX(CSPOSFIX) = #FFF0; ERROP=#FF; %ANALISE CONCLUIDA
04DB B                    RETURN;                   % COM SUCESSO
04E1 B                  END;
04E1 A                     IF PILHA(TOPILHA) =(-6)OR TOPILHA > 0 THEN DO; ERROP=
04F9 B                     9; RETURN; END; % ESPERADO ')'
0502 A                     POSFIX(CSPOSFIX) = PILHA(TOPILHA);
0511 9                     CSPOSFIX = CSPOSFIX + 1;
051B 9                     IF CSPOSFIX > 40 THEN DO; ERROP = 8; RETURN; END;
0530 A                     TOPILHA = TOPILHA - 1;
053A 9                     POSFIX(CSPOSFIX) = #FFF0; ERROP = #FF;% ANALISE CONCLUIDA
054B 9                     RETURN;                                % COM SUCESSO
0551 9                  END;
0551 8                     ELSE
0554 8                         DO; ERROP=7; RETURN; END;
0560 8                  IF TOPILHA >(-1)AND PILHA(TOPILHA) <>(-6)   THEN
0576 5                  DO;
0576 6                     POSFIX(CSPOSFIX) = PILHA(TOPILHA);
0585 6                     PILHA(TOPILHA) = OPERA;
0590 6                     CSINFIX = CSINFIX + 1;
059A 6                     CSPOSFIX = CSPOSFIX + 1;
05A4 6                     IF CSPOSFIX > 40 THEN DO; ERROP = 8; RETURN; END;
05B9 7                     RETURN;
05BF 6                  END;
05BF 5               ELSE
05C2 5                  CSINFIX = CSINFIX + 1;
05CC 5                  TOPILHA = TOPILHA + 1;
05D6 4                  PILHA(TOPILHA) = OPERA;
05E1 4                  RETURN;
05E7 4  END    PROCEDURE-OPERADOR;


05E7 3  %-------------------------------INICIO  DO  PROCESSAMENTO
05E7 3  REPETE:
05E7 3      CALL LIMPA80;
05ED 3      MOVE  'TECLE SEU PEDIDO OU DESCONTINUI(D)' TO AREA80;
```

```
0618 3       CALL  DUPLOWRITE;
061E 3       CSINFIX,CSPOSFIX = 0;
0628 3       TOPILHA = -1;
062F 3       READ TECLAD INTO AREA80;
0636 3       CALL DUPLOWRITE;
063C 3       CALL SEMBRANCO;
0642 3       IF CSINFIX > 79 THEN GO TO REPETE;
064E 4       IF AREA80(CSINFIX) = 'D' THEN EXIT;
065C 4    %-----------------------------ANALISA EXPRESSAO
065C 4       IF (AREA80(CSINFIX) = 'P') AND (AREA80(CSINFIX + 1))= 'A' THEN
0677 4           DO;
0677 5              POSFIX(CSPOSFIX) = -1000; % RESULTANTE PEDIDO ANTERIOR
0683 5              CSINFIX = CSINFIX + 2;
068D 5              CSPOSFIX = CSPOSFIX + 1;
0697 5              GO TO OPERA1;
069A 5           END;
069A 4    QUALIFIC :
069A 3       CALL QUALIFICADOR(ERRO);
06A3 3       IF ERRO <> 0 THEN DO;
06AC 5           CALL LIMPA80;
06B2 5           AREA80(CSINFIX) = '*';
06BC 5           CALL DUPLOWRITE;
06C2 5           MOVE 'ERRO - ESPERADO QUALIFICADOR' TO AREA80;
06E7 5           CALL DUPLOWRITE;  GO TO REPETE;
06F0 5       END;
06F0 4    OPERA1 :
06F0 3       CALL OPERADOR(ERRO);
06F9 3       IF ERRO <> 0  AND  ERRO <> #FF  THEN
0709 4    DO;
0709 5       CALL LIMPA80;
070F 5       AREA80(CSINFIX) = '*';
0719 5       CALL DUPLOWRITE;
071F 5       MOVE 'ERRO - ESPERADO OPERADOR' TO AREA80;
0740 5       CALL DUPLOWRITE;
0746 5       GO TO REPETE;
```

```
0749 5   END;
0749 4        IF ERRO = #FF THEN RETURN;
0753 4        GO TO QUALIFIC;
0756 3   %
0756 3   %
0756 3   END  PROCEDURE-ECANLS;
0757 2   %=====================================================================
```

```
0757 2  %================RESOLUCAO  DAS  EXPRESSOES  BOOLEANAS
0757 2  %
0757 2  DECLARE
0757. 2 1   RESULTANTE      BASED     BSRESU,
0757. 2       2  CONTCHAV            ADDRESS,
0757. 2       2  CHAV      (509)     BYTE,
                                      ^
        157 - MAIS DE 25 ARQUIVOS DEFINIDOS : IGNORA OS ULTIMOS                    <---

0757. 2 1   RESULTAUX,
0757. 2       2   CONTCHAVAUX       ADDRESS,
0759. 2       2   CHAVAUX (509)     BYTE,
                                      ^
        157 - MAIS DE 25 ARQUIVOS DEFINIDOS : IGNORA OS ULTIMOS                    <---

0957. 2       BSRESULTAUX           ADDRESS INITIAL(@RESULTAUX),
0959. 2 1   LISTAINVERTE     BASED  BLI,
0959. 2       2   TAMPAOLI     (2) BYTE,
0959. 2       2   PROXLISIN        ADDRESS,  %APONTA PROX LI DE VALORES SINONIMOS
0959. 2       2   COMPATRIB        BYTE,     %COMPRIMENTO DO VALOR DO ATRIBUTO
0959. 2       2   COMPCHAV         BYTE,     %COMPRIMENTO DA CHAVE PRIMARIA
0959. 2       2   VALORATRIB  (56)BYTE,     %VALOR DO ATRIBUTO
0959. 2  %--------------REGISTRO DE EXPANSAO DE LISTA INVERTIDA
0959. 2 1   LISTAINVERTE2   BASED  BLI,
0959. 2       2   LKEXPANLI        ADDRESS,  %APONTA PROX REG EXPANSAO DA LI
0959. 2       2   QTUTILIZLI       BYTE,     %QUANT POSICOES UTILIZADAS PARA GUAR-
0959. 2                                      %DAR CHAVES DE ENTIDAS
0959. 2       2   CHAVEIDLI    (60)BYTE,     %IDENTIFICADOR ENTIDADES (CHAVE PRIMAR)
0959. 2  %-------------------------------------------------------------------------
0959 2  DECLARE
0959. 2      (NCHAV, OPERALI, LCHAV, POSCHAV, OPERANDOA, OPERANDOB)  ADDRESS,
0965. 2      (HY, HX, NRESOLUCAO)                                    ADDRESS;
0968. 2  %=========================================================================
096B 2  PEDECHAV : PROCEDURE;
0970 3  %   PARA A PRIMEIRA CHAMADA DE UM OPERANDO FORNECA NCHAV=0 E OPERALI=
```

```
0970 3  %   OPERANDO A PROCESSAR; NAS DEMAIS CHAMADAS NAO ALTERAR. - QUANDO FIM
0970 3  %   DE LI RETORNA NCHAV= #FFFF, CSO CONTRARIO DEIXA POSCHAV= POSICAO
0970 3  %   INICIAL CHAVE E LCHAV= COMPRIMENTO.
0970 3  %
0970 3      IF NCHAV = 0 THEN
0979 4      DO; SEEK DECJLI AT OPERALI;
0980 5          READ DECJLI INTO LISTAINVERTE;
0987 5          LCHAV = COMPCHAV;
098F 5          POSCHAV = 4 + COMPATRIB;
099A 5          NCHAV = NCHAV + 1; RETURN;
09A5 5      END;
09A5 4  ELSE DO;
09A8 5          POSCHAV = POSCHAV + LCHAV;
09B3 5          IF QTUTILIZLI <= POSCHAV  AND  LKEXPANLI <> 0 THEN
09C5 6              DO; SEEK DECJLI AT LKEXPANLI;
09CC 7                  READ DECJLI INTO LISTAINVERTE;
09D3 7                  POSCHAV = 0; NCHAV = NCHAV + 1; RETURN;
09E4 7              END;
09E4 6          ELSE IF QTUTILIZLI <= POSCHAV AND LKEXPANLI = 0 THEN
09F9 7              DO; NCHAV = #FFFF; RETURN; END;
0A01 7          NCHAV = NCHAV + 1;
0A0B 5          RETURN;
0A0C 5        END;
0A0C 4  END  PROCEDURE-PEDECHAV;
0A0D 2  %-------------------------------------------------------------------
0A0D 2  %
0A0D 2  HASHCHAV : PROCEDURE(KHASH,BHASH) RESULT(KHASH,BHASH);
0A16 3  %   SE RESULTAR BHASH= #FFFF E' ERRO- CHAVE C/ BRANCO
0A16 3  DECLARE
0A16 3      (KHASH,BHASH)       ADDRESS,
0A16.3      CHAVBASE    BASED   BHASH   (20) BYTE,  %CHAVE A SER TRANSFORMADA
0A16.3      IH                  ADDRESS;
0A18.3      IH = 0;
0A1E 3      KHASH = CHAVBASE(0);
0A28 3    DO  WHILE  IH < LCHAV  AND  CHAVBASE(IH) <> ' ';
```

```
0A3D 4       KHASH = (KHASH <- 2) XOR CHAVBASE(IH);
0A4F 4       IH = IH + 1;
0A59 4     END  DO-WHILE;
0A5C 3       IF IH < LCHAV THEN
0A66 4           DO;
0A66 5               MOVE 'ERRO HASHCHAV = CHAVE COM BRANCO' TO ARA133;
0A8F 5               WRITE IMPRES FROM ARA133;
0A96 5               WRITE VIDEO8 FROM ARA133;
0A9D 5               BHASH = #FFFF;     RETURN;
0AAB 5           END;
0AAB 4       IF KHASH < 0 THEN  KHASH = KHASH XOR #8000;
0ABF 4       KHASH =((KHASH MOD 510) / LCHAV) * LCHAV; %MAPEA EM RESULTANTE
0AD2 3       RETURN;
0AD9 3   END  PROCEDURE-HASHCHAV;
0AD9 2   %=============================================================================
```

```
0AD9 2  INSERESULTAUX : PROCEDURE (INS1, INS2, INS3);
0AE4 3  DECLARE
0AE4 3       (INS1, INS2, INS3, INSX, INSY, HY)        ADDRESS;
0AEA 3  %
0AEA 3       INSX = INS1;
0AF1 3       DO WHILE CHAVAUX(INSX) <> ' '  AND  INSX <> INS3;
0B06 4           INSX = INSX + LCHAV;
0B11 4           IF INSX > (510 - LCHAV) THEN INSX = 0;
0B25 5       END  DO-WHILE;
0B28 3       IF  INSX <> INS3 THEN
0B32 4       DO  INSY = 0 TO (LCHAV - 1);
0B41 5           CHAVAUX(INSX + INSY) = CHAV(INS2 + INSY);
0B59 5           IF INS3 < 510 THEN
0B63 6               CHAVAUX(INS3 + INSY) = ' ';
0B71 6       END   DO-TO;
0B72 4  END   PROCEDURE-INSERESULTAUX;
0B73 2  %
0B73 2  LIMPARESULTAUX : PROCEDURE;
0B78 3       DO  HY=0  TO (LCHAV - 1);
0B87 4           CHAVAUX(HY + HX) = ' ';
0B95 4       END  DO-TO;
0B96 3  END   PROCEDURE-LIMPARESULTAUX;
0B97 2  %==============================RESOLUCAO  DA  FUNCAO  AND
0B97 2  ROTAND : PROCEDURE;
0B9C 3  DECLARE
0B9C 3       (ANDA2, ANDA3, ANDB, ANDB2, ANDB3, ANDB4, ANDB5, RETORNOAND)   LABEL,
0B9C 3       (KHASH, BHASH, HX, AD, AN, HY, KHASH1, FLAGB)                  ADDRESS;
0BAC 3       CONTCHAV = 0;  CHAV(0) = ' ';
0BBC 3       MOVE CHAV(0) TO CHAV(1) FOR 509;
0BCE 3       IF OPERANDOA = 0 OR OPERANDOB = 0 THEN RETURN;
0BDF 4       IF OPERANDOA  <  0  THEN
0BE8 4           DO;
0BE8 5             SEEK  DECJRS  AT  (-OPERANDOA - 1000);
0BF4 5             READ  DECJRS  INTO  RESULTANTE;
0BFB 5             GO TO ANDB;
```

```
0BFE 5           END;
0BFE 4       ELSE
0C01 4         DO;
0C01 5          NCHAV = 0; OPERALI = OPERANDOA;
0C0E 5  ANDA2 :
0C0E 5           CALL PEDECHAV;
0C14 5           IF  NCHAV = #FFFF THEN GO TO ANDB;   % NAO HA' MAIS CHAVE
0C21 6           BHASH = BLI + 3 + POSCHAV;
0C2F 5           CALL HASHCHAV(KHASH,BHASH);
0C3B 5           IF BHASH = #FFFF THEN
0C45 6             DO;
0C45 7               MOVE 'ERRO ROTAND- CHAVE COM BRANCO' TO ARA133;
0C6B 7               WRITE VIDEO8 FROM ARA133;
0C72 7               WRITE IMPRES FROM ARA133; GO TO ANDA2;
0C7C 7             END;
0C7C 6           HX = 0;
0C82 5  ANDA3 :                % INSERE CHAVE EM RESULTANTE
0C82 5         DO WHILE CHAV(KHASH) <> ' ' AND KHASH < (511 -LCHAV);
0C9C 6           KHASH = KHASH + LCHAV;
0CA7 6         END  DO-WHILE;
0CAA 5           IF KHASH >= (511 -LCHAV) THEN
0CB8 6             DO;  IF HX = 1 THEN
0CC1 8                  DO;
0CC1 9                    MOVE 'AREA RESULTANTE INSUFICIENTE - OPERANDO A FUNC'
0CC1 9                         'AO AND'  TO  ARA133;
0CFE 9                    WRITE VIDEO8 FROM ARA133;
0D05 9                    WRITE IMPRES FROM ARA133;  GO TO ANDB;
0D0F 9                  END;
0D0F 8                HX = 1; KHASH = 0; GO TO ANDA3;
0D1E 7              END;
0D1E 6            ELSE
0D21 6                HX = 0;
0D27 6              DO  WHILE  HX < LCHAV;
0D31 6                CHAV(KHASH + HX) = CHAVEIDLI(POSCHAV + HX);
0D4A 6                HX = HX + 1;
```

```
0D54 6              END  DO-WHILE;
0D57 5                CONTCHAV = CONTCHAV + 1;  GO TO ANDA2;
0D64 5          END   ELSE;
0D64 4  ANDB   :
0D64 3       IF OPERANDOB < 0 THEN
0D6D 4         DO;
0D6D 5           SEEK  DECJRS AT (-OPERANDOB - 1000);
0D79 5           READ  DECJRS INTO RESULTAUX;
0D80 5           HX,AN,NCHAV =0; AD = CONTCHAVAUX; CONTCHAVAUX = 0;
0D9B 5  ANDB2 :                % -----LOCALIZA CHAVE EM RESULTAUX
0D9B 5         DO WHILE CHAVAUX(HX) = ' '  AND  AN < AD  AND  HX < (511-LCHAV);
0DBC 6           HX = HX + LCHAV;
0DC7 6         END  DO-WHILE;
0DCA 5           IF  AN >= AD OR HX >= (511 - LCHAV) THEN GO TO RETORNOAND;
0DE3 6           AN = AN + 1;
0DED 5           BHASH = BSRESULTAUX + HX +2;
0DFB 5           CALL HASHCHAV(KHASH,BHASH);
0E07 5           IF BHASH = #FFFF THEN
0E11 6             DO;
0E11 7               MOVE ' ERRO ROTAND- CHAVE COM BRANCO EM RESULTAUX' TO ARA133;
0E41 7               WRITE VIDEO8 FROM ARA133;
0E4C 7               WRITE IMPRES FROM ARA133;
0E53 7               CHAVAUX(HX) = ' ';
0E5D 7               MOVE CHAVAUX(HX) TO CHAVAUX (HX + 1) FOR (LCHAV - 1);
0E75 7               HX = HX + LCHAV;  GO TO ANDB2;
0E83 7             END  DO-BHASH=#FFFF;
0E83 6               KHASH1 = KHASH;   FLAGB = 0;
0E90 5  ANDB3        HY = 0;
0E96 5               DO WHILE HY < LCHAV  AND  CHAV(KHASH1 + HY) <> ' ';
0EB0 6               IF CHAV(KHASH1 + HY) <> CHAVAUX(HX +HY) THEN HY = 1000;
0ED2 7               HY = HY +1;
0EDC 6               END     DO-WHILE;
0EDF 5               IF HY >= 1000 THEN
0EE9 6                 DO;
0EE9 7                   KHASH1 = KHASH1 + LCHAV;
```

```
0EF4 7                    IF KHASH1 > (510 - LCHAV) THEN
0F02 8                      DO; FLAGB = 1; KHASH1 =0; GO TO ANDB3; END;
0F11 8                    IF KHASH1 >= KHASH  AND  FLAGB = 1 THEN
0F22 8                      DO;
0F22 9                        CALL LIMPARESULTAUX;
0F28 9                        HX = HX + LCHAV; GO TO ANDB2;
0F36 9                      END;
0F36 8                    GO TO ANDB3;
0F39 7                  END  DO-HY>=1000;
0F39 6                IF HY < LCHAV AND CHAV(KHASH1 + HY) = ' ' THEN
0F53 6                  DO;    CALL LIMPARESULTAUX;
0F59 7                         HX = HX + LCHAV; GO TO ANDB2;
0F67 7                  END;
0F67 6  %      CHAVES IGUAIS - GUARDA EM RESULTAUX INCREMENTANDO CONTCHAVAUX
0F67 6                CALL INSERESULTAUX(KHASH,KHASH1,HX);
0F76 5                CONTCHAVAUX = CONTCHAVAUX + 1; HX = HX + LCHAV; GO TO ANDB2;
0F8E 5            END    DO-OPERANDOB<0;
0F8E 4  %-----------------OPERANDOB E' LISTA INVERTIDA
0F8E 4                NCHAV,CONTCHAVAUX = 0; OPERALI = OPERANDOB; CHAVAUX(0)= ' ';
0FA8 3                MOVE CHAVAUX(0) TO CHAVAUX(1) FOR 509;
0FB8 3                IF OPERANDOB = 0 THEN GO TO RETORNOAND;
0FC4 4  ANDB4 :
0FC4 3                CALL PEDECHAV;
0FCA 3                IF NCHAV = #FFFF THEN GO TO RETORNOAND; % FIM DE LISTA
0FD7 4                BHASH = BLI + 3 + POSCHAV;
0FE5 3                CALL  HASHCHAV(KHASH,BHASH);
0FF1 3                IF BHASH = #FFFF THEN
0FFB 4                  DO;
0FFB 5                    MOVE ' ERRO ROTAND OPERANDO B - CHAVE COM BRANCO' TO
1027 5            ARA133; WRITE  VIDEO8  FROM  ARA133;
1035 5                    WRITE  IMPRES  FROM  ARA133;
103C 5                  END;
103C 4                KHASH1 = KHASH; FLAGB = 0;
1049 3  ANDB5 :
1049 3                HY = 0;
```

```
104F 3              DO WHILE HY < LCHAV  AND  CHAV(KHASH1 + HY) <> ' ';
1069 4                IF CHAV(KHASH1 + HY) <> CHAVEIDLI(POSCHAV + HY) THEN
1085 5                                                    HY = 1000;
108C 5                HY = HY + 1;
1096 4              END  DO-WHILE;
1099 3                IF HY >= 1000   THEN
10A3 4                  DO;
10A3 5                    KHASH1 = KHASH1 + LCHAV;
10AE 5                    IF KHASH1 > (510 - LCHAV) THEN
10BC 6                    DO; FLAGB = 1; KHASH1 = 0; GO TO ANDB5; END;
10CB 6                IF KHASH1 >= KHASH  AND  FLAGB = 1 THEN
10DC 6                                   GO TO ANDB4;  % CHAVE NAO SATISFAZ AND
10DF 6                GO TO ANDB5;
10E2 5              END  DO-HY=1000;
10E2 4            IF HY <> LCHAV   THEN
10EC 4                                   GO TO ANDB4;  % CHAVE NAO SATISFAZ AND
10EF 4  %-----------CHAVES IGUAIS -GUARDA EM RESULTAUX INCREMENTA CONTCHAVAUX
10EF 4                FLAGB = 20000;
10F6 3                CALL INSERESULTAUX(KHASH,KHASH1,FLAGB); CONTCHAVAUX =
1108 3                CONTCHAVAUX + 1;       GO TO ANDB4;
1112 3  RETORNOAND :     MOVE RESULTAUX TO RESULTANTE FOR 512;
111C 3                  RETURN;
111D 3  END      PROCEDURE-ROTAND;
111D 2  %=============================================================================
```

```
1110 2  ROTNOT  : PROCEDURE;
1122 3          RETURN;
1123 3  END     PROCEDURE-ROTNOT;
1123 2  ROTOR   : PROCEDURE;
1128 3          RETURN;
1129 3  END     PROCEDURE-ROTNOT;
```

```
1129 2  RESOLVE$EXPR : PROCEDURE(ERRORE) RESULT(ERRORE);
1130 3  DECLARE
1130.3      ERRORE       BYTE,
1130.3      (LERAD, I, TR)              ADDRESS,
1136.3      PILHA  (20) ADDRESS,     % TR SEU TOPO
1160.3      (REPETER, FIMRESOLVE, DECJRSPEQUENO)          LABEL;
1160.3      NRESOLUCAO = -1000;
1168 3      ERRORE, CSPOSFIX, TR =0;
1176 3  REPETER :
1176 3      LERAD = (POSFIX(CSPOSFIX) + 3);
1184 3      IF LERAD > 2  OR POSFIX(CSPOSFIX) = (-1000) THEN
19A 4     DO;
19A 5   %-----------TRATA-SE DE ATRIBUTO
19A 5           TR = TR + 1;
1A4 5           PILHA(TR) = POSFIX(CSPOSFIX);
1B3 5           CSPOSFIX = CSPOSFIX + 1;  GO TO REPETER;
1C0 5       END;
1C0 4       IF POSFIX(CSPOSFIX) = #FFF0 THEN GO TO FIMRESOLVE;
1D1 4       IF TR < 1 THEN DO; ERRORE = 11; RETURN; END;
1E6 4       OPERANDOB = PILHA(TR);
1F1 3       TR = TR - 1;
1FB 3       OPERANDOA = PILHA(TR); NRESOLUCAO = NRESOLUCAO - 1;
210 3       PILHA(TR) = NRESOLUCAO;
21B 3       IF OPERANDOA > 0 AND OPERANDOB < 0 THEN
22B 4         DO;
```

```
122B 5           I = OPERANDOA;  OPERANDOA = OPERANDOB; OPERANDOB = I;
1240 5         END;
1240 4     DO  CASE  LERAD;
1246 4           CALL ROTNOT;
124F 4           CALL ROTOR;
1258 4           CALL ROTAND;
1261 4     END     DO-CASE;
1266 3   %-------------------GUARDA EM DECJRS RESULTADO DA RESOLUCAO
1266 3       SEEK  DECJRS  AT (-NRESOLUCAO - 1000);
1272 3       WRITE DECJRS  FROM RESULTANTE EOF DECJRSPEQUENO;
127C 3       CSPOSFIX = CSPOSFIX + 1;
1286 3       GO TO REPETER;
1289 3   DECJRSPEQUENO :
1289 3       MOVE 'ARQ. DECJRS PEQUENO PARA N.RESOLUCOES-RECRIE'  TO ARA133(1);
12C1 3       WRITE  VIDEO8  FROM  ARA133;
12C8 3       WRITE  IMPRES  FROM  ARA133;
12CF 3       EXIT;
12D0 3   FIMRESOLVE :
12D0 3       SEEK  DECJRS  AT  0;
12D6 3       WRITE DECJRS  FROM  RESULTANTE;
12DD 3       RETURN;
12E3 3   END  RESOLVE$EXPR;
12E3 2   %=============================================================
```

```
12E3 2  %=======================PROCESSAMENTO  DE  ECPEDI
12E3 2  DECLARE ERRO11  BYTE, REP   LABEL;
12E4 2      ERRO11 = 01; CALL ECCGAT(ERRO11,POSCHAV,LCHAV);
12F9 2      IF ERRO11 = #FA THEN
1302 3  DO; MOVE ' ERRO EM PROC. ECPEDI - CHAMADA ECCGAT' TO ARA133;
1330 4      WRITE  IMPRES FROM ARA133; EXIT;
1338 4  END;
1338 3  REP : CALL ECANLS; CALL RESOLVE$EXPR(ERRO11);
1347 2      IF ERRO11 = 11 THEN DO; MOVE ' ERRO11 -RESOLVE$EXPR- INESPERADO '
1350 4         'FIM DE EXPRESSAO' TO ARA133 FOR 50;
138A 4      WRITE VIDEO8 FROM ARA133; WRITE IMPRES FROM ARA133; GO TO REP; END;
139B 3      RETURN;
139C 2  END    PROCEDURE-ECPEDI;
```

```
000A 1  [ (BMT,BII,B133) ADDRESS; ]
000A 1  ECRCDD : PROCEDURE(ENDRETORNO,BCH)  RESULT(ENDRETORNO);
0013 2  %           ROTINA PARA RECUPERACAO DE REGISTRO MESTRE QUANDO CHAVE
0013 2  %           E' FORNECIDA EM PASSACHAVE
0013 2  DECLARE
0013.2    CHAR  LITERALLY  'BYTE';
0013.2  %----------ESTRUTURA DO ARQUIVO MESTRE /32 BYTES/
0013 2  %
0013 2  DECLARE
0013.2  1  HEADERM  BASED BMT,
0013.2     2 NOMEEXTM    (6)  BYTE,
0013.2     2 DESCRIM     (9)  BYTE,
0013.2     2 DIMENSAOM        ADDRESS,          % COM HEADER
0013.2     2 TTALREGDADO      ADDRESS,          % REG DADOS
0013.2     2 NGERAMEST        ADDRESS,
0013.2     2 NGERATABM        ADDRESS,
0013.2     2 MODCOMPRIMTAB    ADDRESS,
0013.2     2 BLOCATABM        ADDRESS,
0013.2     2 VAZIOHM     (3)  BYTE,
0013.2  %----------REGISTRO DE DADOS
0013.2  1  DADOSM  BASED  BMT,
0013.2     2 CHAVEDADO  (4)  BYTE,
0013.2     2 ATRIB      (11)  ADDRESS,
0013.2     2 FLAGDADO         BYTE,     % I-RECEM INCLUIDO/A-ALTERADO/V-VELHO
0013.2                                   % X-EXCLUIDO LOGICAMENTE
0013.2     2 ENDINDADO        ADDRESS,
0013.2  %----------REGISTRO DE EXPANSAO DE LISTA DE SINONIMOS
0013.2  1  EXPAN  BASED  BMT,
0013.2     2 CHAVEXPAN  (6)  ADDRESS,
0013.2     2 PONTEXPAN  (6) ADDRESS,
0013.2     2 QUANTEXPAN       BYTE,          % ZERO APONTA O PRIMEIRO
0013.2     2 FLAGEXPAN        BYTE,          % CODIGO 'E'
0013.2     2 ENDCONTINUA      ADDRESS;       % ZERO NAO TEM CONTINUACAO
```

```
0013.2  %------------------------------------------------------------------------
0013 2  %----------ESTRUTURA DO ARQUIVO TABELA DE POMTEIROS DE LISTAS SINONIMOS
0013 2  %       REGISTRO HEADER  NUM.  ZERO
0013 2  DECLARE
0013.2    1  HEADER,
0013.2       3 NOME (6) CHAR ,
001A.2       3 DESCRC (41) CHAR,
0044.2       3 TABGERACAO  ADDRESS ,
0046.2       3 TFLAG    BYTE,            % A-ALTERA    V- ESTADO ANTIGO
0047.2       3 BLOCAG    ADDRESS,         % MAXIMO  255
0049.2       3 COMPRIMENT  ADDRESS,      % QUANTIDADE TOTAL DE PONTEIROS
004B.2       3 QREGISTAB   ADDRESS,      % QUANTIDADE REG.  P/ CONTER TABELA+HEADER
004D.2       3 PONTDISP    ADDRESS,      % INICIO ESPACO DISPONIVEL
004F.2       3 QUANTDISP   ADDRESS ,%QUANTIDADE REGISTROS DISPONIVEIS AINDA
0051.2       3  QUANTEXP   ADDRESS, % QUANT.  REGISTROS UTILIZADOS P/ EXPANSAO
0053.2       3 HEADVAZIO   (433) BYTE,
                                  /\
        157 - MAIS DE 25 ARQUIVOS DEFINIDOS : IGNORA OS ULTIMOS                  <

0205.2       3 DATAUTIL (5) BYTE ,      % (DD/MM/AA) UTILIZACAO ESPACO DISPONIVEL
0208.2       3 DATALTER (5) BYTE,       % (DD/MM/AA) DIMENSAO DA TABELA
0211.2       3 NREGTABH    ADDRESS;
0213.2  %  REGISTROS  DE  PONTEIROS
0213 2    DECLARE
0213.2      BREGPONT      ADDRESS  INITIAL(@HEADER),
0215.2   1    REGPONT   BASED  BREGPONT,
0215.2         3 PONTEIRO   (254) ADDRESS,
                                      /\
        157 - MAIS DE 25 ARQUIVOS DEFINIDOS : IGNORA OS ULTIMOS

0215.2         3 NREGTAB     ADDRESS;
```

```
0215  2  %----------------------VARIAVEIS  DE  USO GERAL
0215  2  DECLARE
0215  2     BCH                      ADDRESS,
0215  2     PASSACHAVE BASED BCH     (4)  BYTE,
0215  2     ENDRETORNO               ADDRESS,
0215  2     I                        ADDRESS,
0217  2     K                        ADDRESS,
0219  2     ERRO                     ADDRESS,
021B  2     ENDLISTA                 ADDRESS,   %END.  PONTEIRO LISTA NO BLOCO
021D  2     ENDULTLIDO               ADDRESS,   %END.  ULTIMO MESTRE LIDO NO ARQ.
021F  2     ENDINCLUSAO              ADDRESS,   %END.  PARA INCLUSAO NOVO MESTRE
0221  2     ENDINCLUSPOEXPAN         ADDRESS,   %END.  INCLUSAO REG.  EXPANSAO
0223  2     PASSACHAVEADD            ADDRESS,   %CHAVE EM HEXADECIMAL
0225  2     PONTDISPAUX              ADDRESS,   %ESPACO DISPONIVEL INICIAL
0227  2     RCDD                     BYTE INITIAL(' '),
0228  2     (BLOCAGEM, COMPRIMENTO) ADDRESS,
022C  2     AREA80  BASED B133 (79) BYTE,
022C  2     (ROTALTERA, ALTERAR, EXCLUIR, REPETEXPAN, REPETEXPAN2, LISTASEMEXPAN)LABEL,
022C  2     (CANCEL, INICPROCES, ERROLISTAEXPAN, REPETEXP, INCLUIR, SALTINSERE)  LABEL;
022C  2  %----------------------DECLARACOES  DE  ARQUIVOS



022C  2  DECLARE
022C  2     (DECJ01, DECJ15, IMPRES, VIDEO8)        FILE;
```

```
022C. 2  %-----ROTINA DE RECUPERACAO DE INFORMACOES DO ARQUIVO MESTRE:
022C  2  %  RECEBE CHAVE PELA VARIAVEL PASSACHAVEADD, RETORNANDO EM ENDRETOR-
022C  2  % NO O ENDERECO DO REGISTRO DESEJADO DEIXANDO TAL REGISTRO EM DADOSM
022C  2  % CASO O MESMO EXISTA; CASO CONTRARIO ENDRETORNO = '*'.
022C  2  %
022C  2  ROTRECUPERAR: PROCEDURE(PONTREGFR,ULTLIDO) RESULT(PONTREGFR,ULTLIDO);
0235  3  DECLARE
0235. 3    ULTLIDO              ADDRESS, %ENDERECO DO ULTIMO MESTRE LIDO
0235. 3    INDEXPAN             ADDRESS, %INDEXADOR DO VETOR CHAVE EXPANSAO
0237. 3    LEREXPAN             ADDRESS, %INDICE PARA COMANDO LEITURA EXPANSAO
0239. 3    ENDPONTFR            ADDRESS, %ENDERECO CALCULADO P/ TABELA
023B. 3    REGTABFR             ADDRESS, %NUMERO DO REGISTRO TABELA A SER LIDO
023D. 3    PONTREGFR            ADDRESS, %POSICAO DO PONTEIRO NO REGISTRO TABELA
023D. 3  %---------- ROTULOS
023D. 3    (FIMRECUPERA,CONSUL40,NAOEXISTE,RECUPEXPAN,CONSUL4,ESTAMEMORIA) LABEL;
023D  3  %----------INICIO DE PROCEDIMENTOS DE RECUPERACAO
023D  3    ENDPONTFR = PASSACHAVEADD  MOD  COMPRIMENTO;
0248  3    REGTABFR = (ENDPONTFR / BLOCAGEM) + 1;
0256  3    PONTREGFR = (ENDPONTFR  MOD  BLOCAGEM);
0261  3    IF REGTABFR  = NREGTAB THEN GO TO ESTAMEMORIA ;% MESMO BLOCO ANTERIOR
026E  4    SEEK DECJ01 AT REGTABFR;
0275  3    READ DECJ01 INTO REGPONT;
027C  3  ESTAMEMORIA :
027C  3    ENDRETORNO = PONTEIRO (PONTREGFR);
0287  3    ULTLIDO = ENDRETORNO;
028E  3  % ENDERECO DE RETORNO DO PRIMEIRO REGISTRO DA LISTA
028E  3    IF ENDRETORNO = 0 THEN GO TO  NAOEXISTE;
029A  4    ELSE
029A  4  CONSUL4 :
029A  4    SEEK DECJ15 AT  ENDRETORNO;
02A1  3  CONSUL40 :
02A1  3    READ DECJ15 INTO DADOSM EOF NAOEXISTE;
02AB  3    IF  FLAGDADO = 'E' THEN  GO TO RECUPEXPAN;
02B8  4      IF ENDPONTFR <> ENDINDADO THEN
02C3  4    DO;                  %DO-2
02C3  5  NAOEXISTE :
02C3  5    ENDRETORNO = '*';
02C9  5    RETURN;
02D0  5    END  FIM-DO-2;    %FIM-DO-2
02D0  4    ELSE
```

```
02D3 4     IF FLAGDADO = 'V' OR FLAGDADO = 'I' OR FLAGDADO = 'A' THEN
02ED 5           DO;                          %DO-3
02ED 6             K=0;
02F3 6           DO I=0 TO  4;                %DO-4
02FE 7             IF  PASSACHAVE (I) <> CHAVEDADO (I) THEN
0310 8             K=1;
0316 8           END  FIM-DO-4;               %FIM-DO-4
0317 6             IF  K=0 THEN RETURN;                          %CHAVES IGUAIS - EXISTE
0327 7           END  FIM-DO-3;               % FIM-DO-3
0327 5     ULTLIDO = ULTLIDO + 1;
0331 3     GO TO CONSUL40;                                       %CONSULTAR O PROXIMO
0334 3   %----------RECUPERACAO EM REG-EXPANSAO DE LISTA
0334 3   RECUPEXPAN : DO;                                       %DO-RECUPEXPAN
0334 4     I=0;
033A 4     K = QUANTEXPAN + 1;
0345 4     DO WHILE PASSACHAVEADD  <>  CHAVEXPAN (I)  AND
0350 4              I < K ;                                     % DO-WHILE-1
035B 5     I=I + 1;
0365 5     END                                                 FIM-DO-WHILE;
0368 4     IF I < K  THEN
0372 5               DO;                                 %DO-6
0372 6               ENDRETORNO = PONTEXPAN (I);
037E 6             GO TO CONSUL4 ;
0381 6               END                                       FIM-DO-6;
0381 5     ELSE
0384 5               IF ENDCONTINUA <> 0  THEN
038E 6           DO;
038E 7               ENDRETORNO = ENDCONTINUA;
0396 7               GO TO CONSUL4;
0399 7           END;
0399 6               ELSE  GO TO NAOEXISTE;
039F 6   END     FIM-RECUPEXPAN;             % FIM-RECUPEXPAN
039F 3   FIMRECUPERA :
039F 3   END;       %----------FIM DA PROCEDURE ROTRECUPERAR----------
03A6 2   %-------------------------------------------------------------------------------
```

```
03A6  2  %----------ROTINA DE VERIFICACAO DOS ARQUIVOS MESTRE E TABELA
03A6  2  %  E' REQUERIDO QUE O CALCULO DOS ENDERECOS NA TABELA E AS LIGACOES
03A6  2  %  ENTRE OS ARQUIVOS ESTEJAM DE ACORDO ; ISTO E' AS GERACOES ESTEJAM
03A6  2  %  COMPATIVEIS. DECJ01 - ARQ. TABELA / DECJ15 - ARQ. MESTRE
03A6  2  %  EM CASO DE ERRO A VARIAVEL ERRO= 1,2,3 - ERRO=0 INDICA CERTO
03A6  2  ROTABERTURA : PROCEDURE;
03AB  3  DECLARE
03AB. 3    NOMEDECJ01 (6) BYTE INITIAL ('DECCJ01'),
03B2. 3    NOMEDECJ15  (5) BYTE INITIAL ('DECCJ15');
03B9. 3  DECLARE
03B9. 3    (SAIABERTURA)                LABEL;
03B9. 3    SEEK DECJ01 AT 0;
03BF  3    READ DECJ01 INTO HEADER;
03C6  3    ERRO = 0;
03CC  3    DO I=0 TO 6;                 %DO-J01
03D7  4    IF NOMEDECJ01 (I) <> NOME (I)
03E5  4      THEN  ERRO=1;
03EF  5    END   FIM-DO-J01;            %FIM-DO-J01
03F0  3    IF ERRO=1  THEN  GO TO  SAIABERTURA;
03FC  4    SEEK DECJ15 AT 0;
0402  3    READ DECJ15 INTO HEADERM;
0409  3    DO I=0 TO 6;                 %DO-J15
0414  4    IF NOMEDECJ15 (I) <> NOMEEXTM ( I)
0422  4      THEN ERRO = 2;
042C  5    END FIM-DO-J15;              %FIM-DO-J15
042D  3    IF ERRO = 2 THEN GO TO SAIABERTURA;
0439  4    IF  NGERATABM = TABGERACAO AND MODCOMPRIMTAB = COMPRIMENT  AND
044A  3      BLOCATABM = BLOCAG   THEN GO TO SAIABERTURA;
0459  4    ERRO = 3;
045F  3  SAIABERTURA :
045F  3  END   FIM-ROTABERTURA;         %FIM-ROTABERTURA
0460  2  %-------------------------------------------------------------------------
```

```
04BF 2  %------------------------INICIO  DO  PROCESSO
04BF 2       IF  RCDD = 9 THEN GO TO INICPROCES;
04CB 3       RCDD = 9;
04D1 2       CALL ROTABERTURA;
04D7 2       PONTDISPAUX = PONTDISP;
04DE 2       COMPRIMENTO = COMPRIMENT;
04E5 2       BLOCAGEM = BLOCAG;
04EC 2     IF ERRO = 0 THEN GO TO INICPROCES;
04F8 3     CALL LIMPA80;
04FE 2     MOVE 'HEADERS DOS ARQUIVOS DECCJ01 E DECCJ15 NAO COMPATIVEIS' TO
0536 2        AREA80 (0)  FOR  54;
053F 2     CALL  DUPLOWRITE;
0545 2     EXIT;
0546 2  %----------------------INICIO  DA  INTERACAO
0546 2  INICPROCES :
0546 2     CALL LIMPA80;
054C 2               CALL TRANSCHAVEADD;
0552 2               CALL ROTRECUPERAR(ENDLISTA,ENDULTLIDO);
055E 2               RETURN;
0564 2  %
0564 2  %
```

ANEXO III - PROGRAMA PARA PROCESSAMENTO DE PEDIDOS

```
000A 1  ECCJ60 . DO;
000A 2  %AUTOR-EDUARDO CARNEIRO CAMPELO JUNIOR
000A 2  %       PROGRAMA PARA RECUPERACAO DE REG. MESTRE ATRAVES DA
000A 2  %       FORMULACAO DE EXPRESSAO LOGICA.
000A 2  %*************************************************************
000A 2  DECLARE
000D. 2      VETRTB  (192) BYTE GLOBAL INITIAL (193*' '),   % GLOBAL
00CE. 2      ARA133  (132) BYTE GLOBAL INITIAL (133*' '),   % GLOBAL
0153. 2      REGMEST (31)  BYTE,
0173. 2      REGLI   (63)  BYTE,
01B3. 2 %    RESULTANTE (511) BYTE,
01B3. 2      POSFIX  (30)  ADDRESS GLOBAL,                  % GLOBAL
01F1. 2      ARA512  (511) BYTE,
03F1. 2      BLI       ADDRESS   GLOBAL INITIAL (@REGLI),   % GLOBAL
03F3. 2      BII       ADDRESS   GLOBAL INITIAL (@ARA512),  % GLOBAL
03F5. 2      BMT       ADDRESS   GLOBAL INITIAL (@REGMEST), % GLOBAL
03F7. 2      BTB       ADDRESS   GLOBAL INITIAL (@VETRTB),  % GLOBAL
03F9. 2      B133      ADDRESS   GLOBAL INITIAL (@ARA133),  % GLOBAL
03FB. 2      BSRESU    ADDRESS   GLOBAL INITIAL (@ARA512),  % %GLOBAL
03FD. 2      NREG      BASED BSRESU   ADDRESS,
03FD. 2      LCHAVE    ADDRESS,
03FF. 2      (I,K,J,KJ)       ADDRESS,
0407. 2      RETORNO      BYTE,
0408. 2     (INICIO, SALTO)      LABEL;
0408. 2 %-------------------------------------------------------------
0408 2  %-----------------------ARQUIVOS
0408 2  DECLARE
0408. 2      DECJTB  FILE ON 'DISCO'  RECORD  51,
0408. 2      DECJLI  FILE ON 'DISCO'  RECORD  64,
0408. 2      DECJII  FILE ON 'DISCO'  RECORD 512,
0408. 2      DECJ01  FILE ON 'DISCO'  RECORD 512,
0408. 2      DECJ15  FILE 'DECJ20' ON 'DISCO'  RECORD  32,
0408. 2      IMPRES  FILE ON 'IMPRES' RECORD 133,
0408. 2      VIDEO8  FILE ON 'VIDEO'  RECORD 80,
0408. 2      TECLAD  FILE ON 'TECLAS' RECORD 80,
0408. 2      SYSDIR  FILE 'INDICE' ON 'DISCO'  RECORD 20,
0408. 2      DECJRS  FILE ON 'DISCO'  RECORD 512;
0408. 2 %
0408 2  %-------------------PROCEDURES
0408 2  LOAD :   PROCEDURE  (ROTINA,DISCO) BYTE PLTI; END;
0408 2  ECTBMT : PROCEDURE  PLTI; END;
0408 2  ECPEDI : PROCEDURE  PLTI; END;
0408 2  ECRCDD : PROCEDURE (KJ,J) PLTI RESULT(KJ); END;
0408 2  %
0408 2  %-------------------INICIO  PROCESSAMENTO
0408 2      RETORNO = LOAD (@'ECTBMT',@'       ');
0422 2      IF RETORNO <> 0 THEN EXIT;
042C 3      CALL  ECTBMT;              % CARREGA TABELA DEFINICAO MESTR
0432 2      MOVE VETRTB(7) TO LCHAVE FOR 2;
043E 2  INICIO :
043E 2      RETORNO = LOAD(@'ECPEDI',@'       ');
0458 2      CALL  ECPEDI,
045E 2      ARA133(0) = ' ';
0467 2      MOVE ARA133(0) TO ARA133(1) FOR 132; ARA133(5) = '0';
047F 2      MOVE 'ENTIDADES RESULTANTES DO SEU PEDIDO = SE DESEJA RE
047F 2           ' (S)' TO ARA133(7) FOR 65;
```

```
04CB 2       K = NREG;
04D2 2       I = 5;
04D8 2     DO  WHILE K <> 0;
04E1 3       ARA133(I) = (K MOD 10) + '0';
04F2 3       K = K / 10;
04FC 3       I = I - 1;
0506 3     END  DO-WHILE;
0509 2       WRITE IMPRES FROM ARA133;
0510 2       WRITE VIDEO8 FROM ARA133;
0517 2       READ  TECLAD INTO ARA133(1);
0521 2       IF  ARA133(1) = 'S' THEN GO TO INICIO;
0530 3       RETORNO = LOAD(@'ECRCDD',@'      ');
0549 2       I = 2; K = NREG;
0556 2       DO WHILE K <> 0;
055F 3            J = I + BII;
056A 3            IF ARA512(I)= ' ' THEN GO TO SALTO;
057A 4            CALL ECRCDD(KJ,J);
0586 3            IF KJ <> '*' THEN WRITE VIDEO8 FROM REGMEST;
0596 4            SEEK DECJRS AT 0; READ DECJRS INTO ARA512;
05A3 3            K = K - 1;
05AD 3     SALTO .
05AD 3            I = I + LCHAVE;
05B8 3       END  DO-WHILE;
05BB 2       GO TO INICIO;
05BE 2       EXIT;
05BF 2     %
05BF 2     %
05BF 2     %
05BF 2     %
9=BE 2     %
```

```
***   R E F E X   ***        *  V1-M02  *      17/07/80

COMANDO FORNECIDO :
ECCJ60(ECTBMT,ECPEDI,ECRCDD);M

DIAGNOSTICOS :

R-26 - LINKEDICAO COMPLETADA NORMALMENTE

MAPA DE ALOCACAO

MODULO : ECCJ60      (END. CARGA = 4090)

     NOME        TIPO     DISCO       E. C.      E. E.      ENTRY
     ECCJ60      BASE                 4094       4094
     VETRTB      VAR.                 4094       40A1
     ARA133      VAR.                 4094       4162
     POSFIX      VAR.                 4094       4247
     BLI         VAR.                 4094       4485
     BII         VAR.                 4094       4487
     BMT         VAR.                 4094       4489
     BTB         VAR.                 4094       448B
     B133        VAR.                 4094       448D
     BSRESU      VAR.                 4094       448F
     LOAD        ROT.                 4654       4654
     TDIRET      ROT.                 4747       4747
     LINK        ROT.                 4A94       4A94

MODULO : ECTBMT      (END. CARGA = 4AAA)

     NOME        TIPO     DISCO       E. C.      E. E.      ENTRY
     ECTBMT      BASE                 4AAA       4AAA

MODULO : ECPEDI      (END. CARGA = 4AAA)

     NOME        TIPO     DISCO       E. C.      E. E.      ENTRY
     ECPEDI      BASE                 4AAA       4AAA
     ECISLI      ROT.                 5E46       5E46
     ECHDLI      ENT.                 5E46       5EE7       **
     ECRATB      ROT.                 6583       6583
     ECCGAT      ENT.                 6583       6611       **

MODULO : ECRCDD      (END. CARGA = 4AAA)

     NOME        TIPO     DISCO       E. C.      E. E.      ENTRY
     ECRCDD      BASE                 4AAA       4AAA
```

BIBLIOGRAFIA

(1) KNUTH, Donald E. - The Art of computer programming; sorting and searching. Reading, Mass., Addison - Wesley, 1973. 722p. v.3.

(2) KNUTH, Donald E. - The Art of computer programming; fundamental algorithms. Reading, Mass., Addison - Wesley, 1968. 634p. v.1

(3) SUNDGREN, BO - Theory of data bases. New York, Petrocelli/Charter, 1975. 244p.

(4) LEFKOVITZ, David - Data management for on-line systems. New Jersey, Hayden Book, 1974. 289p

(5) LEFKOVITZ, David - File structures for on-line systems. New York, Sparton Books, 1969. 215p.

(6) MARTIN, James - Computer data-base organization. New Jersey, Prentice-Hall, c1975. 558p.

(7) YOURDON, Edward - Design of on-line computer systems. New Jersey, Prentice-Hall, c1972. 608p.

(8) DATE, C.J. - An Introduction to database systems. Reading, Mass., Addison-Wesley, c1975. 336p.

(9) CRUZ, Pedro Nogueira - Desenvolvimento e implementação de um sistema de arquivo em disco para acesso aleatório. Rio de Janeiro, COPPE/UFRJ, 1975. (Tese M. Sc.).

(10) SOUZA, Jano Moreira de - Algoritmos de hashing para problemas específicos. Rio de Janeiro, COPPE/UFRJ, 1978. (Tese M. Sc.)

(11) FAISSOL, Sérgio Zarur - Método de aceso indexado para o terminal inteligente. Rio de Janeiro, COPPE/UFRJ, 1977. (Tese M. Sc.).

(12) CARVALHO, Alfredo Veiga de - Um Sistema conversacional de consulta para artigos de periódicos. Rio de Janeiro, PUC, 1973. (Tese M.Sc.).

(13) SILVA, Eduardo Dória - Recuperação de informação para microprocessadores. Rio de Janeiro, COPPE/UFRJ, 1976. Tese M. Sc.

(14) CURA, José Carlos Vida - Sistema de entrada e saída no terminal Inteligente (TI). Rio de Janeiro, NCE/UFRJ, 1976.

(15) RODRIGUES, Guilherme Chagas - Sistema operacional para Terminal Inteligente (TI). Rio de Janeiro, NCE/UFRJ, 1976.

(16) MELLO, Paulo - Descrição da linguagem PLTI. Rio de Janeiro, NCE/UFRJ, 1979.

(17) INTEL CORPORATION - PL/M Programming Manual. 1975.

(18) DUEIRE, Pedro José Caminha - Projeto de sistema de informação; exemplificado com uma aplicação ao PLANASA, a nível do BNH. Recife, CETEPE, 1978. 81p. (Tese M.Sc., COPPE/UFRJ, 1977).

(19) LANGEFORS, Börge - In: DUEIRE[18] pag. 15.

(20) ANDERSON, Henry & BERRA, P. Bruce - Minimun cost selection of secondary indexes for formatted files. ACM Trans. DB Syst., New York, 2(1): 68-90. Mar. 1977.

(21) CÁRDENAS, A.F. - Analysis and performance of inverted data base strutures. Comm. ACM, New York (5): 253-63, May, 1975. In: ANDERSON[20].

(22) CÁRDENAS, A.F. - Evoluation and selection of file organization; a model and system. Comm. ACM, New York, 16(9):540-8, Sep. 1973.

(23) COMER, Douglas - The Difficulty of optimum index selection. ACM Trans. DB Syst., New York, 3(4): 440-5, Dec. 1978.

(24) LIU, Jane W. S. - Algorithms for parsing search queries in systems with inverted file organiza tion. ACM Trans. DB. Syst. New York, 1(4): 299-316, Dec. 1976.

(25) CASEY, R.G. - Design of tree structures for effi cient querying. Comm. ACM, New York, 16(9): 549-56, Sep. 1973.

(26) LUM, V.Y. & LING, H. - An Optimization problem on the selection of secondary keys. Chicago, Proc. ACM - Amnual Conf. 1971. p.349-356.

(27) SEVERANCE, D.G. & CARLIS, J.V. - A Practical approach to selecting record access paths. Comp. Surveys ACM, New York, 9(4):259-90, Dec. 1977.

(28) SEVERANCE, D.G. - Identifier search mechanisms; a survey and generalized model. Comp. Surveys ACM, New York, 6(3): 175-94, Sep. 1974.

(29) SEVERANCE, Dennis G. & DUHNE, Ricardo - A Practitioner's guide to addressing algorithms. Comm. ACM, New York, 19(6): 314-26: June, 1976.

(30) EISNER, Mark J. & SEVERANCE, Dennis G. - Mathematical Techniques for efficient record segmentation in large shared databases. Journal ACM, New York, 23(4): 619-35, Oct., 1976.

(31) LUM, V.Y.; YUEN, P.S.T.; DODD, M. - Key-to - address transform techniques; a fundamental performance study on large existing formatted files. Comm. ACM, New York, 14(4): 228-39, Apr. 1971.

(32) MAURER, W.D. - An Improved hash code for scatter storage. Comm. ACM, New York, 11(1): 35-8, Jan. 1968.

(33) MAURER, W.D. & LEWIS, T.G. - Hash table methods. Comp. Surverys ACM, New York, 7(1): 5-19, Mar. 1975.

(34) MORRIS, Robert - Scatter storage techniques. Comm. ACM, New York, 11(1): 38-43, Jan. 1968.

(35) MARCH, Salvatore T. & SEVERANCE, Dennis G. - The Determination of efficent record; segmentations and blocking factors for shared data files. ACM Trans. D.B. Syst. New York, 2(3); 279-96, Sep. 1977.

(36) FORSYTHE, Alexandra I. et alii - Ciência de computadores. Rio de Janeiro, Ao Livro Técnico, 1972. Cap. 10, p. 398-411. v.2 (Série Ciência de Computação).

(37) GAUTHIER, Richard & PONTO, Stephen - Designing systems programs. New Jersey, Prentice-Hall, Inc., 1970. 274p.